

Université de Montréal

**Improving Sampling, Optimization and Feature Extraction in
Boltzmann Machines**

par Guillaume Desjardins

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des arts et des sciences
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Décembre, 2013

© Guillaume Desjardins, 2013.



Résumé

L'apprentissage supervisé de réseaux hiérarchiques à grande échelle connaît présentement un succès fulgurant. Malgré cette effervescence, l'apprentissage non-supervisé représente toujours, selon plusieurs chercheurs, un élément clé de l'Intelligence Artificielle, où les agents doivent apprendre à partir d'un nombre potentiellement limité de données. Cette thèse s'inscrit dans cette pensée et aborde divers sujets de recherche liés au problème d'estimation de densité par l'entremise des machines de Boltzmann (BM), modèles graphiques probabilistes au coeur de l'apprentissage profond. Nos contributions touchent les domaines de l'échantillonnage, l'estimation de fonctions de partition, l'optimisation ainsi que l'apprentissage de représentations invariantes.

Cette thèse débute par l'exposition d'un nouvel algorithme d'échantillonnage adaptatif, qui ajuste (de façon automatique) la température des chaînes de Markov sous simulation, afin de maintenir une vitesse de convergence élevée tout au long de l'apprentissage. Lorsqu'utilisé dans le contexte de l'apprentissage par maximum de vraisemblance stochastique (SML), notre algorithme engendre une robustesse accrue face à la sélection du taux d'apprentissage, ainsi qu'une meilleure vitesse de convergence. Nos résultats sont présentés dans le domaine des BMs, mais la méthode est générale et applicable à l'apprentissage de tout modèle probabiliste exploitant l'échantillonnage par chaînes de Markov.

Tandis que le gradient du maximum de vraisemblance peut-être approximé par échantillonnage, l'évaluation de la log-vraisemblance nécessite un estimé de la fonction de partition. Contrairement aux approches traditionnelles qui considèrent un modèle donné comme une boîte noire, nous proposons plutôt d'exploiter la dynamique de l'apprentissage en estimant les *changements successifs* de log-partition encourus à chaque mise à jour des paramètres. Le problème d'estimation est reformulé comme un problème d'inférence similaire au filtre de Kalman, mais sur un graphe bi-dimensionnel, où les dimensions correspondent aux axes du temps et au paramètre de température.

Sur le thème de l'optimisation, nous présentons également un algorithme permettant d'appliquer, de manière efficace, le gradient naturel à des machines de Boltzmann comportant des milliers d'unités. Jusqu'à présent, son adoption était limitée par son haut coût computationnel ainsi que sa demande en mémoire. Notre algorithme, Metric-Free Natural Gradient (MFNG), permet d'éviter le calcul explicite de la matrice d'information de Fisher (et son inverse) en exploitant un solveur linéaire combiné à un produit matrice-vecteur efficace. L'algorithme est prometteur: en terme du nombre d'évaluations de fonctions, MFNG converge plus rapidement

que SML. Son implémentation demeure malheureusement inefficace en temps de calcul.

Ces travaux explorent également les mécanismes sous-jacents à l'apprentissage de représentations invariantes. À cette fin, nous utilisons la famille de machines de Boltzmann restreintes “spike & slab” (ssRBM), que nous modifions afin de pouvoir modéliser des distributions binaires et parcimonieuses. Les variables latentes binaires de la ssRBM peuvent être rendues invariantes à un sous-espace vectoriel, en associant à chacune d'elles, un vecteur de variables latentes continues (dénommées “slabs”). Ceci se traduit par une invariance accrue au niveau de la représentation et un meilleur taux de classification lorsque peu de données étiquetées sont disponibles. Nous terminons cette thèse sur un sujet ambitieux: l'apprentissage de représentations pouvant séparer les facteurs de variations présents dans le signal d'entrée. Nous proposons une solution à base de ssRBM bilinéaire (avec deux groupes de facteurs latents) et formulons le problème comme l'un de “pooling” dans des sous-espaces vectoriels complémentaires.

Mots-clés: réseaux de neurones, apprentissage profond, apprentissage non-supervisé, apprentissage supervisé, machines de Boltzmann, modèles à base d'énergie, Markov Chain Monte Carlo (MCMC), échantillonnage, fonction de partition, gradient naturel, modèle bilinéaire, apprentissage de représentations.



Summary

Despite the current widescale success of deep learning in training large scale hierarchical models through supervised learning, unsupervised learning promises to play a crucial role towards solving general Artificial Intelligence, where agents are expected to learn with little to no supervision. The work presented in this thesis tackles the problem of unsupervised feature learning and density estimation, using a model family at the heart of the deep learning phenomenon: the Boltzmann Machine (BM). We present contributions in the areas of sampling, partition function estimation, optimization and the more general topic of invariant feature learning.

With regards to sampling, we present a novel adaptive parallel tempering method which dynamically adjusts the temperatures under simulation to maintain good mixing in the presence of complex multi-modal distributions. When used in the context of stochastic maximum likelihood (SML) training, the improved ergodicity of our sampler translates to increased robustness to learning rates and faster per epoch convergence. Though our application is limited to BM, our method is general and is applicable to sampling from arbitrary probabilistic models using Markov Chain Monte Carlo (MCMC) techniques. While SML gradients can be estimated via sampling, computing data likelihoods requires an estimate of the partition function. Contrary to previous approaches which consider the model as a black box, we provide an efficient algorithm which instead tracks the change in the log partition function incurred by successive parameter updates. Our algorithm frames this estimation problem as one of filtering performed over a 2D lattice, with one dimension representing time and the other temperature.

On the topic of optimization, our thesis presents a novel algorithm for applying the natural gradient to large scale Boltzmann Machines. Up until now, its application had been constrained by the computational and memory requirements of computing the Fisher Information Matrix (FIM), which is square in the number of parameters. The Metric-Free Natural Gradient algorithm (MFNG) avoids computing the FIM altogether by combining a linear solver with an efficient matrix-vector operation. The method shows promise in that the resulting updates yield faster per-epoch convergence, despite being slower in terms of wall clock time.

Finally, we explore how invariant features can be learnt through modifications to the BM energy function. We study the problem in the context of the spike & slab Restricted Boltzmann Machine (ssRBM), which we extend to handle both binary and sparse input distributions. By associating each spike with several slab variables, latent variables can be made invariant to a rich, high dimensional subspace resulting in increased invariance in the learnt representation. When using

the expected model posterior as input to a classifier, increased invariance translates to improved classification accuracy in the low-label data regime. We conclude by showing a connection between invariance and the more powerful concept of disentangling factors of variation. While invariance can be achieved by pooling over subspaces, disentangling can be achieved by learning multiple complementary views of the same subspace. In particular, we show how this can be achieved using third-order BMs featuring multiplicative interactions between pairs of random variables.

Keywords: neural network, deep learning, unsupervised learning, supervised learning, Boltzmann machines, energy-based models, Markov Chain Monte Carlo, parallel tempering, partition function, natural gradient, bilinear models, feature learning.



Contents

Résumé	ii
Summary	iv
Contents	vi
List of Figures	xi
List of Tables	xii
List of Abbreviations	xiii
Acknowledgements	xv
1 Introduction	1
1.1 Introduction to Machine Learning	3
1.2 Learning the Model Parameters	6
1.2.1 Stochastic Gradient Descent	6
1.2.2 Exploiting Curvature	8
1.3 Supervised Learning	9
1.3.1 Logistic Regression	11
1.3.2 Multilayer Perceptron	11
1.3.3 Convolutional Neural Networks	14
1.4 Unsupervised Learning	15
1.4.1 Latent Variables and Expectation Maximization	16
1.4.2 Variational Methods	18
1.4.3 Directed vs. Undirected Graphical Models	19
1.4.4 Unsupervised Learning for Feature Extraction	21
2 Unsupervised Deep Learning	24
2.1 Boltzmann Machines	24
2.2 Maximum Likelihood Learning	26
2.3 Restricted Boltzmann Machines	29
2.3.1 RBM for Binary Data	29
2.3.2 RBM for Continuous Data	31

2.4	Approximations to Maximum Likelihood and Other Inductive Principles	33
2.4.1	Stochastic Maximum Likelihood	33
2.4.2	The Fast-Weight Effect and FPCD	35
2.4.3	Tempered SML	35
2.4.4	Contrastive Divergence	37
2.4.5	Variational SML	38
2.5	Multi-Layer Models	40
2.5.1	Deep Belief Networks	40
2.5.2	Deep Boltzmann Machines	41
2.6	Higher-Order Models	44
2.6.1	Third-Order Boltzmann Machines	44
2.6.2	Gated RBM	45
2.6.3	Factored Gated RBM	46
2.6.4	Bilatent Models of Data	47
2.7	Estimating the Partition Function	49
2.7.1	Importance Sampling	49
2.7.2	Annealed Importance Sampling	50
2.7.3	Bridge Sampling	52
2.8	Non-probabilistic Generative Models	53
2.8.1	Score Matching	53
2.8.2	Auto-Encoders	54
2.8.3	Auto-Encoders as Generative Models	56
2.8.4	Predictive Sparse Decomposition (PSD)	57
3	Prologue to First Article	59
3.1	Article Details	59
3.2	Context	59
3.3	Contributions	60
3.4	Recent Developments	61
4	Adaptive Parallel Tempering for Stochastic Maximum Likelihood Learning of RBMs	62
4.1	Introduction	62
4.2	SML with Optimized Parallel Tempering	65
4.2.1	Parallel Tempered SML (SML-PT)	65
4.2.2	Return Time and Optimal Temperatures	66
4.2.3	Optimizing \mathcal{T} while Learning	67
4.3	Results and Discussion	68

4.4	Conclusion	71
5	Prologue to Second Article	74
5.1	Article Details	74
5.2	Context	74
5.3	Contributions	75
5.4	Recent Developments	75
6	On Tracking the Partition Function	77
6.1	Introduction	77
6.2	Previous Work	79
6.2.1	Annealed Importance Sampling	80
6.2.2	Bridge Sampling	80
6.3	Training Restricted Boltzmann Machines	81
6.4	Parallel Tempering	82
6.5	Tracking the Partition Function	83
6.5.1	AIS between learning iterations	83
6.5.2	Bridging the parallel tempering temperature gaps	84
6.5.3	Kalman Filtering the Log Partition Function	85
6.5.4	Inference over the latent state	87
6.6	Experimental Results	88
6.6.1	Comparing to Exact Likelihood	88
6.6.2	Comparing to AIS for Large-scale Models	89
6.6.3	Early-Stopping Experiments	90
6.7	Conclusion and Discussion	91
7	Prologue to Third Article	93
7.1	Article Details	93
7.2	Context	93
7.3	Contributions	94
7.4	Recent Developments	94
8	Metric-Free Natural Gradient for Joint-Training of Boltzmann Machines	96
8.1	Introduction	96
8.2	The Natural Gradient	98
8.2.1	Motivation and Derivation	98
8.2.2	Natural Gradient for Boltzmann Machines	99
8.3	Metric-Free Natural Gradient Implementation	101
8.4	Experiments	102


8.5	Discussion and Future Work	104
9	Prologue to Fourth Article	106
9.1	Article Details	106
9.2	Context	106
9.3	Contributions	107
10	The Spike-and-Slab RBM and Extensions to Discrete and Sparse Data Distributions	109
10.1	Introduction	109
10.1.1	The Gaussian RBM as a Model of Images	111
10.2	The Spike-and-Slab RBM	114
10.3	Positive Definite Constraints	118
10.3.1	Constraining Λ	119
10.3.2	Constraining Φ_i	119
10.4	Learning in the ssRBM	120
10.5	Related Models of Conditional Covariance	121
10.5.1	The Mean and Covariance RBM	122
10.5.2	Mean - Product of Student's T-distributions	123
10.5.3	Comparing the ssRBM to the mcRBM, mPoT	123
10.6	Exp. I: ssRBM on Natural Images	124
10.6.1	Classification	125
10.6.2	Model Samples	128
10.7	ssRBM for Discrete Data	129
10.7.1	Model	129
10.7.2	Exp. II: Binary Visible Data	131
10.8	ssRBM for Spike-and-Slab Data	132
10.8.1	Model	133
10.8.2	Exp. III: Spike-and-Slab Data	134
10.9	Subspace Spike-and-Slab RBM	135
10.9.1	Exp. IV: Subspace ssRBM	137
10.10	Conclusion	139
11	Prologue to Fifth Article	141
11.1	Article Details	141
11.2	Context	141
11.3	Contributions	142
11.4	Recent Developments	143
12	Disentangling Factors of Variation via Generative Entangling	145
12.1	Introduction	145
12.2	Learning Invariant Features Versus Learning to Disentangle Features	147

12.3	Higher-order Spike-and-Slab Boltzmann Machines	149
12.3.1	Higher-order Interactions as a Multi-Way Pooling Strategy	153
12.3.2	Variational inference and unsupervised learning	154
12.3.3	The Challenge of Unsupervised Learning to Disentangle	156
12.4	Related Work	156
12.5	Experiments	158
12.5.1	Toy Experiment	158
12.5.2	Toronto Face Dataset	159
12.6	Conclusion	162
A	Variational Inference	163
A.0.1	Variational Lower Bound	163
A.0.2	Euler-Lagrange Equations	164
B	On Tracking The Partition Function	166
B.1	Parallel Tempering Algorithm	166
B.2	Kalman Filter	167
B.3	Simultaneous Tracking and Learning	167
C	Metric-Free Natural Gradient	171
C.0.1	Expected Hessian of $\log Z$ and Fisher Information.	171
C.0.2	Derivation of Equation 8.5	172
	References	173



List of Figures

1.1	Logistic Regression and a Multi-Layer Perceptron	12
1.2	Convolutional Neural Network	15
1.3	Expectation-Maximization	17
1.4	Directed Bayes Networks	19
2.1	Deep Belief Network and Deep Boltzmann Machine	40
2.2	Samples of a Deep Boltzmann Machine on NORB	43
2.3	Graphical Depiction of (Factored) Gated RBM	47
4.4	Adaptive temperatures during SML-APT training	73
6.1	Graphical model capturing the evolution of the log-partition function	86
10.1	GRBM inductive bias	112
10.2	Inpainting using the GRBM vs. spike & slab RBM	113
10.3	Sensitivity analysis of the ssRBM activation function.	117
10.4	ZCA-whitened training data and parameters learnt by the ssRBM	125
10.5	Samples of an ssRBM trained on CIFAR-10	129
10.6	MNIST classification results using the binary-input ssRBM	132
10.7	Filters learnt by the binary-input ssRBM, trained on MNIST	133
10.8	Filters learnt by a subspace ssRBM on MNIST and CIFAR-10	139
12.1	Higher-Order ssRBM energy function	149
B.1	Graphical model capturing the evolution of the log-partition function	168
B.2	Inference equations for tracking the partition function	169



List of Tables

6.1	Estimated likelihoods of misc. UCI datasets, obtained via tracking .	91
10.1	Comparison of ssRBM parametrizations on CIFAR-10 classification	126
10.2	Comparison of ssRBM classification performance vs. state-of-the-art	127
10.3	Classification error obtained by ssDBNs on MNIST.	135
10.4	MNIST classification error for the subspace ssRBM	136
10.5	CIFAR-10 classification error for a fully-connected subspace ssRBM	138
12.1	Classification accuracy of higher-order ssRBM on Toronto Face Dataset	161



List of Abbreviations

AE	Auto-Encoder
AI	Artificial Intelligence
AIS	Annealed Importance Sampling
ANN	Artificial Neural Network
AST	Adaptive Simulated Tempering
BM	Boltzmann Machine
CAE	Contractive Auto-Encoder
CAST	Coupled Adaptive Simulated Tempering
CD	Contrastive Divergence
CG	Conjugate Gradient
CNN	Convolutional Neural Network
cRBM	Covariance Restricted Boltzmann Machine
DAE	Denosing Auto-Encoder
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
FA	Factor Analysis
FPCD	Persistent Contrastive Divergence with fast-weights
FIM	Fisher Information Matrix
G3RBM	Gated 3-rd Order Restricted Boltzmann Machine
GN	Gauss Newton
GRBM	Gaussian Restricted Boltzmann Machine
GSN	Generative Stochastic Network
HF	Hessian Free

HMC	Hybrid Monte Carlo
hossRBM	Higher-Order Spike & Slab RBM
I.I.D	Independent and Identically-Distributed
LIS	Langevin Importance Sampling
MCMC	Markov Chain Monte Carlo
mcRBM	Mean-Covariance Restricted Boltzmann Machine
MFNG	Metric-Free Natural Gradient
MH	Metropolis Hastings
MLE	Maximum Likelihood Estimator
MLP	Multi-Layer Perceptron
mPoT	Mean Product of Student-t
MRF	Markov Random Field
MSE	Mean-Squared Error
NCE	Noise Contrastive Estimation
NLL	Negative Log-Likelihood
PCD	Persistent Contrastive Divergence (also SML)
PSD	Predictive Sparse Decomposition
PT	Parallel Tempering
RBF	Radial Basis Function
RBM	Restricted Boltzmann Machine
RNN	Recurrent Neural Network
SIFT	Scale Invariant Feature Transform
SGD	Stochastic Gradient Descent
SM	Score Matching
SML	Stochastic Maximum Likelihood (also PCD)
SML-PT	SML with Parallel Tempering
SML-APT	SML with Adaptive Parallel Tempering
ssRBM	Spike & Slab Restricted Boltzmann Machine
SVM	Support Vector Machines
TA	Tensor Analyzers
WL	Wang-Landau



Acknowledgements

I am extremely privileged to have pursued my graduate studies within the LISA lab. Through their passion for research, openness and thought provoking discussions, lab members have created a truly motivating and rewarding workplace. I am particularly indebted to my advisors, Yoshua Bengio and Aaron Courville for their mentorship over the years.

I have benefited immensely from their unique but complementary approaches to research, as well as their encyclopedic knowledge of the field. In particular, I would like to thank Aaron for his unique ability to work alongside students to develop intuitions into reality. Many ideas put forward in this thesis would not have been brought to fruition were it not for his guidance and friendship. I am extremely grateful to Yoshua for his trust and support over the years. I can only hope that future students will benefit as much as I did from his willingness to share and discuss ideas, his overall enthusiasm and his knack for grand scale visions and thinking outside the box.

I would also like to thank my friends and colleagues at LISA. You have made the long hours spent in the lab truly enjoyable. I hope to keep in touch, if not collaborate with some of you in the near future. Many thanks as well to the developers of Theano ([Bergstra et al., 2010](#)) and Pylearn2 ([Goodfellow et al., 2013](#)), libraries which were instrumental to the work presented herein.

To my wife and family, thank you for your unending support. None of this would have been possible without you.

Last but not least, I would like to recognize the financial support of the NSERC postgraduate scholarship program, the Canada Research Chairs, CIFAR and the Université de Montréal. We are also grateful for Calcul Québec and Compute Canada for their computational resources.

1

Introduction

Artificial perception is the keystone of Artificial Intelligence (AI). By giving computers the ability to extract meaningful data from signals such as speech or vision, computers will finally gain the ability to perceive, reason and interact with the world. The field of neural networks and *deep learning* has made remarkable headway in the past few years towards achieving this goal. The exponential increase in computational power and the wide availability of data have enabled the end-to-end training of complex systems starting from the raw data and whose parameters are optimized jointly for a given task. This is in stark contrast to the more traditional feature engineering approach which relies on domain experts to craft low-level features, in order to extract relevant information from the signal. Deep learning systems currently in use at Google and Microsoft have dramatically improved the state of the art in speech recognition (Dahl et al., 2010; Hinton et al., 2012), large scale object recognition in natural images (Krizhevsky et al., 2012), machine translation and even drug discovery ⁱ.

While the seminal Deep Learning paper of Hinton et al. (2006) relied on a combination of supervised and unsupervised learning, much of the recent successes of deep learning can be attributed to standard supervised learning. While earlier work in this area had relied on shallow neural networks, i.e. networks having a single non-linearity, today's models comprise several layers of non-linear transformations which act to extract a hierarchical representation of the data. Such models can now be trained successfully due in large part to advances in modeling, in particular the use of novel activation functions more amenable to gradient descent, as well as increasingly powerful forms of regularization.

One drawback of such methods however is their reliance on large amounts of labeled data, which is generally expensive and sometimes even impossible to obtain. In such settings where few labeled examples are available, methods exploiting unsupervised learning remain state-of-the-art, as shown in several recent machine

i. Merck Molecular Activity Challenge. <https://www.kaggle.com/c/MerckActivity>

learning competitions (Goodfellow et al., 2013; Mesnil et al., 2011). Using human perception as a guide, one also wonders whether supervised learning is truly the yellow brick road to solving general machine perception. Perception “in the wild” undoubtedly involves a myriad of tasks to be performed jointly (object detection, position estimation, motion prediction, etc.) and for which it seems infeasible to first have to collect labeled data. Some tasks may even be novel, in that they have to be learnt through very few (if not zero) training examples, again a domain in which unsupervised learning has proven crucial (Salakhutdinov et al., 2010; Richard Socher and Ng, 2013). Finally, generative models of data can also be predicted to play a large role in perception pipelines where the signal is noisy, ambiguous or subject to large occlusions. In this setting, models can use their knowledge of the world to “fill in” missing or occluded information, prior to classification. These observations have greatly motivated the work presented in this thesis, which centers on using Boltzmann machines (BM), a class of probabilistic generative model, for performing feature extraction and density modeling. Each chapter of this thesis thus tackles fundamental issues in learning and using such models: from better sampling and training algorithms, to exploring novel energy formulations that promote invariant feature extraction. The document is structured as follows.

Chapter 1 presents a general purpose introduction to Machine Learning, including areas of supervised and unsupervised learning. A particular focus is placed on the Multi-layer Perceptron, a model at the heart of the deep learning phenomenon.

Chapter 2 narrows the focus to unsupervised deep learning, presenting a comprehensive review of Restricted, Deep and general Boltzmann Machines, their training algorithms along with tools required for their analysis.

Subsequent chapters present work which appeared in either journal, conference proceedings or workshops.

Chapters 3 and 4 present a novel adaptive parallel tempering algorithm, which dynamically optimizes the hyper-parameters of Parallel Tempering (PT) enabling a more robust estimation of the model’s sufficient statistics and consequently, better convergence properties during training.

Chapters 5 and 6 extend this work by exploiting the adaptive tempering framework to dynamically track the partition function during the course of training.

Chapters 7 and 8 While first-order gradient descent is the method of choice for training BMs, this chapter introduces an algorithm belonging to the family of truncated Newton methods, to efficiently train BMs via the natural gradient.

The remaining chapters focus on the topic of feature extraction.

Chapters 9 and 10 present extensions to the Spike & Slab Restricted Boltzmann Machine (ssRBM), a model specifically crafted to capture covariance information in the input. First introduced in [Courville et al. \(2011a\)](#), the ssRBM is extended to binary and sparse data. This journal paper also highlights the importance of subspace pooling in learning invariant representations.

Chapters 11 and 12 conclude this thesis with a more ambitious endeavour: a model which aims to disentangle the factors of variation present in the data, via multiplicative interactions of latent variables.

1.1 Introduction to Machine Learning

The goal of Machine Learning (ML) is to develop algorithms which learn to model statistical regularities in data, such that it can apply this knowledge to novel situations. An ML algorithm thus takes as input a set of training data \mathcal{D} and outputs a function $f^* \in \mathcal{F}$, which is optimal in some respect. The fitness of a function to model or make predictions on \mathcal{D} , is encoded in a task-specific loss-function \mathcal{L} . The optimal function f^* can then be defined as the function which minimizes the average loss on the training set, a procedure known as Empirical Risk Minimization. Mathematically, this translates to:

$$f_{ERM}^* \leftarrow \arg \min_{f \in \mathcal{F}} \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathcal{L}(x, f) := \arg \min_{f \in \mathcal{F}} \mathcal{R}(\mathcal{D}, f), \quad (1.1)$$

where $\mathcal{R}(\mathcal{D}, f)$ is a short-hand notation for the empirical risk. When working with probabilistic models, ERM is equivalent to the *maximum likelihood learning* principle, which aims to pick the model parameters θ which maximize the probability of the training data under the model distribution.

While learning consists in minimizing the empirical risk on a fixed training set, the true quantity we care to minimize is $\mathbb{E}_\pi[\mathcal{L}(x, f)]$, i.e. expected loss under the complete empirical distribution $\pi(x)$, with $x \sim \pi(x)$, also known as the *generalization error*. $\mathcal{R}(\mathcal{D}, f)$ is however a biased estimator of this quantity. Given a sufficiently wide family of functions \mathcal{F} , we can always minimize $\mathcal{R}(\mathcal{D}, f)$ to an arbitrarily small value. To obtain an unbiased estimator of the generalization error, we therefore resort to using a separate held-out set called the test set $\mathcal{D}_{\text{test}}$ and monitor $\mathcal{R}(\mathcal{D}_{\text{test}}, f)$ during training. The model is said to be *overfitting* when minimizing Eq. 1.1 leads to an increase in $\mathcal{R}(\mathcal{D}_{\text{test}}, f)$.

Regularization

There are two basic strategies for preventing overfitting: (1) we can pick \mathcal{F} to be small enough, such that overfitting is close to impossible or (2) we can maintain \mathcal{F} as is, but add a regularization term $\Omega(f)$ to the cost function of Eq. (1.1) such that complex functions f are penalized. The regularized optimization objective then becomes:

$$f^* \leftarrow \arg \min_{f \in \mathcal{F}} [\mathcal{R}(\mathcal{D}, f) + \lambda \Omega(f)], \quad (1.2)$$

where λ controls the relative weight of the regularization term with respect to \mathcal{R} . Since λ cannot be optimized through direct minimization of Eq. (1.2) (it would trivially lead to $\lambda^* = 0$), it is referred to as a hyper-parameter. Typical algorithms can have dozens of hyper-parameters, which are optimized separately on a held-out dataset $\mathcal{D}_{\text{valid}}$, called the validation set. The process which picks optimal values for f and all associated hyper-parameters is referred to as *model selection*. It is formalized below in the case of a single hyper-parameter $\lambda \in \Lambda$, where Λ represents

a (finite) set of candidate values.

$$\{f_{\lambda_i}^*; \lambda_i \in \Lambda\} \leftarrow \{\arg \min_{f \in \mathcal{F}} [\mathcal{R}(\mathcal{D}, f) + \lambda_i \Omega(f)]; \lambda_i \in \Lambda\} \quad (1.3)$$

$$f^* \leftarrow \arg \min_i \{\mathcal{R}(\mathcal{D}_{\text{valid}}, f_{\lambda_i}^*) + \lambda_i \Omega(f_{\lambda_i}^*); \lambda_i \in \Lambda\}. \quad (1.4)$$

When dealing with multiple hyper-parameters, one can simply adjust the procedure to consider the Cartesian product of all allowable hyper-parameter values. The optimal hyper-parameters will again be those yielding minimal error on the validation set. This procedure is referred to as *grid search*. Grid search can be very expensive however as it scales exponentially in the number of hyper-parameters. Recently, more efficient model selection algorithms have been proposed including *random search* (Bergstra and Bengio, 2012) and *black-box optimization* methods (Snoek et al., 2012a; Bergstra et al., 2011).

Bias vs. Variance

Define \mathbf{f} to be the solution of the unconstrained minimization problem under the full data distribution, i.e. $\mathbf{f} \leftarrow \arg \min \mathbb{E}_{\pi} [\mathcal{L}(x, f)]$. Empirical risk minimization yields an estimate f_{ERM}^* of \mathbf{f} which is both biased and suffers from variance. The bias of the estimate stems from our choice of model family \mathcal{F} , which may not be rich enough to encompass the true underlying function \mathbf{f} , i.e. $\mathbf{f} \notin \mathcal{F}$. The variance in our estimate on the other hand stems from the finite nature of our training data: different instantiations of the training set $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_i$, would each yield a separate function $f_1^*, f_2^*, \dots, f_i^*$, each having their own generalization error.

Maximum Likelihood vs. Bayesian Learning

In the maximum likelihood framework, learning is seen as a function minimization problem. This results in a single probabilistic model for making predictions on test data, which is subject to overfitting. Bayesian approaches to machine learning avoid this issue by considering the parameters themselves as random variables. Bayes theorem is unsurprisingly at the core of this approach:

$$p(\theta | \mathcal{D}) = \frac{p(\mathcal{D} | \theta) p(\theta)}{p(\mathcal{D})}. \quad (1.5)$$

Here, $p(\theta)$ is a prior distribution on the model parameters. Learning in this setting amounts to *sampling* model parameters from the posterior distribution $p(\theta \mid \mathcal{D})$. This approach, while computationally expensive, is immune to overfitting as predictions are averaged over all possible models, taking into account model complexity and the data likelihood. Alternatively, one can also optimize the posterior directly, a method called *Maximum A Posteriori* (MAP). This approach has interesting connections to maximum likelihood learning, since $\log p(\theta \mid \mathcal{D}) = \log p(\mathcal{D} \mid \theta) + \log p(\theta) + \text{cte}$ follows the form of Eq. 1.2, with $\log p(\theta)$ acting as the regularization term.

Non-parametric Models

In this thesis, we only consider parametric forms of \mathcal{F} , such that $\mathcal{F} = \{f(x; \theta) : \theta \in \Theta\}$. There are however non-parametric approaches to machine learning, which make no inherent assumptions about the class of function we wish to learn. Such algorithms often use the training data itself to model the distribution, relying on notions of proximity to generalize. The quintessential non-parametric classifier is the Nearest-Neighbor algorithm (Bishop, 2006) which predicts the label y' associated with test example x' , as a function of the labels of neighboring training samples (according to some distance metric). These methods however suffer from the *curse of dimensionality*ⁱ and their inability to generalize non-locally.

1.2 Learning the Model Parameters

This section describes the most popular methods for performing the minimization problem of Eq. 1.2, which exploit first and second order derivatives of the loss function.

1.2.1 Stochastic Gradient Descent

Assuming the loss $\mathcal{L}(\theta)$ is differentiable, with $\theta \in \mathbb{R}^N$, first-order gradient descent (GD) provides a simple way to minimize Eq. (1.2). GD is an iterative al-

i. The curse of dimensionality states that the number of training examples must grow exponentially with the dimensionality of the data manifold.

gorithm which sequentially updates the parameters θ^t , by taking a small step in the opposite direction of the gradient. This gradient is given by the vector of partial derivatives $J = [\frac{\partial \mathcal{L}}{\partial \theta_1}, \dots, \frac{\partial \mathcal{L}}{\partial \theta_d}]$ evaluated at θ^t , while the length of the step is controlled by a learning rate hyper-parameter ϵ .

Algorithm 1 Batch Gradient Descent Algorithm

```

Initialize model to  $f(x; \theta^0)$ 
while stopping condition is not met do
   $\delta\theta \leftarrow 0$ 
  for all  $z \in \mathcal{D}$  do:  $\delta\theta \leftarrow \delta\theta + \frac{\partial \mathcal{L}(z; f(z; \theta_t))}{\partial \theta} \Big|_{\theta = \theta^t}$ 
   $\theta^{t+1} \leftarrow \theta^t - \epsilon \delta\theta$ ;  $t \leftarrow t + 1$ 
end while
return  $f(x; \theta_t)$ 

```

Formally, for a given value of the parameters θ , the gradient descent direction ∇_{GD} can be formalized as the solution to the following constrained minimization problem.

$$\begin{aligned} \nabla_{GD} &\leftarrow \operatorname{argmin}_{\Delta\theta} \mathbb{E}_{\pi} [\mathcal{L}(f(x; \theta + \Delta\theta))] \\ \text{s.t. } &(\Delta\theta)^T (\Delta\theta) = cte. \end{aligned} \quad (1.6)$$

In other words, we are looking for the optimal direction which minimizes our expected loss while ensuring *constant progress* in parameter space (i.e. the norm of our parameter update is constant). In first-order gradient descent, the above problem is solved by replacing the objective function by its first-order Taylor approximation, and rewriting the constraint as a Lagrangian. This results in a function $\kappa(\Delta\theta)$, which is a second-order polynomial of $\Delta\theta$:

$$\kappa(\Delta\theta) = cte + \mathbb{E}_{\pi} [J] \Delta\theta + \lambda \Delta\theta^T \Delta\theta, \quad (1.7)$$

∇_{GD} is obtained by solving the equation $\frac{\partial \kappa}{\partial \Delta\theta} = 0$ for $\Delta\theta$:

$$\nabla_{GD} = -\epsilon \cdot \mathbb{E}_{\pi} [J]^T, \quad (1.8)$$

with learning rate $\epsilon = -1/(2\lambda)$.

∇_{GD} can be approximated in a number of ways. Algorithm 1 shows a particular

variant of GD called *Batch Gradient Descent* (BGD), which approximates the expectation with an empirical average over the complete training set. Given certain conditionsⁱ on the learning rate ϵ , this procedure is guaranteed to converge to the global minimum if \mathcal{L} is convex, or otherwise, to a local minimum.

An alternative learning algorithm, known as *Stochastic Gradient Descent* (SGD) has proven very efficient in practice (LeCun et al., 1998). Its computational efficiency stems from a finite sample approximation to the expectation \mathbb{E}_π . While this does not technically follow the gradient of $\mathcal{R}(\mathcal{D}, f_\theta)$, SGD converges much faster in practice due to the redundancies in the data. The randomness introduced by the stochastic updates has also been shown to help with escaping from local minima. In theory, SGD approximates the expected loss-derivative with a single point estimate. The resulting optimization procedure can sometimes suffer from excessive variance in this estimate. *SGD with mini-batches* addresses this issue by estimating the gradient on a small group of training examples $\mathcal{X} = \{x^{(n)} \in \mathcal{D}; 1 \leq n \leq N\}$, called a *mini-batch*.

Despite being limited to a linear convergence rate, several extensions to gradient descent can offer faster convergence, under some conditions. The most popular of these methods is probably *gradient descent with momentum* also known as the *Heavy-Ball method* (Nocedal and Wright, 2006). Its update direction is given by interpolating the (latest) first-order gradient with the previous update direction:

$$\theta_{t+1} = \theta_t - \epsilon_1 \mathbb{E}_\pi [J]^T + \epsilon_2 (\theta_t - \theta_{t-1}). \quad (1.9)$$

1.2.2 Exploiting Curvature

The gradient descent direction achieves a linear rate of convergence by approximating the loss function around θ_t by its first-order Taylor approximation. In theory, faster rates can be achieved by exploiting higher-order terms of the Taylor approximation. *Newton's Method* starts from the constrained minimization problem of Eq. 1.6 and replaces the loss function by a second-order Taylor expansion. The Newton direction is then obtained by minimizing the following function:

$$\kappa(\Delta\theta) = \epsilon \Delta\theta^T \mathbb{E}_\pi [J] \Delta\theta + \frac{1}{2} \Delta\theta^T \mathbb{E}_\pi [H] \Delta\theta + \epsilon \Delta\theta^T \Delta\theta.$$

i. The learning rate should have a decreasing profile. Denoting the learning rate at time t as ϵ_t , GD will converge to a global or local minimum if $\lim_{t \rightarrow \infty} \sum_t \epsilon_t = \infty$ and $\lim_{t \rightarrow \infty} \sum_t \epsilon_t^2 < \infty$.

Here $H := \left[\frac{\partial^2 \mathcal{L}_\theta}{\partial \theta^2} \right]$ is the Hessian of the loss function, a matrix square in the number of parameters N . The Newton direction is again obtained by minimizing $\kappa(\Delta\theta)$, yielding:

$$\nabla_N = - (H + \epsilon)^{-1} \cdot \mathbb{E}_\pi [J]^T \quad (1.10)$$

While Newton's method achieved a quadratic rate of convergence, each evaluation of the gradient is more computationally intensive as it requires inverting an $N \times N$ matrix. Many algorithms have been developed which trade-off approximating ∇_N for computational efficiency. For example, *truncated Newton* methods often run only a few iterations of the *Conjugate Gradient* algorithm to obtain a coarse approximation to the solution of the linear system $Hx = -\mathbb{E}_\pi [J]^T$. *Quasi-Newton* methods on the other hand avoid computing H directly. They exploit temporal smoothness in consecutive values of H_t : new estimates can then be obtained via low-rank updates of first-order gradients. The most popular method in this family is probably the (L)BFGS method (Byrd et al., 1995).

For a complete review of optimization methods, we refer the reader to the book of Nocedal and Wright (2006).

1.3 Supervised Learning

We have so far avoided discussing the particular form of the loss-function \mathcal{L} . This is because the choice of \mathcal{L} is tightly linked to the task being considered. In *supervised learning*, the goal is to learn a mapping between observations $x \in \mathcal{R}^D$ and an associated label y . The dataset \mathcal{D} thus contains pairs of observations (x, y) such that $\mathcal{D} = \{(x^{(i)}, y^{(i)}) : 1 \leq i \leq N\}$. We consider the pairs $(x^{(i)}, y^{(i)})$ as being i.i.d. samples of the true distribution $\pi(x, y)$.

Classification

When the task involves classifying x into one of M distinct categories, y is technically discrete. For practical reasons however, we treat y as a vector using a *one-hot encoding* of its class: y is thus a vector of length M whose entries are all 0,

except for $y_j = 1$ to indicate that x belongs to class j . Given the above notation, the goal is then to learn a model $p_\theta(y|x)$ of the true conditional distribution $\pi(y|x)$. Note that $p_\theta(y|x)$ is also vector-valued and contains the estimated class membership probabilities $p_\theta(y_j|x)$ of input x . The class predicted by the model is then given by $\arg \max_j p_\theta(y_j|x)$. For binary classification, a popular choice for \mathcal{L} is the *cross-entropy loss* function, defined as:

$$\mathcal{L}_{\text{CE}}((x, y), p_\theta) = -y \log p_\theta(y|x) - (1 - y) \log(1 - p_\theta(y|x)) \quad (1.11)$$

In the case of multi-class classification, this generalizes to:

$$\mathcal{L}_{\text{MCE}}((x, y), p_\theta) = \sum_j -y_j \log p_\theta(y_j|x) \quad (1.12)$$

Regression

In *regression* tasks, the label y is continuous with $y \in \mathcal{R}^M$. An example application would be to predict the position of an object in an image. In this case, the goal is for $f_\theta(x)$ to model certain statistics of the true posterior distribution. Assuming a Gaussian noise model on the observations y , we can learn a model of the conditional mean $\mathbb{E}[\pi(y|x)]$, by minimizing Eq. 1.2 with the *mean-squared loss*:

$$\mathcal{L}_{\text{MSE}}((x, y), p_\theta) = (f_\theta(x) - y)^2 \quad (1.13)$$

Probabilistic Interpretation

While the choice of these loss functions may appear arbitrary, their particular form can be easily derived from a probabilistic perspective. Binary classification can be thought of as modeling the target or class label y as a Bernoulli distributed random variable, i.e. $y \sim \mathcal{B}(y; p) = p^y(1 - p)^{1-y}$. Instead of parametrizing and learning p directly however, we instead define $p := f(x; \theta)$ and learn the parameters θ of some function f . Minimizing Eq. 1.11, then corresponds to maximizing the probability of the empirical distribution under the model defined by $f_\theta(x; \theta)$. For a classification task involving M possible targets, y is simply modeled as being a multinomial distribution. Similarly, the mean-squared error loss typically used in regression stems from modeling $y \sim \mathcal{N}(y; f(x; \theta), \sigma^2)$, i.e. a normal distribution with mean $f(x; \theta)$ and standard deviation σ .

1.3.1 Logistic Regression

The simplest and most popular supervised learning module is undoubtedly (binary) Logistic Regression, shown in Figure 1.1 (left). It is a directed graphical model which learns the conditional distribution $p(y | x)$, i.e. the predictive distribution of the class label $y \in \{0, 1\}$ given an input $x \in \mathbb{R}^D$. Mathematically, it is defined as follows:

$$p_{\theta}(y | x) = \text{sigmoid}(c + \sum_i W_i x_i) = \text{sigmoid}(a(x)). \quad (1.14)$$

Logistic regression belongs to the class of *linear classifiers* as it carves the input space \mathcal{R}^D in two mutually exclusive regions, using a *linear decision boundary*. This boundary is defined by the hyper-plane with tangent vector W and offset c . Concretely, the class conditional probability is obtained in two steps. First, the linear mapping $a(x)$ can be interpreted as computing a distance between point x and the separating hyper-plane (with the sign reflecting class membership). The sigmoid function, defined as:

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}, \quad (1.15)$$

then squashes this distance to the unit interval, in order for it to be interpretable as a probability measure. It is referred to as an *activation function*. Optimizing the model parameters $\theta = [c, W]$ using the cross-entropy loss, yields a convex objective which can be minimized analytically. Logistic regression can be easily adapted to the multi-class setting, where $y \in \{0, 1\}^M$ by using a weight-matrix $W \in \mathbb{R}^{D \times M}$, bias vector $c \in \mathbb{R}^M$ and the softmax activation function, which given an input vector x yields a vector whose i -th entry is given by:

$$[\text{softmax}(x)]_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (1.16)$$

1.3.2 Multilayer Perceptron

Multi-Layer Perceptrons (MLP) or Artificial Neural Networks are the non-linear extensions of Logistic Regression. While they can be used both for regression and classification; we again focus on the latter. A single hidden-layer MLP models the

class conditional as follows:

$$h(x) = \text{sigmoid}(c + \sum_i^D W_{ij}^{(1)} x_i) := \text{sigmoid}(a^{(1)}(x)) \quad (1.17)$$

$$p_\theta(y | x) = \text{sigmoid}(b + \sum_j^{N_h} W_{kj}^{(2)} h_j) := \text{sigmoid}(a^{(2)}(h)). \quad (1.18)$$

A graphical depiction of the model is shown in Figure 1.1 (right). The reader should recognize Eq. (1.17) as N_h independent logistic regression classifiers, which implements a non-linear mapping of the data distribution from \mathcal{R}^D to \mathcal{R}^{N_h} . Each hidden unit h_j represents the probability that the j -th binary feature (encoded by the vector $W^{(1)T}_j$) is present in x . Equation (1.18) then stacks a second layer of logistic regression on top of this hidden representation, with a weight matrix $W^{(2)} \in \mathcal{R}^{N_h \times 1}$. This layer performs a second non-linear mapping from \mathcal{R}^{N_h} to \mathcal{R} which implements a linear classification boundary in the space of hidden variables. The mapping performed by the hidden layer allows the MLP to implement a powerful *non-linear decision boundary*. By scaling the number of hidden units, MLPs can actually implement arbitrarily complex decision boundaries, a property known as *universal approximation* (Hornik et al., 1989). This comes at a price however: the loss function is no longer convex meaning that MLPs suffer from problems of local minima during optimization.

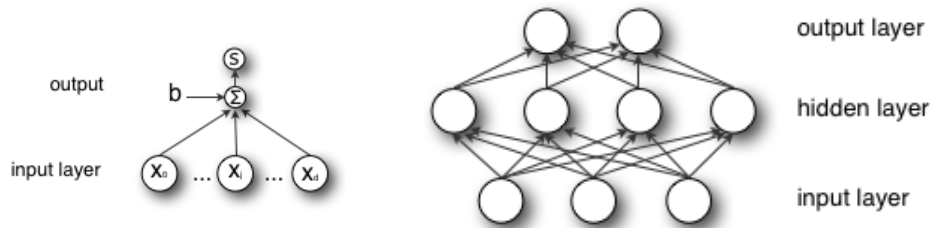


Figure 1.1: (left) Graphical Representation of Logistic Regression. Directed connections from x_i to the summation node represent the weighted contributions $W_{ij}x_i$. s represents the sigmoid activation function. (right) Multi-Layer Perceptron. Each unit in the hidden layer represents a logistic regression classifier. The hidden layer then forms the input to another stage of logistic classifiers.

Backpropagation

The parameters $\theta = [c, W^{(1)}, b, W^{(2)}]$ of the MLP can be learnt by gradient descent on the cross-entropy loss function \mathcal{L}_{CE} . Gradients $\frac{\partial \mathcal{L}_{CE}}{\partial \theta}$ can be computed efficiently through an algorithm called *backpropagation* (Rumelhart et al., 1986), which applies dynamic programming to the chain rule of derivation. The procedure starts by computing the gradient on \mathcal{L}_{CE} with respect to the linear output units o_k , defined as $\frac{\partial \mathcal{L}_{CE}}{\partial a_k^{(2)}} = \frac{\partial \mathcal{L}_{CE}}{\partial y_k} \frac{\partial y_k}{\partial a_k^{(2)}}$. The gradients on the weights of each layer are then obtained as follows (gradients on the biases can be obtained in a similar manner):

$$\frac{\partial \mathcal{L}_{CE}}{\partial W_{kj}^{(2)}} = \frac{\partial \mathcal{L}_{CE}}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial W_{kj}^{(2)}}. \quad (1.19)$$

$$\frac{\partial \mathcal{L}_{CE}}{\partial W_{ij}^{(1)}} = \sum_k \left(\frac{\partial \mathcal{L}_{CE}}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial h_j} \right) \frac{\partial h_j}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial W_{ij}^{(1)}}. \quad (1.20)$$

SGD is the method of choice for optimizing MLPs, with the number of hidden units N_h and learning rate ϵ being treated as hyper-parameters.

Ill-conditioning of Deep Networks

SGD can break-down however for deep networks consisting of a cascade of non-linear functions $h^{(l)}(x) : \mathbb{R}^{m-1} \rightarrow \mathbb{R}^m$, for $l \in [1, L]$. By extending the derivation of Eq. 1.20 to the multi-layer setting, we can write the gradients on the parameters of layer l in vector format, as an expression involving products of jacobians:

$$\frac{\partial \mathcal{L}_{CE}}{\partial \theta^{(l)}} = \frac{\partial \mathcal{L}_{CE}}{\partial h^{(L)}} \left[\prod_{\ell=l}^{L-1} \frac{\partial h^{(\ell+1)}}{\partial h^{(\ell)}} \right] \frac{\partial h^{(l)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial \theta^{(l)}}. \quad (1.21)$$

This is problematic in that, for deep networks (with large L), the gradients on the parameters of layer one involve products of L jacobians. Depending on the eigenvalues of these matrices, this product can be extremely ill-behaved leading to the famous *vanishing or exploding gradient* problem, first analyzed in Hochreiter (1991) and independently by Bengio et al. (1994) in the case of Recurrent Neural Networks (RNN). These issues can be mitigated with proper initialization of the network parameters (Glorot and Bengio, 2010; Sutskever et al., 2013); alternative activation functions such as the hyperbolic tangent, *Rectified Linear Unit* (ReLU)

(Nair and Hinton, 2010; Glorot et al., 2011), softsign (Bergstra et al., 2009) or max non-linearity (Goodfellow et al., 2013). Alternatively, second-order methods have also been shown to help with the ill-conditioning of deep networks, either through learning rate adaptation schemes which incorporate curvature information (Schaul et al., 2012), momentum (Sutskever et al., 2013) or second-order optimization methods such as Hessian-Free optimization (Martens, 2010).

Regularization

Because MLPs are non-linear classifiers, great care must be taken to avoid over-fitting. One can thus employ L1ⁱ or L2 regularizationⁱⁱ, which penalizes the norms of the parameters. *Early-stopping* can also be used to avoid over-fitting: after each parameter update, we monitor the performance on the validation set and stop the optimization process when $\mathcal{R}(\mathcal{D}_{\text{valid}}, p_\theta)$ becomes worse than $\mathcal{R}(\mathcal{D}, p_\theta)$. Recently, *Dropout* (Hinton et al., 2012) in combination with Rectified Linear units (ReLU) has proven to be a much more effective form of regularization and has led to state-of-the-art results on a number of benchmark tasks. At training time, dropout multiplies each hidden unit with a binary random variable, effectively preventing the zeroed-out units from contributing to the network’s prediction. This procedure effectively trains an exponential number of models with shared parameters, each having their own connectivity pattern. At test time, one can recover the average prediction (across the exponential mixture of MLPs) by a simple feed-forward pass and a linear rescaling of the parameters.

1.3.3 Convolutional Neural Networks

Another member of the MLP family worth mentioning is the *Convolutional Neural Network* (CNN) (LeCun and Bengio, 1994), which is employed in Chapter 10. CNNs replace the dot-product in the activation function of the i -th layer by a convolution operation. The hidden layer of a CNN is thus given by the vector $h(x)$, with entries $h_j(x) = \text{sigmoid}(c + W_j^T * x)$, where $*$ represents the convolution operation. Fig. 1.2 shows a CNN applied to a black & white image.

When the input x is a 2D image, the convolutional operator leads to each filter

-
- i. L1 regularization is defined as $\Omega(f_\theta) = \sum_i |\theta_i|$
 - ii. L2 regularization is defined as $\Omega(f_\theta) = \sum_i |\theta_i|^2$

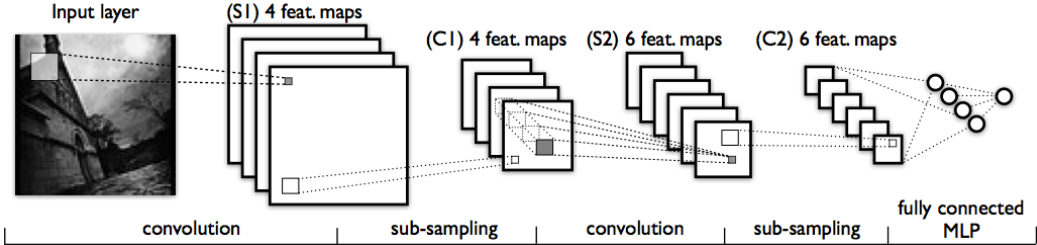


Figure 1.2: An example of a convolutional neural network, similar to LeNet-5. The CNN alternates convolutional and sub-sampling layers.

W_j^T generating a 2D-lattice of activations called a *feature map*. These represent filtered versions of the input. One can think of all the “pixels” in a feature map as individual hidden units, which share parameters and are connected to a subset of the input called the *receptive field*. The size of this receptive field is directly controlled by the dimension of the filter W_j^T .

The main advantage of convolutional architectures is their scalability: their number of parameters $|\theta|$ does not necessarily increase with the size of the input image. This makes learning CNNs statistically efficient, leading to competitive results across many datasets (Krizhevsky, 2010; Le et al., 2010; Jarrett et al., 2009). CNNs are also naturally equivariant: given a translation operator T , a translated input $T(x)$ leads to a translated hidden representation $T(h(x))$. By introducing *max-pooling units* (which subsample each filter map by preserving only the maximal value in their receptive field), CNNs become invariant to small translations in the input. Reminiscent of dropout, introducing stochasticity in the max-pooling operation has been shown to greatly improve generalization performance (Zeiler and Fergus, 2013) of convolutional networks.

1.4 Unsupervised Learning

The principle behind unsupervised learning is to extract the underlying structure in the empirical distribution. The training data thus consists of a set of unlabeled examples $\mathcal{D} = \{x^{(i)} : 1 \leq i \leq N\}$, with $x^{(i)} \sim \pi(x)$. Unsupervised learning encompasses a wide range of models and applications. In *clustering* algorithms such

as k-Means (Lloyd, 1982), the goal is to associate each example $x^{(i)}$ to a centroid, which serves to summarize local statistics of the distribution. In *dimensionality reduction*, one seeks to learn a mapping from a D to an N -dimensional space, with $N \ll D$, which preserves as much information about the input distribution as possible. Popular algorithms in this family include Principal Component Analysis (PCA), Independent Component Analysis (Comon, 1994) as well as non-linear dimensionality reduction techniques such as Local Linear Embedding (Roweis and Saul, 2000) and t-SNE (van der Maaten and Hinton, 2008).

In this work, we are predominantly interested in the problem of *density estimation*, which seeks to learn a model $p_\theta(x)$ which minimizes $\text{KL}(p_\theta \| q)$. The most popular loss function in this context is the *negative log-likelihood loss*.

$$\mathcal{L}_{NLL}(x, p_\theta) = -\log p_\theta(x) \tag{1.22}$$

1.4.1 Latent Variables and Expectation Maximization

We are particularly interested in models which capture statistics of the input distribution through *latent variables*, i.e. unobserved quantities which help to explain key properties in the input. Such models define a joint-distribution $p_\theta(x, h)$ over the observation vector x and a vector h of latent variables. For simple models where the latent variables can be marginalized analytically (e.g. the Restricted Boltzmann Machine, Section 2.3.1), one can learn the parameters θ through maximum likelihood of the marginal distribution $p(x) = \sum_h p_\theta(x, h)$. When latent variables are binary however, this marginalization can often require a sum which is exponential in the number of latent variables. In this setting and when the posterior distribution $p_\theta(h | x)$ is tractable, the maximum likelihood learning rule must be adapted slightly, in a procedure known as *Expectation-Maximization* (EM).

Instead of optimizing the (log) marginal distribution directly, the EM algorithm iteratively optimizes the expected logarithm of the joint distribution $p_\theta(x, h)$, under the model's posterior. Somewhat surprisingly, both of these iterative gradient-based methods can be shown to yield the same fixed points, i.e. local maxima of the likelihood function. The crux of the proof relies on decomposing the log-likelihood

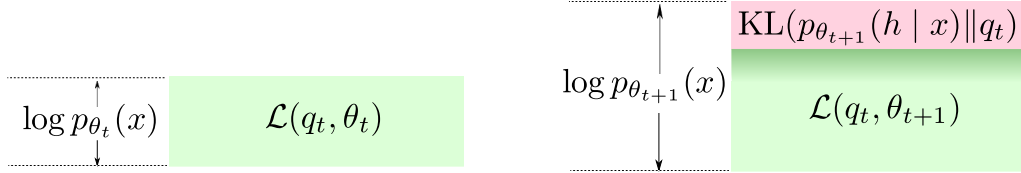


Figure 1.3: Expectation-Maximization. See text for details.

into a lower-bound $\mathcal{L}(q, \theta)$ and a positive KL divergence term, a decomposition which holds for any choice of density $q(h)$. See Appendix A for a detailed derivation.

$$\begin{aligned} \log p(x) &= \mathcal{L}(q, \theta) + \text{KL}(q \| p) & (1.23) \\ \mathcal{L}(q, \theta) &= \sum_h q(h) \log \left\{ \frac{p(x, h; \theta)}{q(h)} \right\} \\ \text{KL}(q \| p) &= - \sum_h q(h) \log \left\{ \frac{p(h | x; \theta)}{q(h)} \right\} \end{aligned}$$

Sketch of Proof. Starting from the model parameters θ_t , we choose q_t to be the exact model posterior, causing the KL divergence term to vanish and yielding $\mathcal{L}(q_t, \theta_t) = \log p(x; \theta_t)$. Since parameters θ_{t+1} are obtained via gradient ascent of the functional \mathcal{L} , we can also write $\mathcal{L}(q_t, \theta_t) \leq \mathcal{L}(q_t, \theta_{t+1})$, with equality holding only if the procedure has already converged to a local maxima. Finally, since the lower-bound interpretation holds for any $q(h)$ and since $\text{KL}(q_t \| p(h | x; \theta_{t+1})) > 0$ (q_t is no longer the true model posterior for parameters θ_{t+1}), we also know that $\mathcal{L}(q_t, \theta_{t+1}) < \log p(x; \theta_{t+1})$.

Putting all this together, we can write

$$\log p(x; \theta_t) = \mathcal{L}(q_t, \theta_t) \leq \mathcal{L}(q_t, \theta_{t+1}) < \log p(x; \theta_{t+1}).$$

□

For a more thorough treatment of the EM algorithm, we refer the reader to Bishop (2006).

1.4.2 Variational Methods

The EM algorithm can be applied to models having an intractable posterior distribution by using the framework of variational inference (Bishop, 2006). In this case, the E-step aims to find a tractable distribution q which approximates the true model posterior. Since q is only an approximation however, the E-step does not lead to a tight lower-bound since $KL(q||p(h||x)) > 0$. As such, the M-step (which optimizes the lower-bound wrt. the model parameters) is no longer guaranteed to improve the log-probability of the model on every iteration. Despite this lack of theoretical guarantees, variational learning methods have been shown to work well in practice given the right choice of distribution q .

While picking the right variational distribution can be an art form in and of itself, two popular methods are used in practice. One can pick $q(x;\omega)$ to be part of some parametric family $\mathcal{Q} = \{q(x;\omega) : \omega \in \Omega\}$. The E-step then amounts to finding the optimal parameters ω^* which maximize the lower-bound $\mathcal{L}(q)$ (or alternatively minimize the KL divergence). Another popular method which is used in Chapters 8 and 12 is to constrain q to be factorial, a method known as *mean-field* (Parisi, 1988). Formally, mean-field assumes the following factorized form for the variational distribution: $q(x) = \prod_1^M q_i(x_i)$, i.e. a product of M factors over disjoint sets of random variables. The Euler-Lagrange equation (derived in Appendix A) specifies how to derive, from this constraint, the closed-form analytic solution to each factor q_i .

$$\log q_i(x_i) = \mathbb{E}_{q_{/i}} [\log p(x, h)], \text{ with } q_{/i} := \prod_{j \in [1, M], j \neq i} q_j(x_j). \quad (1.24)$$

Starting from some initial conditions, one iterates over the above equations for each factor q_i . Each such iteration is guaranteed to increase the variational lower-bound. This process is repeated until each q_i converges, at which point the E-step is complete: the lower-bound is as tight as possible given the mean-field approximation. The algorithm then proceeds with the M-step by performing a single step of gradient ascent of $\mathcal{L}(q)$ wrt. the model parameters. We again refer the reader to Bishop (2006); Koller and Friedman (2009) for more details.

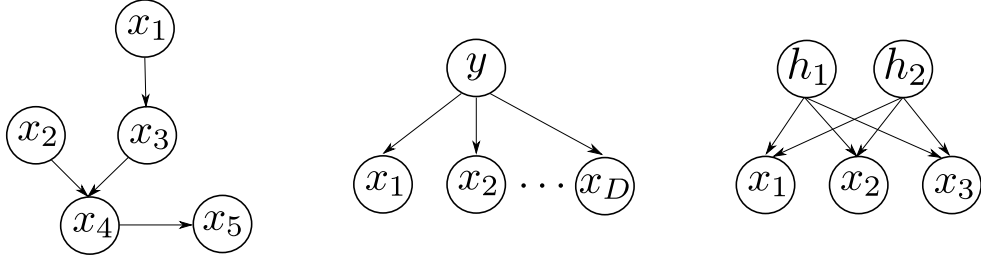


Figure 1.4: Directed Bayes Networks. (left) $p(x_1, \dots, x_5) = p(x_5 | x_4)p(x_4 | x_2, x_3)p(x_3 | x_1)p(x_2)p(x_1)$. (middle) Naive Bayes classifier. (right) Factor Analysis or Sparse Coding.

1.4.3 Directed vs. Undirected Graphical Models

It can be prohibitively expensive to learn a full joint distributions $p_\theta(x)$, as the naive parametrization tends to grow exponentially in the dimensionality of x . One can drastically reduce the number of parameters to learn by exploiting conditional independencies present (or assumed to be present) in the data distribution. The field of *graphical models* (Koller and Friedman, 2009) formalizes these concepts by combining elements of probability and graph theory. A graphical model represents a probability distribution $p_\theta(x)$, $x \in \mathcal{R}^D$ through a graph $\mathcal{G} = (V, E)$, where $V = \{1, \dots, D\}$ is the set of vertices and $E \subset V \times V$ the set of edges. Each vertex V_i represents a random variable x_i , while the *absence of an edge* between variables encodes conditional independencies present in the underlying distribution. The probability mass function is defined as a product of factors, computed over local subsets of the vertices. There are two broad families of graphical models.

Directed Bayes Networks In *Directed Bayes Networks* (BN) or directed graphical models, $p_\theta(x)$ is defined as a product of normalized conditional distributions, where the conditional dependencies are captured via a Directed Acyclic Graph (DAG). Denoting $V_\pi(i)$ to be the set of parents of node V_i (and $x_{\pi(i)}$ the associated random variables), the pdf associated with \mathcal{G} is obtained as:

$$p_\theta(x) = \prod_{i=1}^D p_{\theta_i}(x_i | x_{\mathcal{A}(i)}). \quad (1.25)$$

One should recognize the chain rule of probability, where some factors in the complete expansion $p(x) = p(x_D | x_{D-1}, \dots, x_1)p(x_{D-1} | x_{D-2}, \dots, x_1) \dots p(x_2 |$

$x_1)p(x_1)$ have been zeroed-out due to conditional independencies encoded in the graph structure. Examples of BNs are given in Figure 1.4 for an arbitrary distribution (left) along with the *naive Bayes classifier* (middle) which models the input variables (x_1, x_2, \dots, x_D) as being independent when conditioned on the class label y .

Figure 1.4 (right) shows the graphical model underpinning several popular feature (or dictionary) learning methods, which model the input x as the linear combination of a latent code $h \in \mathbb{R}^N$ with a feature (or dictionary) matrix $W \in \mathbb{R}^{D \times N}$. More precisely, real-valued inputs are modeled as $p(x | h) = \mathcal{N}(x | Wh + \mu, \Psi)$, with mean vector $\mu \in \mathbb{R}^D$ and covariance matrix $\Psi \in \mathbb{R}^{D \times D}$. Setting the prior $p(h)$ to be Gaussian with an isotropic or diagonal covariance matrix Ψ recovers the famous probabilistic PCA (Tipping and Bishop, 1999) and Factor Analysis algorithms (Basilevsky, 1994) respectively. *Sparse Coding* (Olshausen and Field, 1996) models the input as the linear combination of a small subset of basis filters (columns of the weight matrix W). This is achieved by using a sparsity inducing prior on h , such as the Laplace distribution, and setting $N \gg D$. Compared to similar feature learning algorithms presented in Chapter 2, Sparse Coding (and directed models in general) benefits from the property of *explaining away*. While h_i and h_j are independent random variables, they become dependent when conditioning on the input variable x . This is a powerful reasoning mechanism which allows the latent features to “compete” in explaining the input. Unfortunately, this also tends to make inference more complex, often requiring iterative inference algorithms. Sampling in BNs is also very straightforward and can be performed in a single top-down pass in a procedure known as *ancestral sampling*.ⁱ

Markov Random Fields *Undirected graphical models*, also known as *Boltzmann Machines* or *Markov Random Fields* (MRFs) with latent variables are the central topic of this thesis. Since edges are undirected, the decomposition $p_\theta(x)$ cannot rely on a topological ordering of the graph, but instead decomposes the pdf as a product of factors defined over *cliques* of the graph, i.e. subset of vertices which are fully connected. Denoting x_{C_k} as the variables belonging to the k -th clique, the

i. To sample from the model of Fig. 1.4 (left), start by sampling from the ancestral nodes, then repeatedly sample from the conditional distributions following a topological ordering of the nodes. The procedure is as follows: $x_1 \sim p(x_1)$, $x_2 \sim p(x_2)$, $x_3 \sim p(x_3 | x_1 = x_1)$, $x_4 \sim p(x_4 | x_3 = x_3, x_2 = x_2)$, $x_5 \sim p(x_5 | x_4 = x_4)$.

pdf can then be written as:

$$p_{\theta}(x) = \frac{1}{Z} \prod_{k=1}^K \psi(x_{c_k}). \quad (1.26)$$

Here ψ are the *potential functions* which are constrained to be positive, and Z is the *partition function* or normalization constant. Popular examples of MRFs include Ising Models, Conditional Random Fields (Lafferty et al., 2001) and the Restricted and Deep Boltzmann Machines which are the subject of Chapter 2.

The formalism of graphical models represents a powerful framework for reasoning about distributions. Tasks such as computing posterior distributions, MAP estimates or marginals can all be made more efficient by exploiting the structure of the graph. The simplest such algorithm, *variable elimination*, exploits conditional independencies to determine the ordering in which variables should be summed (or integrated) out. *Message passing* algorithms exploit dynamic programming to perform these tasks efficiently, by reusing intermediate computations. When \mathcal{G} is tree-structured, *sum-product* and *max-product* algorithms can compute marginals or MAP (respectively) of full or conditional distributions in a time linear in the number of nodes. Message passing algorithms can also be extended to perform exact inference on general acyclic graphs via the *junction tree algorithm* (which is exponential in the treewidth of the graph), or approximate inference in general “loopy” graphs using *loopy belief propagation*. For a more thorough treatment of this material, we refer the reader to Koller and Friedman (2009); Wainwright and Jordan (2008).

1.4.4 Unsupervised Learning for Feature Extraction

The intersection of unsupervised and supervised learning has recently become a dynamic area of research. This was spearheaded by the work of Hinton et al. (2006) which showed that the parameters of Restricted Boltzmann Machines, learnt in an unsupervised manner, could be used as an initialization point for feed-forward classification networks such as MLPs and fine-tuned for the supervised task through the standard backpropagation algorithm. Compared to random initialization, this led to an increased robustness to local minima as well as better generalization performance. This was later analyzed by Erhan et al. (2010), which showed that

unsupervised pre-training could be interpreted as a powerful new form of regularization for traditional MLPs. Since, unsupervised feature learning has also proven crucial to other formalisms such as Self-Taught Learning (Raina et al., 2007) and Transfer Learning (Mesnil et al., 2011) and has spawned an entire field of research, called **Deep Learning**.

At a high-level, the above confirms a very simple intuition: the latent variables which are good at capturing statistical structure in $\pi(x)$ can also be useful for modeling related conditional distributions $\pi(y | x)$. Instead of using the parameters learnt through unsupervised learning to initialize a feed-forward network, one can thus simply treat the model’s posterior distribution $p_\theta(h | x)$ as providing a non-linear mapping from $\mathbb{R}^D \rightarrow \mathbb{R}^{N_h}$ and use the resulting representation as input to a classifier. The following question then comes to mind: how can we encourage representations learnt through unsupervised learning to be useful for auxiliary tasks, such as classification? Is the maximum likelihood learning rule of Eq. 1.22 sufficient to discover such a representation? What are the particular model structures which encourage learning a richer and more diverse set of features? These are the questions which the newly emerging field of *unsupervised feature learning* aims to answer.

Bengio (2009); Wiskott and Sejnowski (2002); Goodfellow et al. (2009); Bengio and Courville (2013) have identified *invariance* as the key property of what constitutes a “good” representation. Ideally, latent factors should capture useful properties of the input (e.g. useful for discriminating between classes) while being invariant to other irrelevant factors of variations. One such example in object categorization tasks is the “contrast” present in natural images. Contrast can greatly impact global statistics, but is completely irrelevant for object identity. Unsurprisingly, preprocessing natural images with Local Contrast Normalization (LCN) (Pinto et al., 2008) (i.e. making the input itself invariant to local changes in contrast) is a well known technique for improving object classification performance in natural images, as is using richer model classes which learn features based on second-order statistics of the input, such as the Spike & Slab RBM (Courville et al., 2011b), the topic of Chapter 10.

Other factors of variation such as object position or identity are inherently more difficult to deal with, as they combine in a very non-linear manner to generate the

image pixels. Furthermore, each one of these factors may be considered relevant, depending on whether the final task is one of object classification or object detection: object recognition system should be invariant to position, while object localization should be invariant to object identity. Bengio (2009) thus proposes **disentangling the factors of variation** as the holy-grail of feature extraction. The resulting representation should factorize such that the variations along a given factor are encoded in a subset of the latent units, which are themselves invariant to all other factors of variation. The Bilinear RBM proposed in Chapter 12 is our attempt at tackling this very difficult issue.

2

Unsupervised Deep Learning

Hinton et al. (2006), Bengio et al. (2007) and Ranzato et al. (2007) represent a significant breakthrough in the field of neural network research. This work exploited unsupervised learning in order to initialize the parameters of a deep MLP, through a greedy layer-wise pretraining algorithm. Fine-tuning these parameters through backpropagation led to state of the art results and pioneered a new field of research. Deep Learning combines two core concepts: leveraging unsupervised learning algorithms for feature extraction and using deep hierarchical models which are hypothesized to be more statistically efficient than shallow architectures. This chapter focuses on one of the main building blocks of deep networks: the Restricted Boltzmann Machine.

2.1 Boltzmann Machines

In the remainder of this document, we focus on a particular class of probabilistic generative model called Boltzmann Machines (BM) (Hinton et al., 1984). BMs encode a probability distribution $p(x)$ through an energy function, which assigns low energy to states with high probability, and high energy to less probable states. This energy is a function of model parameters θ and random variables which can either be observed (visible) or latent (hidden). The parameters of the model serve to encode relationships between these various random variables, with latent variables acting as explanatory factors of the data. Formally, the energy of the visible units v and hidden units h is denoted by $E(v, h; \theta)$ or equivalently $E_\theta(v, h)$ ⁱ.

i. Depending on context, we may also drop the dependence on θ from the notation altogether and write $E(v, h)$.

Energy is converted to probability through the Gibbs distribution:

$$p(v, h) = \frac{1}{Z} \exp(-E_\theta(v, h)) \quad (2.1)$$

where $Z = \sum_{v,h} \exp(-E_\theta(v, h))$ is the finite partition function, which ensures that $p(v, h)$ is normalized. We can recover the marginal distribution $p(v)$ by summing over the latent variables:

$$p(v) = \frac{1}{Z} \sum_h \exp(-E_\theta(v, h)). \quad (2.2)$$

It is also useful to define the Free-Energy function:

$$F_\theta(v) = -\log \sum_h \exp(-E_\theta(v, h)), \quad (2.3)$$

which allows us to write $p(v) = \frac{1}{Z} \exp(-F_\theta(v))$.

The prototypical Boltzmann machine energy function is defined over a set of binary random variables $x \in \{0, 1\}^n$, a concatenation of the vector of visible and hidden variables. It is defined as:

$$E_{BM}(x) = -\frac{1}{2} x^T W x - b^T x. \quad (2.4)$$

Here the parameters of the model are the symmetric weight matrix $W \in \mathbb{R}^{n \times n}$ and bias vector $b \in \mathbb{R}^n$. Denoting x_{-i} as the vector x with the i -th entry skipped, the above energy function inducesⁱ the following conditional distribution:

$$p(x_i | x_{-i}) = \text{sigmoid} \left(\sum_{j \neq i} W_{ij} x_j + b_i \right) \quad (2.5)$$

At a high-level, we can see that each unit in the BM is stochastically activated, with an activation probability which depends on the dot-product between the feature vector $W_{i(-i)}$ (defined as the vector W_i without entry W_{ii}) and the vector of random variables x_{-i} . This feature vector can thus be thought of as expressing a preference for the particular pattern of activations preferred by unit x_i . Repeatedly iterating

ⁱ. We skip the derivation of this specific conditional distribution, in favor of the RBM conditionals of Section 2.3.1.

over the above conditional (for all i) implements the *Gibbs sampling* procedure which can be used to generate samples from either the full model distribution $p(x)$ or from any conditional distributions by clamping a subset of the units. As we shall see in the following section, the goal of learning is then to fit the parameters such that the sufficient statistics of the model (in this case $x_i x_j$ and x_i) are approximately equal (on average) when the model is left “free-running” (with no units clamped) or when a subset of the units are clamped to training data.

Relationship to Other Models

Boltzmann Machines are also closely related to Markov Random Field and Ising models. They belong to the more general class of Energy-Based Models (LeCun and Huang, 2005; LeCun et al., 2006), which are un-normalized density models. Similar to Boltzmann machines, EBMs define an energy function $E(x)$, the parameters of which are adapted to lower the energy around training examples and raise the energy of configurations not supported by the data. Contrary to BMs however, the energy does not necessarily integrate to a finite quantity. As we shall see in Section 2.8, this lack of partition function greatly impacts their training algorithm.

2.2 Maximum Likelihood Learning

As with any other probabilistic model, the parameters of a Boltzmann machine can be estimated through maximum likelihood. Recall the following notation: a training set \mathcal{D} having i.i.d. examples drawn from an empirical distribution $\pi(x)$ and a model denoted as $p(x; \theta)$. Maximum likelihood finds the optimal parameters θ^* which maximize the likelihood function $\mathcal{L}(\mathcal{D}, p_\theta) = \prod_{x^{(i)} \in \mathcal{D}} p(x^{(i)}; \theta)$:

$$\begin{aligned} \theta^* &\leftarrow \arg \max_{\theta} \mathcal{L}(\mathcal{D}, p_\theta) \Leftrightarrow \\ \theta^* &\leftarrow \arg \min_{\theta} \mathbb{E}_{\pi} [-\log p(x; \theta)], \end{aligned} \tag{2.6}$$

with the equivalence stemming from the monotonicity of the logarithm. Note that we also recast maximum likelihood as a minimization problem in order to follow

convention. Alternatively, one can view maximum likelihood as minimizing the KL-divergence between the empirical and model distributionsⁱ, defined as:

$$\text{KL}(\pi\|p) = \sum_x \pi(x) \log \frac{\pi(x)}{p(x)} \quad (2.7)$$

Provided the free-energy function is tractable and differentiable almost everywhere, Eq. 2.6 can be minimized through gradient descent (as shown in Section 1.2). The gradient of the negative log-likelihood (NLL) function of a Boltzmann Machine with free-energy function $F(x)$, can be derived as follows:

$$\begin{aligned} -\log p(x; \theta) &= F(x) + \log Z \\ &= F(x) + \log \sum_x \exp[-F(x)] \\ -\frac{\partial \log p(x)}{\partial \theta} &= \frac{\partial F(x)}{\partial \theta} + \frac{1}{Z} \sum_x \frac{\partial \exp[-F(x)]}{\partial \theta} \\ &= \frac{\partial F(x)}{\partial \theta} - \frac{1}{Z} \sum_x \exp[-F(x)] \frac{\partial F(x)}{\partial \theta} \\ &= \frac{\partial F(x)}{\partial \theta} - \sum_x p(x) \frac{\partial F(x)}{\partial \theta} \\ &= \frac{\partial F(x)}{\partial \theta} - \mathbb{E}_p \left[\frac{\partial F(x)}{\partial \theta} \right] \end{aligned} \quad (2.8)$$

Replacing the expectation of Eq. 2.6 by a sample average, the maximum likelihood SGD update equations for θ , at the t -th parameter update, becomes:

$$\theta_{t+1} \leftarrow \theta_t - \epsilon \left(\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \left[\frac{\partial F(x)}{\partial \theta} \right] - \mathbb{E}_p \left[\frac{\partial F(x')}{\partial \theta} \right] \right) \quad (2.9)$$

We denote the model with parameters θ_t as $p_t(x)$. It can be seen that the log-likelihood gradient involves the computation of two terms. The first term can be computed analytically and has the effect of decreasing the (free)-energy at training points. It is referred to as the positive phase. The summation can be taken across

i. The equivalence can be proven trivially.

$\text{KL}(\pi\|p) = \sum_x \pi(x) \log \frac{\pi(x)}{p(x)} = \sum_x \pi(x) \log \pi(x) - \sum_x \pi(x) \log p(x)$.
 $\arg \min_{\theta} \text{KL}(\pi\|p) \Leftrightarrow \arg \min_{\theta} - \sum_x \pi(x) \log p(x) \Leftrightarrow \arg \max_{\theta} \mathbb{E}_{\pi} [\log p(x)]$

the entire training set for batch gradient descent, or over a subset of the data when using gradient descent with mini-batches. The second term aims to increase the probability of all configurations of the visible units, and is called the negative phase. Its computation is problematic however, since it involves an expectation over the model distribution, which is intractable for all but the smallest models.

The log-likelihood gradient can thus further be approximated by replacing the expectation with a sample average. Given the ability to generate a set of model samples $\mathcal{X}_t = \{x_n \sim p_t(x); 0 < n < N\}$, a gradient estimator can be computed and applied, yielding:

$$\theta_{t+1} \leftarrow \theta_t - \epsilon \left(\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \left[\frac{\partial F(x)}{\partial \theta} \right] - \frac{1}{N} \sum_{x' \in \mathcal{X}_t} \left[\frac{\partial F(x')}{\partial \theta} \right] \right) \quad (2.10)$$

We refer to x as positive samples, since they are training examples used to estimate the positive phase, while x' are referred to as negative or model samples or alternatively, as particles of some sampling algorithm. Typically, we choose $N = |\mathcal{D}|$ in order to match the variances of the positive and negative phases.

When the free-energy function is not tractable, as is the case with the Deep Boltzmann Machine (DBM, see Section 2.5.2), we can re-derive the update equation with respect to the model's energy function $E(x)$.

$$\theta_{t+1} \leftarrow \theta_t - \epsilon \left(\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathbb{E}_{p(h|v=x)} \left[\frac{\partial E(x)}{\partial \theta} \right] - \mathbb{E}_p \left[\frac{\partial E(x')}{\partial \theta} \right] \right) \quad (2.11)$$

For these models, estimating the positive phase often becomes the bottleneck. While several algorithms have been developed which allow for efficient approximations to the negative phase gradients, these same methods cannot be used to approximate expectations with respect to the model's posterior. The common solution is then to resort to variational learning methods, presented in Sections 1.4.2 and 2.4.5.

2.3 Restricted Boltzmann Machines

2.3.1 RBM for Binary Data

The RBM is characterized by the following energy function:

$$E_{\theta}(v, h) = - \sum_{i=1}^{n_v} \sum_{j=1}^{n_h} W_{ij} h_j v_i - \sum_j c_j h_j - \sum_i b_i v_i, \quad (2.12)$$

which groups n_v visible units and n_h hidden units into two separate layers, interacting through the weight matrix $W \in \mathbb{R}^{n_h \times n_v}$. Connections between units of the same layer are prohibited, differentiating it from other more general Boltzmann Machines. $b \in \mathbb{R}^{n_v}$ and $c \in \mathbb{R}^{n_h}$ are the offsets of the visible and hidden units respectively, and serve much the same purpose as in the MLP. While the random variable v and h can belong to many probability distributions, we will focus here on the binary-binary RBM, where $v \in \{0, 1\}^{n_v}$ and $h \in \{0, 1\}^{n_h}$.

Conditionals

Before we can perform learning or inference in this model, we first need to derive its conditional distributions $p(h|v)$ and $p(v|h)$.

$$\begin{aligned} p(h|v) &= \frac{p(v, h)}{p(v)} = \frac{\exp[-E_{\theta}(v, h)]}{\sum_h \exp[-E_{\theta}(v, h)]} \\ &= \frac{\exp\left[-\sum_j h_j (c_j + \sum_i W_{ij} v_i)\right] \exp(-\sum_i b_i v_i)}{\sum_h \exp\left[-\sum_j h_j (c_j + \sum_i W_{ij} v_i)\right] \exp(-\sum_i b_i v_i)} \\ &= \frac{\prod_j \exp[-h_j (c_j + \sum_i W_{ij} v_i)]}{\sum_h \prod_j \exp[-h_j (c_j + \sum_i W_{ij} v_i)]} \\ &= \prod_j \frac{\exp[-h_j (c_j + \sum_i W_{ij} v_i)]}{1 + \exp[-c_j - \sum_i W_{ij} v_i]} \end{aligned}$$

We recover the following activation functions:

$$p(h|v) = \prod_j p(h_j|v) \quad (2.13)$$

$$p(h_j = 1|v) = \text{sigmoid} \left(c_j + \sum_i W_{ij} v_i \right) \quad (2.14)$$

By symmetry of the energy function, we can write:

$$p(v|h) = \prod_i p(v_i|h) \quad (2.15)$$

$$p(v_i = 1|h) = \text{sigmoid} \left(b_i + \sum_j W_{ij} h_j \right) \quad (2.16)$$

Eqs.(2.13-2.16) show that hidden units are conditionally independent, given the visible layer (and vice-versa). This property greatly facilitates generating the set \mathcal{X}_t of negative phase samples, required for learning in Eq. 2.10. Because of its factorial posteriors, we can generate approximate samples of $p(v)$ efficiently by performing block Gibbs sampling (Robert and Casella, 1999). We can obtain each sample $x' \in \mathcal{X}_t$ by simulating a Markov chain which alternates sampling $h^{(k)} \sim p_t(h|v = v^{(k-1)})$ and $v^{(k+1)} \sim p_t(v|h^{(k)})$. The samples in \mathcal{X}_t are the $v^{(k)}$'s obtained for large values of k (typically in the order of thousands), at which point the Markov chain is deemed to have converged to its stationary distribution.

Free-Energy

We can derive the free-energy of the binary-binary RBM as follows:

$$\begin{aligned}
F_\theta(v) &= -\log \sum_h \exp^{-E_\theta(v,h)} \\
&= -\log \left[\exp \left(-\sum_i b_i v_i \right) \sum_h \prod_j \exp \left[-h_j \left(c_j + \sum_i W_{ij} v_i \right) \right] \right] \\
&= -\sum_i b_i v_i - \log \prod_j \left[1 + \exp \left[-c_j - \sum_i W_{ij} v_i \right] \right] \\
&= -\sum_i b_i v_i - \sum_j \log \left[1 + \exp \left(-c_j - \sum_i W_{ij} v_i \right) \right] \tag{2.17}
\end{aligned}$$

2.3.2 RBM for Continuous Data

When v is a continuous vector in \mathbb{R}^{n_v} , the energy function of the RBM must be adapted slightly. Here we aim to model $p(v | h)$ as the product of independent Gaussian distributions where as in Eq. 2.16, the mean of $p(v_i | h)$ is given by the product $W_i \cdot h$. Introducing the parameter $\Lambda_{ii} \in \mathbb{R}$ to be the diagonal precision matrix of unit v_i , we thus want:

$$\begin{aligned}
p_{GRBM}(v_i | h) &= \mathcal{N} \left(v_i; b_i + \sum_j W_{ij} h_j, \Lambda_{ii}^{-\frac{1}{2}} \right) \\
&\propto \exp \left\{ -\frac{\Lambda_{ii}}{2} \left(v_i - \left[b_i + \sum_j W_{ij} h_j \right] \right)^2 \right\}, \tag{2.18}
\end{aligned}$$

where $\mathcal{N}(x; \mu, \sigma)$ denotes that x is normally distributed with mean μ and standard deviation σ . Given this modeling objective, we can now work backwards from the Gaussian RBM (GRBM) conditional to define the GRBM energy function. It is obtained by taking the log of Eq.2.18 and expanding the square. Since RBMs prohibit connections between hidden units, we drop all terms involving products $h_j h_k$, with $k \neq j$, along with any constant terms (terms which are not a function of the random variables v and h). Denoting Λ as the diagonal matrix whose i -th

entry on the diagonal is Λ_{ii} , the resulting energy function is thus given by:

$$E_{GRBM}(v, h) = \frac{1}{2}v^T \Lambda v - v^T \Lambda W h - b^T v - c^T h \quad (2.19)$$

With a few lines of algebra, this energy function can be shown to yield the following conditional distribution $p(h_j | v)$,

$$p(h_j = 1 | v) = \text{sigmoid} \left(c_j + \sum_i \Lambda_{ii} W_{ij} v_i \right) \quad (2.20)$$

and free-energy function:

$$F_{GRBM}(v) = \frac{1}{2}v^T \Lambda v - b^T v - \sum_j \log \left[1 + \exp \left(-c_j - \sum_i v_i \Lambda_{ii} W_{ij} \right) \right]. \quad (2.21)$$

In practice, it can be very difficult to learn the precision parameter Λ through first-order gradient descent. It is therefore common practice to standardize the dataset to have unit standard deviation (per input dimension) and keep the conditional precision fixed to some value greater or equal to one (Hinton, 2010).

Capturing higher-order statistics

While the GRBM was specifically crafted to model continuous data, it remains a coarse model for learning rich and complex distributions such as that formed by natural images. In particular, the GRBM inference process (and hence the representation formed by its latent variables when conditioned on the visible units) is unable to take into account second (or higher) order statistics of the input pixels.ⁱ This is especially damning as a core statistical property of natural images is the high correlation between neighboring pixels (Simoncelli and Olshausen, 2001). Concretely, this limitation stems from the GRBM's conditional distribution $p(h | v)$, which involves a simple linear combination of the input pixels v_k . Capturing higher-order statistics would require products such as $v_k \cdot v_l$ to appear in the activation function: in other words higher-order polynomials of the input vector v .

Two such models have been proposed in the Boltzmann Machine family. The

ⁱ. This limitation mirrors that found in the GRBM generative process, whereby $p(x | h)$ is constrained to have a diagonal covariance matrix.

mean-covariance RBM (Ranzato and Hinton, 2010) implements this by introducing “covariance” hidden units which interact with the visible layer through a quadratic energy term in v . It is thus a special case of third-order Boltzmann machines, which are presented in Section 2.6.1. The spike and slab RBM (Courville et al., 2011a) takes a different route altogether. It augments each binary hidden unit h_j with a continuous slab variable $s_j \in \mathbb{R}$, whose role is to modulate the precise contribution of each feature vector $W_{.j}$ in the visible-hidden unit interaction term. While the posterior $p(h, s \mid v)$ remains linear in v , the slab variables can be integrated out analytically, yielding a sigmoidal activation function of a second-order polynomial in v . The spike and slab formulation also has the benefit of being amenable to Gibbs sampling unlike the mcRBM, with Ranzato and Hinton (2010) resorting to Hybrid Monte Carlo (Neal, 1996) instead. The spike and slab RBM, along with several of its extensions are the topic of the paper presented in Chapter 10.

2.4 Approximations to Maximum Likelihood and Other Inductive Principles

The maximum likelihood learning rule of Eq. 2.10 require a fresh set of negative phase samples for each gradient update. This is impractical as it technically requires running a Markov chain to convergence for each gradient update! For this reason, various algorithms have been developed, which differ both in their computational complexity and in their approximation to the negative phase. These are covered in the following section.

2.4.1 Stochastic Maximum Likelihood

Stochastic Maximum Likelihood (SML) or Persistent Contrastive Divergence (PCD) (Younes, 1998; Tieleman, 2008) exploits the fact that the model changes only slightly between consecutive gradient updates. As such, samples \mathcal{X}_{t-1} are still somewhat representative of model p_t . SML thus initializes the Markov chains at time-step t with elements from \mathcal{X}_{t-1} , drastically reducing the burn-in time of the Markov chain in the process. Only a few steps of Gibbs sampling are typically used

to generate the samples \mathcal{X}_t . In practice, this number can even be as low as one (Tieleman, 2008). Algorithm 2 formalizes the SML update algorithm.

Algorithm 2 Stochastic Maximum Likelihood Update

Obtain (mini)-batch of training examples $\mathcal{X}_t^+ = \{x \in \mathcal{D}\}$.
 Initialize (mini)-batch of samples $x_t^{(0)}$ from \mathcal{X}_{t-1} , approx. samples of p_{t-1} .
for $k = 0 : K$ **do**
 Generate (mini)-batch of samples $h_t^{(k)} \sim p_t(h|v = x_t^{(k)})$.
 Generate (mini)-batch of samples $x_t^{(k+1)} \sim p_t(v|h = h_t^{(k)})$.
end for
 Define \mathcal{X}_t as the (mini)-batch of samples $x_t^{(k)}$.
 $\theta_{t+1} \leftarrow \theta_t - \epsilon \left(\frac{1}{|\mathcal{X}_t^+|} \sum_{x \in \mathcal{X}_t^+} \left[\frac{\partial F(x)}{\partial \theta} \right] - \frac{1}{|\mathcal{X}_t|} \sum_{x' \in \mathcal{X}_t} \left[\frac{\partial F(x')}{\partial \theta} \right] \right)$.

The crucial assumption in SML is that k steps of Gibbs sampling are sufficient to reach the new equilibrium distribution. Unfortunately, this may not always be the case. Denoting the state of the Markov chain after k steps of Gibbs sampling as $p_t^{(k)}$, Tieleman and Hinton (2009) showed that SML actually follows the gradient of $\text{KL}(\pi \| p_t) - \text{KL}(p_t^{(k)} \| p_t)$. If the gradient vectors of both these terms have a positive dot-product, SML will successfully minimize $\text{KL}(\pi \| p_t)$, and thus maximize the model likelihood. Otherwise, SML may diverge from the maximum likelihood solution.

SML thus relies on $\text{KL}(p_t^{(k)} \| p_t)$ being small throughout training. However, for a fixed value of k , this quantity is known to increase with training time. Intuitively, this occurs because as the parameters increase in magnitude, the Markov chain becomes increasingly deterministic (the conditional probabilities become increasingly peaked around 0/1), which in turn reduces the mixing rate or ergodicityⁱ of the chain. Younes (1998) establishes some necessary conditions for convergence, showing that a decreasing learning rate schedule is required to offset this loss of ergodicity.ⁱⁱ

i. We use the term “ergodicity” rather loosely, to reflect the amount of time required for the states sampled by the Markov chain to reflect the true expectation we wish to measure.

ii. The learning rate ϵ_t should decrease such that $\sum_{t=0}^{\infty} \epsilon_t = \infty$ while $\sum_{t=0}^{\infty} \epsilon_t^2 < \infty$.

2.4.2 The Fast-Weight Effect and FPCD

It may come as a surprise to the Markov chain practitioner that a single step of Gibbs sampling suffices to draw samples from $p_t(x)$ when initialized with samples from $p_{t-1}(x)$. [Tieleman and Hinton \(2009\)](#)'s investigations into the matter revealed an interesting link between the mixing rate of the Markov chain and learning: the gradient update at time-step $t - 1$ biases the Markov chain towards moving *away* from the configuration \mathcal{X}_{t-1} . By reducing the probability of \mathcal{X}_{t-1} , learning encourages the chain to move to another region of input space, increasing the mixing rate of the chain in the process. This phenomenon, coined the “fast-weight effect”, is especially pronounced when using large learning rates. Unfortunately, this is at odds with SML which requires a decreasing learning rate to guarantee convergence.

Fast PCD (FPCD) ([Tieleman and Hinton, 2009](#)) addresses this issue by introducing an additional set of *fast weights* W_F , which combine additively with the normal RBM weights W in the negative phase of learning. Weights W_F are updated using the same gradient as W , but using a separate learning rate which remains large throughout training. This allows the “normal” learning rate (the one operating on all other parameters of the model) to be annealed throughout training, while still maintaining the fast-weight effect in the negative phase Markov chain. A strong L2 penalty on W_F ensures that their effect on the model is only temporary and that the KL divergence between the models defined with and without the fast-weights remains small. While its theoretical foundations are weak, the FPCD update algorithm of [Algorithm 3](#) has been found to work well in practice.

2.4.3 Tempered SML

Alternatively, one may also use more powerful sampling algorithms in the negative phase of learning, such that the resulting Markov chain maintains good mixing properties throughout training. One such strategy employed in [Desjardins et al. \(2010b\)](#); [Cho et al. \(2010\)](#); [Salakhutdinov \(2010b,a\)](#), relies on generating samples from the tempered Gibbs distribution $p(v; \beta_i) = \frac{1}{Z(\beta_i)} \exp[-\beta_i F(v)]$, where $\beta \in [0, 1]$ is an inverse temperature parameter. At high temperatures ($\beta \ll 1$), $p(v; \beta)$ becomes more uniform over the sampling space, resulting in a smoothed version of $p(v; \beta = 1)$ which is easier to sample from. [Desjardins et al. \(2010b\)](#); [Salakhutdinov \(2010b,a\)](#) offer competing solutions on how to best exploit these

Algorithm 3 Fast PCD Update

$W_{F,t}$: fast-weight matrix at time-step t .

$\theta_t = \{W_t, b_t, c_t\}$: “normal” RBM parameters at time-step t .

$\theta_{F,t} = \{W_t + W_{F,t}, b_t, c_t\}$: “fast” RBM parameters at time-step t .

λ : hyper-parameter controlling amount of L2 regularization on W_F .

ϵ, ϵ_F : “normal” and “fast” learning rates.

Obtain (mini)-batch of training examples $\mathcal{X}_t^+ = \{x \in \mathcal{D}\}$.

Initialize (mini)-batch of samples $x_t^{(0)}$ from \mathcal{X}_{t-1} , approx. samples of p_{t-1} .

for $k = 0 : K$ **do**

 Generate (mini)-batch of samples $h_t^{(k)} \sim p_t(h|v = x_t^{(k)}; \theta_{F,t})$.

 Generate (mini)-batch of samples $x_t^{(k+1)} \sim p_t(v|h = h_t^{(k)}; \theta_{F,t})$.

end for

Define \mathcal{X}_t as the (mini)-batch of samples $x_t^{(k)}$.

$$\theta_{t+1} \leftarrow \theta_t - \epsilon \left(\frac{1}{|\mathcal{X}_t^+|} \sum_{x \in \mathcal{X}_t^+} \left[\frac{\partial F(x; \theta_t)}{\partial \theta} \right] - \frac{1}{|\mathcal{X}_t|} \sum_{x' \in \mathcal{X}_t} \left[\frac{\partial F(x'; \theta_{F,t})}{\partial \theta} \right] \right).$$

$$W_{F,t+1} \leftarrow W_{F,t} - \epsilon_F \left(\frac{1}{|\mathcal{X}_t^+|} \sum_{x \in \mathcal{X}_t^+} \left[\frac{\partial F(x; \theta_t)}{\partial W_F} \right] - \frac{1}{|\mathcal{X}_t|} \sum_{x' \in \mathcal{X}_t} \left[\frac{\partial F(x'; \theta_{F,t})}{\partial W_F} \right] \right) + \lambda W_{F,t}.$$

fast-mixing chains to generate samples from $p(v; \beta = 1)$.

In [Desjardins et al. \(2010b\)](#), we proposed using Parallel Tempering (PT) in the negative phase of SML, which resulted in increased performance and added robustness towards the choice of learning rate and the number of training epochs. PT relies on sampling from an extended system composed of multiple RBM models: $\mathcal{M}_t = \{\mathcal{M}_{1,t}, \mathcal{M}_{2,t}, \dots, \mathcal{M}_{M,t}\}$ using a set of M parallel Markov chains, with the i -th chain drawing samples from the associated RBM, $\mathcal{M}_{i,t}$. We introduce the notation $\mathcal{M}_{i,t}$ to refer to the model at time-step t , with probability distribution $p_t(v; \beta_i) \equiv p_{i,t}(v)$ and associated partition function $Z_{i,t}$. Formally, PT provides a mechanism for sampling from the product distribution $\mathbf{p}_t = \prod_{i=1}^M \{p_t(v; \beta_i)\}$ by alternating two Metropolis-Hastings transition operators which leave \mathbf{p}_t invariant.

The first consists in performing k steps of Gibbs sampling independently, for each of the M model distributions. This operator leaves each chain invariant (as in the standard RBM setting) and thus also \mathbf{p}_t . For large M , low values of β_i have the effect of smoothing the energy function, making the MCMC simulation increasingly efficient with temperature. The second transition operator consists in swapping replicas (or samples) between neighboring chains. Denoting $x_{i,t} \sim p_{i,t}$, the swap between chains $(i, i+1)$ is accepted with probability $r_{i,t}$, computed by the standard Metropolis-Hastings algorithm ([Metropolis et al., 1953](#)). Since each swap is local

in that it only affects densities p_i and p_{i+1} , most terms in the expression of \mathbf{p}_t drop out, yielding:

$$r_{i,t} = \max \left(1, \frac{p_{i,t}(x_{i+1,t})p_{i+1,t}(x_{i,t})}{p_{i,t}(x_{i,t})p_{i+1,t}(x_{i+1,t})} \right). \quad (2.22)$$

These swaps ensure that samples from highly ergodic chains are gradually swapped into lower temperatures. This allows samples to escape from local minima of the energy landscape, by gradually being swapped to higher temperatures, mixing, and slowly annealing back to the nominal temperature.

Several swapping schedules¹ have been proposed in [Lingenheil et al. \(2009\)](#) which satisfy detailed balance, guaranteeing that for large enough k , $x_{i,t} \sim p_{i,t}$. While [Desjardins et al. \(2010b\)](#) employed a random selection strategy, [Lingenheil et al. \(2009\)](#) advocates using the Deterministic Even Odd (DEO) algorithm. The resulting SML-PT update algorithm is shown in Algorithm 4.

Algorithm 4 SML with Parallel Tempering (SML-PT) Update

Obtain (mini)-batch of training examples $\mathcal{X}_t^+ = \{x \in \mathcal{D}\}$.
 Initialize (mini)-batch of samples $x_{i,t}^{(0)}$ from $\mathcal{X}_{i,t-1}$, $\forall i \in [1, M]$.
 Perform k steps of Gibbs sampling for $\mathcal{M}_{i,t}$, yielding a (mini)-batch of samples $x_{i,t}, \forall i$.
for all *even* chains i (and all samples in mini-batch) **do**
 swap $x_{i,t} \leftrightarrow x_{i+1,t}$ with probability $r_{i,t}$.
end for
for all *odd* chains i (and all samples in mini-batch) **do**
 swap $x_{i,t} \leftrightarrow x_{i+1,t}$ with probability $r_{i,t}$.
end for
 Define $\mathcal{X}_{i,t}$ as the (mini)-batch of samples $x_{i,t}$.
 $\theta_{t+1} \leftarrow \theta_t - \epsilon \left(\frac{1}{|\mathcal{X}_t^+|} \sum_{x \in \mathcal{X}_t^+} \left[\frac{\partial F(x; \theta_t)}{\partial \theta} \right] - \frac{1}{|\mathcal{X}_{1,t}|} \sum_{x' \in \mathcal{X}_{1,t}} \left[\frac{\partial F(x'; \theta_{F,t})}{\partial \theta} \right] \right)$.

2.4.4 Contrastive Divergence

Contrastive Divergence (CD) ([Hinton, 2002](#)), the algorithm first proposed for training Deep Networks, remains one of the leading inductive principles for training energy-based models. It relies on the same functional form of the maximum

i. The swapping schedule refers to the algorithm for selecting the pairs (i, j) of chains to swap and its ordering.

likelihood gradient (Eq. 2.6), but changes the nature of the negative samples. Similar to SML, negative samples \mathcal{X}_t are generated by running a Markov chain (with $p_t(v)$ as its stationary distribution) for only k -steps. The difference however, is that these chains are initialized with a positive training example instead of \mathcal{X}_{t-1} (as with SML). Since the equilibrium distribution of the Markov chain will tend towards $\pi(x)$ with learning (assuming convergence), initializing its state from a training example $x \in \mathcal{D}$ should help bypass the long burn-in process required by maximum likelihood. The CD- k update algorithm is shown in Algorithm 5.

Algorithm 5 Contrastive Divergence Update

Obtain (mini)-batch of training examples $\mathcal{X}_t^+ = \{x \in \mathcal{D}\}$.

Initialize (mini)-batch of samples $x_t^{(0)}$ from \mathcal{X}_t^+ .

for $k = 0 : K$ **do**

Generate (mini)-batch of samples $h_t^{(k)} \sim p_t(h|v = x_t^{(k)})$.

Generate (mini)-batch of samples $x_t^{(k+1)} \sim p_t(v|h = h_t^{(k)})$.

end for

Define \mathcal{X}_t as the (mini)-batch of samples $x_t^{(k)}$.

$\theta_{t+1} \leftarrow \theta_t - \epsilon \left(\frac{1}{|\mathcal{X}_t^+|} \sum_{x \in \mathcal{X}_t^+} \left[\frac{\partial F(x)}{\partial \theta} \right] - \frac{1}{|\mathcal{X}_t|} \sum_{x' \in \mathcal{X}_t} \left[\frac{\partial F(x')}{\partial \theta} \right] \right)$.

CD has proven extremely successful as a pre-training and feature extraction algorithm for deep networks. From a generative model point of view however, CD has been shown to perform worse in terms of log-likelihood than other algorithms in the SML family (Marlin et al., 2010; Desjardins et al., 2010b).

2.4.5 Variational SML

Estimating the positive phase gradient of Eq. 2.11 requires computing an expectation of the energy derivative over the model’s posterior distribution. For some models like the DBM or bilinear ssRBM of Chapters 8 and 12, this term does not have a closed form solution. Furthermore, using MCMC to approximate this expectation is problematic as it is an expectation wrt. a conditional distribution. One would thus have to run an MCMC simulation to convergence for each training example, a prohibitively expensive solution. This is in stark contrast to the negative phase expectation which admits an efficient solution based on MCMC (see Section 2.4.1). For these types of models, we instead turn to *variational learning* methods, which were first described in Section 1.4.2.

When using the mean-field assumption for BMs, recall that $q(h) = \prod_1^M q_i(h_i)$ and $q_{/i} = \prod_{j \neq i} q_j(h_j)$. Starting from the Euler-Lagrange equation (Eq. 1.24), the logarithm of factor i takes on a simple form: it is the expected value of the (negative) energy function under $q_{/i}$:

$$\begin{aligned} \log q_i(h_i) &= \mathbb{E}_{q_{/i}} [\log p(x, h)] \\ &= \mathbb{E}_{q_{/i}} \left[\log \frac{1}{Z} \exp \{-E(x, h)\} \right] \\ &= \mathbb{E}_{q_{/i}} [-E(x, h)] + \text{cte}, \end{aligned} \quad (2.23)$$

where the constant term stems from the log-partition function and is simply rolled into the normalization constant of $q_i(x_i)$. The E-step thus consists in clamping the visible units to x and repeatedly iterating ($\forall i \in [1, M]$) Eq. 2.23 until convergence, yielding $q^*(h)$: the optimal mean-field distribution minimizing $KL(q||p(h||x))$.

As in the standard EM algorithm (see Section 1.4.1), the M-step then consists in a gradient ascent step of the lower-bound $\mathcal{L}(q)$ wrt. θ , computed as follows:

$$\begin{aligned} \mathcal{L}(q) &= \sum_h q^*(h) \log \left\{ \frac{p(x, h; \theta)}{q^*(h)} \right\} \\ &= -\mathbb{E}_{q^*(h)} [E(x, h; \theta)] - \log Z + \mathcal{H}[q^*(h)], \\ \frac{\partial \mathcal{L}(q)}{\partial \theta} &= -\mathbb{E}_{q^*(h)} \left[\frac{\partial}{\partial \theta} E(x, h; \theta) \right] + \mathbb{E}_{p(v, h)} \left[\frac{\partial E}{\partial \theta} \right]. \end{aligned} \quad (2.24)$$

Notice that the update direction is almost identical to Eq. 2.11, with the only difference being that the positive phase expectation is computed wrt. our variational approximation. The variational Stochastic Maximum Likelihood algorithm is obtained by again approximating the negative phase expectation via a sample average which computes sufficient statistics over a persistent Markov chain, as in Section 2.4.1.

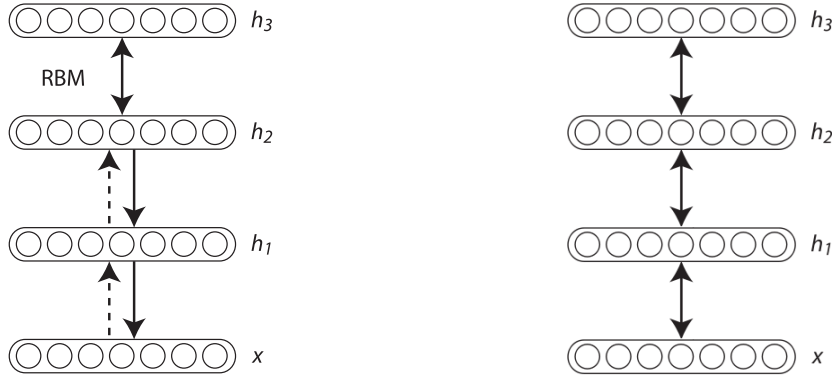


Figure 2.1: Graphical model of the Deep Belief Network (DBN) and Deep Boltzmann Machine (DBM). (left) A DBN is composed of undirected connections between the top two layers and directed connections going down. The hashed arrows represent the conditional distribution $p(h^{(l)}|h^{(l-1)})$ learnt during the pretraining phase and can be used to perform approximate inference. (right) DBMs are similar in structure but have undirected connections between all layers. See Sec. 2.5.2. Images reproduced with permission from Bengio (2009).

2.5 Multi-Layer Models

RBM s can serve as simple unsupervised feature extractors, which learn to model the input distribution through a linear combination of basis functions. Inspired by the success of deep MLPs, RBMs have been extended to include multiple layers of latent variables, which together form a powerful hierarchical representation. Two popular variants are presented below.

2.5.1 Deep Belief Networks

Deep Belief Networks (Hinton et al., 2006) learn to extract such a deep hierarchical representation, by stacking multiple layers of RBMs. Their training algorithm is a greedy iterative procedure, which trains the RBM in layer l to model the posterior distribution of the RBM at layer $l-1$. In doing so, one obtains an increasingly more abstract representation of the input. Denoting $v := h^{(0)}$ and L the number of layers, its joint distribution is given by:

$$p_{\theta}(v, h^{(1)}, \dots, h^{(L)}) = \left[\prod_{l=1}^{L-1} p(h^{(l-1)}|h^{(l)}) \right] p(h^{(L-1)}, h^{(L)}). \quad (2.25)$$

The graphical model is depicted in Fig. 2.1.

In its original formulation, the DBN was applied to a classification problem, by training the L -th RBM on the concatenation of $h^{(L-1)}$ and the class labels. The bottom-up recognition weights were used to perform approximate inference, by setting $h^{(l)}$ to the expected value of $p(h^{(l)}|h^{(l-1)})$ (see Eq. 2.13). The label was then selected as the one having minimal free-energy, as measured by the L -th RBM. Conversely, one may also sample from the model by first sampling $h^{(L-1)} \sim p(h^{(L-1)}, h^{(L)})$ and using the top-down directed connections to generate v .

Bengio et al. (2007) proposed an alternative method for performing classification using DBNs. They showed that the parameters obtained by the greedy layer-wise strategy could be used as an initialization point for deep (feed-forward) MLPs (with an equivalent architecture). The model parameters could then be fine-tuned through standard supervised learning (i.e. backpropagation algorithm of Section 1.3.2). At the time of publication, both of the above pretraining strategies led to significant improvements on the MNIST classification task, dropping from 1.8% error on MNIST to 1.2%. Erhan et al. (2010) later showed that these gains in performance stemmed largely from a regularization effect, i.e. constraining the parameters of the model to be close to those obtained via unsupervised learning.

The benefit of these hierarchical models was epitomized in Lee et al. (2009) however, where a convolutional version of the DBN (CDBN) was trained on natural images. In a completely unsupervised manner, the CDBN learnt lower-level features resembling Gabor-filter, reminiscent of cells in area V1 of the visual cortex (Olshausen and Field, 1996), which combined to form object parts and object detectors in the higher layers. The features encoded by a each layer were also found to be increasingly invariant and class specific.

2.5.2 Deep Boltzmann Machines

The Deep Boltzmann Machine (DBM) (Salakhutdinov and Hinton, 2009a) is the natural extension of the RBM to deep architectures. It is composed of multiple layers of random variables, with undirected connections between units in adjacent layers, as shown in Fig. 2.1 (right). The energy function of a two layer DBM is

given below:

$$\begin{aligned}
 E(v, h^{(1)}, h^{(2)}) = & - \sum_{i=1}^{n_v} \sum_{j=1}^{n_{h1}} h_j^{(1)} W_{ij}^{(1)} v_i - \sum_{j=1}^{n_{h1}} \sum_{k=1}^{n_{h2}} h_k^{(2)} W_{jk}^{(2)} h_j^{(2)} \\
 & - \sum_k c_k^{(2)} h_k^{(2)} - \sum_j c_j^{(1)} h_j^{(1)} - \sum_i b_i v_i,
 \end{aligned} \tag{2.26}$$

We modify the RBM notation slightly, introducing the superscript notation to indicate that weights $W^{(k)}$ and biases $c^{(k)}$ belong to the k -th hidden layer. From the graphical depiction of Fig. 2.1, it is clear that layers $h^{(2)}$ and v are conditionally independent given $h^{(1)}$. As such, their conditional distributions are identical to those obtained through RBMs with weights $W^{(2)}$ and $W^{(1)}$ respectively. $p(h^{(1)}|v, h^{(2)})$ on the other hand is much more interesting and shows how deeper layers help modulate the activations of lower-level units:

$$p(h_j^{(1)} = 1|v, h^{(2)}) = \text{sigmoid} \left(\sum_i^{n_v} W_{ij}^{(1)} v_i + \sum_k^{n_{h2}} W_{jk}^{(2)} h_k^{(2)} + c_j^{(1)} \right). \tag{2.27}$$

Unlike the RBM, the DBM posterior is not factorial and does not have a closed form solution. As such, the maximum likelihood gradient can either be computed as shown in Eq. 2.11, by estimating the positive phase expectation via MCMC, or more commonly, as in Eq. 2.24 using variational inference to estimate the positive phase statistics. Negative phase sampling remains efficient through block Gibbs sampling however: even layers can be sampled jointly when conditioned on odd layers, and vice-versa.

Layer-wise Pretraining . Reminiscent of the difficulties in training deep MLPs, Salakhutdinov and Hinton (2009a) found that first-order gradient descent was incapable of optimizing the joint likelihood $p(v, h^{(1)}, h^{(2)}) \propto \exp[-E(v, h^{(1)}, h^{(2)})]$ from random parameter initialization. They thus proposed an adapted version of the DBN layer-wise pretraining algorithm, which is modified to account for future top-down interactions. Pretraining is then followed by joint training via variational SML. Salakhutdinov (2010a); Salakhutdinov and Hinton (2009a) reports state of the art results on test-set log-likelihood for the MNIST (LeCun et al., 1998) and NORB (LeCun et al., 2004) datasets. The samples generated by these trained models are shown in Fig. 2.2.

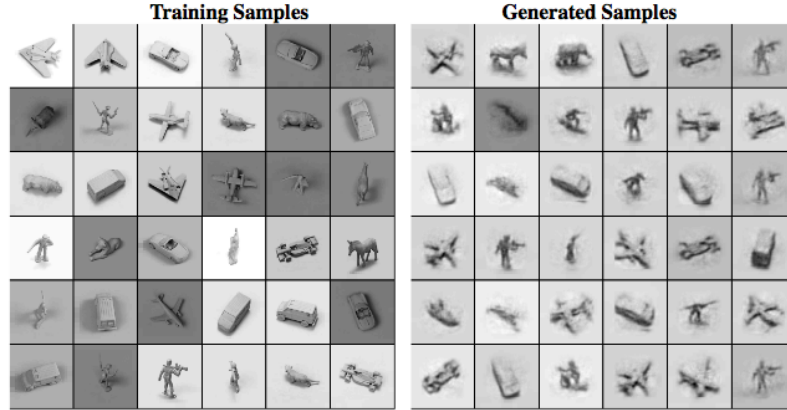


Figure 2.2: (left) Examples images from the NORB dataset. (right) Samples generated by a trained DBM. Image reproduced with permission from [Salakhutdinov and Hinton \(2009a\)](#).

The Centering Trick [Montavon and Muller \(2012\)](#) recently showed that with a simple reparametrization of the DBM energy function, one could do away with greedy layer-wise pretraining and directly learn the model parameters jointly through simple gradient descent. The trick consists in performing a linear reparametrization of the energy function such that singleton or pairwise potential functions are taken wrt. “centered” random variables, i.e. variables having zero mean, as shown below:

$$\begin{aligned}
 E(v, h^{(1)}, h^{(2)}) = & - (v^T - \zeta_v)^T W^{(1)} (h^{(1)} - \zeta_{h1}) \\
 & - (h^{(1)T} - \zeta_{h1})^T W^{(2)} (h^{(2)} - \zeta_{h2}) \\
 & - (v^T - \zeta_v) b - (h^{(1)T} - \zeta_{h1}) c^{(1)} - (h^{(2)T} - \zeta_{h2}) c^{(2)}.
 \end{aligned} \tag{2.28}$$

The centering coefficients $\{\zeta_v, \zeta_{h1}, \zeta_{h2}\}$ are not learnt, but adapted deterministically to maintain zero-mean activation of each variable. Surprisingly, the method also works by fixing the centering coefficients of the latent variables to 0.5 (a good approximation to their expected value given small initial weights), and the visible centering coefficients to the log-odds ratio of the empirical distribution, a well known trick for speeding up optimization of RBMs ([Hinton, 2010](#)). [Montavon and Muller \(2012\)](#) provides empirical evidence that the centering reparametrization improves the conditioning of the Hessian, thus improving the convergence rate of

first-order gradient descent.

The centering trick is closely related to data normalization for RBM training (Tang and Sutskever, 2011) and the skip-connections used in feed-forward MLPs (Schraudolph, 1998; Raiko et al., 2012).

2.6 Higher-Order Models

The RBM belongs to the family of Boltzmann Machines whose conditional distributions are linear in either v or h . Recently, multiple models have been proposed with energy terms which are quadratic in the visible units, such as the mean-covariance RBM (Ranzato et al., 2010; Ranzato and Hinton, 2010) and the spike & slab RBM (Courville et al., 2011a,b), allowing hidden units to capture both mean pixel intensities, as well as their variance. At the time of their publication, this increased representational capacity translated directly to state of the art results on natural image datasets such as CIFAR10 (Krizhevsky and Hinton, 2009).

2.6.1 Third-Order Boltzmann Machines

Higher-order Boltzmann Machines (BM) are another promising family of models, which allow for more complex relationships between random variables, through multiplicative interactions. They are “higher-order” in the sense that each weight connects to more than two random variables, allowing them to capture correlations between three or more units. The general form of a third-order binary RBM is given below:

$$E(v, g, h) = - \sum_{i=1}^{n_g} \sum_{j=1}^{n_h} \sum_{k=1}^{n_v} W_{ijk} v_k g_i h_j - \sum_{i=1}^{n_g} b_i g_i - \sum_{j=1}^{n_h} d_j h_j - \sum_{k=1}^{n_v} c_k v_k. \quad (2.29)$$

Random variables are grouped into three layers $v \in \{0, 1\}^{n_v}$, $g \in \{0, 1\}^{n_g}$, $h \in \{0, 1\}^{n_h}$ and interact through the weight 3-tensor $W \in \mathcal{R}^{n_v \times n_g \times n_h}$. $c \in \mathcal{R}^{n_v}$, $b \in \mathcal{R}^{n_g}$, $d \in \mathcal{R}^{n_h}$ are again offsets which account for the base-rate activity of the units. When both layers v and g are observed, with layer h being latent, the model

is referred to as a Gated RBM (G3RBM)ⁱ and has been used extensively to model image transformations (Memisevic and Hinton, 2007, 2010) as well as sequences (Taylor and Hinton, 2009). We explore two of its variants in the sections below.

2.6.2 Gated RBM

We can get a better insight into third-order RBMs by looking at what happens to the energy-function of Eq. (2.29) when we condition on one of the variables. For example, conditioning on g , we obtain the energy function:

$$E(v, h|g) = - \sum_{j=1}^{n_h} \sum_{k=1}^{n_v} W_{jk}^g v_k h_j - \sum_{j=1}^{n_h} d_j h_j - \sum_{k=1}^{n_v} c_k v_k \quad (2.30)$$

$$\text{with } W_{jk}^g = \sum_{i=1}^{n_g} W_{ijk} g_i$$

This is simply a “normal” (second-order) RBM, whose weights are modulated by the activations of g , as shown in Fig. 2.3 (left). Gated RBMs (Memisevic and Hinton, 2007) exploit this mechanism to model image transformations between an input image g and a transformed version v . By maximizing the conditional distribution $p(v, h|g)$, the G3RBM essentially learns an exponential mixture of RBMs: one for every image being conditioned on (albeit with shared parameters). Instead of modeling image features as in traditional RBMs, the G3RBM will thus learn to model the differences between inputs g and v .

Another way to view this, is to see what happens when conditioning on the hidden units (Fig. 2.3 (left)). A fixed configuration of h yields an RBM with weight matrix W^h , with entries $\sum_j W_{ijk} h_j$, which serves to model the joint distribution $p(g, v|h)$. The conditional activation of v will thus be given by $p(v_k = 1|g, h) = \text{sigmoid}(\sum_i W_{ik}^h g_i)$, which is a non-linear mapping from g to v .

In Memisevic and Hinton (2007), the G3RBM was successfully applied to modeling transformations, as well as extracting optical flow from natural videos. The quality of the mapping provided by the G3RBM can also be used to define a metric which is invariant to the learnt transformations $T(x)$. Indeed, if two images x and y are related through T , the euclidian distance between y and the “mapped” version of x , $\hat{y} \sim p(v|g = x)$, should be low (and high otherwise).

i. The acronym GRBM is usually reserved for the Gaussian RBM.

2.6.3 Factored Gated RBM

Unfortunately, the G3RBM suffers from a prohibitively high number of parameters, which limits its application to models with a small number of units. Fortunately, the G3RBM is in some sense, over-parametrized. Consecutive video frames in natural images are often related by a small number of locally coherent transformations, stemming from either global motion (i.e. from a pan/tilt of the camera) or the independent motion of objects within the scene. One can thus imagine encoding these transformations with much fewer parameters than $n_g \times n_h \times n_v$.

To this end, [Memisevic and Hinton \(2010\)](#) proposed a factored version of the G3RBM, where the dense weight tensor is replaced by the outer product of 3 low-rank matrices, as shown in Eq. (2.31). A graphical depiction of the model is shown in Fig. 2.3 (right).

$$E(v, g, h) = - \sum_{f=1}^{n_f} \left[\sum_{i=1}^{n_g} W_{if}^g g_i \right] \left[\sum_{j=1}^{n_h} W_{jf}^h h_j \right] \left[\sum_{k=1}^{n_v} W_{kf}^v v_k \right] \quad (2.31)$$

$$- \sum_{i=1}^{n_g} b_i g_i - \sum_{j=1}^{n_h} d_j h_j - \sum_{k=1}^{n_v} c_k v_k.$$

The weight tensor W is thus replaced by 3 weight matrices $W^g \in \mathcal{R}^{n_f \times n_g}$, $W^h \in \mathcal{R}^{n_f \times n_h}$ and $W^v \in \mathcal{R}^{n_f \times n_v}$. These project the inputs g and v into an n_f dimensional factor space, where they interact multiplicatively. Latent factors then encode a preference for a given pattern of activation of the factors. n_f thus controls the complexity of the learnable transformations, along with the computational complexity of the model.

[Memisevic and Hinton \(2010\)](#) have shown that such a model can learn the same types of transformations as a G3RBM, but with less parameters and at a fraction of the computational cost. While [Memisevic and Hinton \(2010\)](#) focused on simple global affine transformations, early results suggest that factored G3RBM can also learn more complex, local transformations: factors then specialize to model transformations in specific regions of the input space.

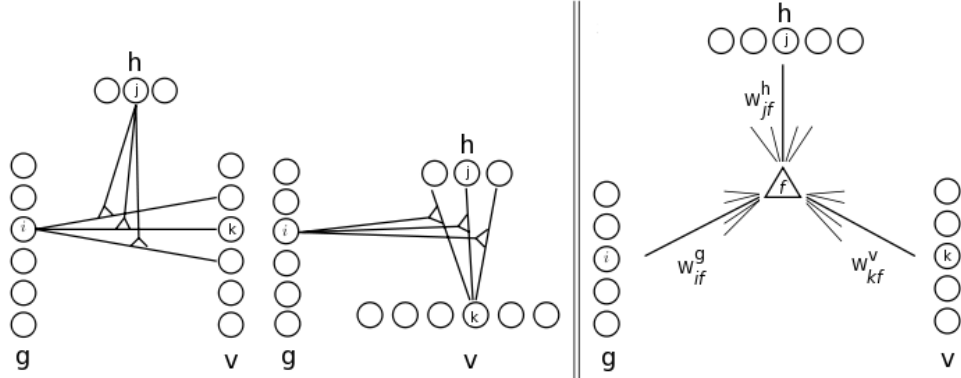


Figure 2.3: Graphical depiction of Gated RBM with (left) dense weight matrix and (right) factored weight matrix. Image reproduced with permission from [Memisevic and Hinton \(2010\)](#).

2.6.4 Bilateral Models of Data

While the G3RBM exploits the higher-order RBM terms to great success, they remain limited in the expressiveness of the model: just like RBMs, they represent data as linear combinations of latent factors. This is a severe limitation since, as was mentioned in Chapter 1, real world images are the result of complex interactions between many factors of variation (scene geometry, textures, lighting, etc.). Multiplicative interactions of latent factors thus seem interesting for their ability to enrich the model.

[Tenenbaum and Freeman \(2000\)](#) were the first to explore this question in the context of directed graphical models, followed by [Grimes and Rao \(2005\)](#); [Olshausen et al. \(2007\)](#). They put forward a bilinear generative model, wherein images are formed by the pairwise multiplicative interactions of two latent factors: one representing the “content”, with the other accounting for the “style” or the transformations present in the image. Borrowing the notation from [Grimes and Rao \(2005\)](#), an image \mathbf{z} of dimension K is formed as follows:

$$\mathbf{z} = f(x, y) = \sum_{i=1}^m \sum_{j=1}^n \mathbf{w}_{ij} x_i y_j, \quad (2.32)$$

where \mathbf{w}_{ij} is the K dimensional basis vector $\{\mathbf{w}_{ijk}; 1 \leq k \leq K\}$ and x_i, y_j are scalar coefficients which weigh the contribution of each basis in the final image. For a fixed value of \mathbf{y} , we can rewrite Eq. (2.32) as $\mathbf{z} = \sum_{i=1}^m W_i^y x_i$, where we

have defined $W^y = \sum_j \mathbf{w}_{ij} y_j$. Much like in the G3RBM, a different set of M basis vectors is thus obtained for each value of \mathbf{y} . Those are in turn linearly combined to form the final image \mathbf{z} . Under the assumption of additive Gaussian noise (with variance σ^2), parameters can then be learnt by maximizing the image likelihood given by $P(\mathbf{z}|\mathbf{x}, \mathbf{y}) = \mathcal{N}(\mathbf{z}; f(\mathbf{x}, \mathbf{y}), \sigma^2)$.

Unfortunately, this is a very difficult optimization problem: Grimes and Rao (2005) theorize that since there are many ways to decompose an image, $P(\mathbf{z}|\mathbf{x}, \mathbf{y})$ is plagued by local minima. Their solution is thus to add additional constraints on \mathbf{x} and \mathbf{y} : these codes should be sparse, meaning that only a few x_i 's and y_i 's should be active for a given image. The resulting **bilinear sparse coding** algorithm then corresponds to minimizing the following cost function:

$$E(\{\mathbf{w}_{ij}\}, \mathbf{x}, \mathbf{y}) = \left\| \mathbf{z} - \sum_{i=1}^m \sum_{j=1}^n \mathbf{w}_{ij} x_i y_j \right\|^2 + \alpha \sum_{i=1}^m S(x_i) + \beta \sum_{j=1}^n S(y_j), \quad (2.33)$$

where α and β are hyper-parameters, and $S(\mathbf{x})$ is a sparsity function which penalizes the activation of \mathbf{x} . As with normal (linear) sparse coding, the above can be minimized through the Expectation-Maximization (EM) algorithm), an iterative procedure which alternates (1) inferring the latent states given fixed values of the parameters and (2) minimizing Eq. (2.33) with respect to parameters, given the values of \mathbf{x} and \mathbf{y} obtained in step 1. The procedure is nevertheless adapted slightly to account for the bilinear encoding. A simplified version of the algorithm used in Grimes and Rao (2005) is shown in Algorithm 6.

When trained on natural images undergoing a series of translations, Algorithm 6 succeeds in learning a set localized edge detectors, whose location is controlled by \mathbf{y} . Of critical importance, the bilinear sparse coding generates a representation \mathbf{x} of the image which remains invariant to transformations of the input, while preserving the details of the transformation in \mathbf{y} .

Algorithm 6 Bilinear Sparse Coding Update

$\{T_s\}$: set of transformations we wish to model, $1 \leq s \leq r$.

T_c represents the identity transformation.

Randomly initialize parameters $W \in \mathcal{R}^{k \times m \times n}$.

Randomly initialize $Y \in \mathcal{R}^{n \times r}$, with $Y_s = \{Y_{js}\}$ representing transform T_s .

while not converged **do**

 Generate (mini)-batch of training examples $Z = \{z \in \mathcal{D}\}$.

 Infer X given parameters W , code Y_c and sparsity α .

for $s \in [1..r]$ **do**

 Generate transformed images $Z' = T_s(Z)$.

 Infer Y'_s given Z', X, W, α .

 Update Y_s as a moving average of Y'_s .

$W \leftarrow W - \frac{1}{r} \frac{\partial E(W, X, Y_s)}{\partial W}$.

 Normalize columns of Y_s .

end for

 Normalize basis vectors $\{\mathbf{w}_{ij}\}$.

end while

2.7 Estimating the Partition Function

Performing model selection can be particularly tricky for RBMs. Because the partition function is usually intractable, we lose the ability to compare models in terms of probabilities. Two options are thus available for choosing hyperparameters: using a surrogate criteria such as classification error to assess the quality of the learnt features, or using *estimates* of the partition function. In this section, we briefly review the most popular algorithms which can be used to this effect. In particular, Annealed Importance Sampling and Bridge Sampling will play a crucial role in Chapter 6, where we propose a novel algorithm for tracking the partition function of RBMs during learning.

2.7.1 Importance Sampling

Importance Sampling offers the basic tools for approximating the ratio of two partition functions. Let $p_1(x) = \frac{1}{Z_1} \tilde{p}_1(x)$ and $p_M(x) = \frac{1}{Z_M} \tilde{p}_M(x)$, we can rewrite

Z_1 as:

$$Z_1 = \sum_x \tilde{p}_1(x) = \sum_x p_M(x) \frac{\tilde{p}_1(x)}{p_M(x)} = \sum_x p_M(x) w(x) = \mathbb{E}_{p_M} [w(x)], \quad (2.34)$$

as long as $p_M(x) > 0$ whenever $\tilde{p}_1(x) > 0$, and where $w(x) = \tilde{p}_1(x)/p_M(x)$ are defined as the *importance weights*. Assuming Z_M is known, that we can easily sample from p_M and denoting $\mathcal{X}_M = \{x_n \sim p_M(x); 1 \leq n \leq N\}$, we can estimate Z_1 as follows:

$$Z_1 \approx \frac{1}{|\mathcal{X}_M|} \sum_{x \in \mathcal{X}_M} w(x) \Leftrightarrow \quad (2.35)$$

$$\log Z_1 \approx \log Z_M + \log \sum_{x \in \mathcal{X}_M} \frac{\tilde{p}_1(x)}{\tilde{p}_M(x)} - \log |\mathcal{X}_M| \quad (2.36)$$

Unfortunately, [Minka \(2005\)](#) showed that minimizing the variance of the importance sampling estimate of Z_1 is equivalent to minimizing a divergence of p_M and p_1 . This means that the above procedure will therefore work well if $p_M \approx p_1$, making it much less attractive in practice.

2.7.2 Annealed Importance Sampling

To get around this limitation, one can define a set of intermediate distributions $p_i(x) = \frac{1}{Z_i} \tilde{p}_i(x)$, which “bridge the gap” between p_1 and p_M , as first described in [Neal \(2001\)](#). This will enable us to write the ratio Z_1/Z_M as the product of importance weights, measured on neighboring chains (p_i, p_{i+1}) . We start with the importance sampling identity for chains (p_1, p_2) :

$$Z_1 = \sum_{x_1} p_2(x_1) \frac{\tilde{p}_1(x_1)}{p_2(x_1)} \quad (2.37)$$

If T_i is a transition operator which leaves p_i invariant (such as the Gibbs sampling operator), then by definition $p_2(x_1) = \sum_{x_2} p_2(x_2) T_2(x_1; x_2)$. This allows us

to rewrite Eq. 2.37 as:

$$\begin{aligned}
Z_1 &= \sum_{x_1} \sum_{x_2} p_2(x_2) T_2(x_1; x_2) \frac{\tilde{p}_1(x_1)}{p_2(x_1)} \\
&= \sum_{x_1} \sum_{x_2} \frac{\tilde{p}_2(x_2)}{Z_2} T_2(x_1; x_2) \tilde{p}_1(x_1) \frac{Z_2}{\tilde{p}_2(x_1)} \\
&= \sum_{x_1} \sum_{x_2} p_3(x_2) T_2(x_1; x_2) \frac{\tilde{p}_2(x_2)}{p_3(x_2)} \frac{\tilde{p}_1(x_1)}{\tilde{p}_2(x_1)}
\end{aligned}$$

Recursively (and shifting the indices of x_i to the right by one), this leads to:

$$\begin{aligned}
Z_1 &= \sum_{x_2} \sum_{x_3} \cdots \sum_{x_M} p_M(x_M) T_{M-1}(x_{M-1}; x_M) \cdots T_2(x_2; x_3) \\
&\quad \frac{\tilde{p}_{M-1}(x_M)}{p_M(x_M)} \frac{\tilde{p}_{M-2}(x_{M-1})}{\tilde{p}_{M-1}(x_{M-1})} \cdots \frac{\tilde{p}_2(x_3)}{\tilde{p}_3(x_3)} \frac{\tilde{p}_1(x_2)}{\tilde{p}_2(x_2)}. \tag{2.38}
\end{aligned}$$

Replacing the above expectations with a sample average, the AIS estimate of the log-partition is then given by:

$$\begin{aligned}
\log Z_1 &\approx \log Z_M + \frac{1}{N} \sum_{n=1}^N w^{(n)}, \text{ with} \\
w^{(n)} &= \frac{\tilde{p}_{M-1}(x_M^{(n)})}{\tilde{p}_M(x_M^{(n)})} \frac{\tilde{p}_{M-2}(x_{M-1}^{(n)})}{\tilde{p}_{M-1}(x_{M-1}^{(n)})} \cdots \frac{\tilde{p}_2(x_3^{(n)})}{\tilde{p}_3(x_3^{(n)})} \frac{\tilde{p}_1(x_2^{(n)})}{\tilde{p}_2(x_2^{(n)})}. \tag{2.39}
\end{aligned}$$

The variance of this estimator is:

$$\sigma_{AIS}^2 \approx \frac{\text{Var}[w^{(n)}]}{[\sum_n w^{(n)}]^2}. \tag{2.40}$$

Once again, we use $x_i^{(n)}$ to denote the n -th sample of $p_i(x)$. eq 6.2 is at the basis of the Annealed Importance Sampling (AIS) procedure (Neal, 2001), which is presented as Algorithm 7. Salakhutdinov and Murray (2008) shows how AIS can be applied to estimating the partition function of an RBM.

Algorithm 7 Annealed Importance Sampling

for $j \in [1..N]$ **do**
 sample $x_M \sim p_M$.
 sample $x_{M-1} \sim T_{M-1}$.
 ...
 sample $x_2 \sim T_2$.
 compute $w^{(j)} = \frac{\tilde{p}_{M-1}(x_M) \tilde{p}_{M-2}(x_{M-1}) \dots \tilde{p}_2(x_3) \tilde{p}_1(x_2)}{\tilde{p}_M(x_M) \tilde{p}_{M-1}(x_{M-1}) \dots \tilde{p}_3(x_3) \tilde{p}_2(x_2)}$
end for
 $\log Z_1 \approx \log Z_M + \log \sum_{j=1}^N w^{(j)} - \log N$.

2.7.3 Bridge Sampling

Bridge sampling (Bennett, 1976) addresses the shortcomings of importance sampling in a slightly different manner. It relies on a single distribution p_* , which interpolates between p_M (pdf with known partition function Z_M) and p_1 (pdf whose partition function Z_1 we wish to estimate). Given the sample sets $\mathcal{X}_M = \{x_n \sim p_M(x); 1 \leq n \leq N\}$ and $\mathcal{X}_1 = \{x'_n \sim p_1(x); 1 \leq n \leq N\}$, the difference in the log-partition functions is simply estimated by the difference of log-importance weights between each distribution and the bridge.

$$\begin{aligned}
 \log Z_1 &\approx \log Z_M + \log \left[\frac{\sum_{n=1}^N \tilde{p}_*(x_n)}{\tilde{p}_M(x_n)} \right] - \log \left[\frac{\sum_{n=1}^N \tilde{p}_*(x'_n)}{\tilde{p}_1(x'_n)} \right] \\
 &= \log Z_M + \log \left[\sum_{n=1}^N u_n \right] - \log \left[\sum_{n=1}^N v_n \right],
 \end{aligned} \tag{2.41}$$

with $u_n = \tilde{p}_*(x_n)/\tilde{p}_M(x_n)$ and $v_n = \tilde{p}_*(x'_n)/\tilde{p}_1(x'_n)$. The variance of this estimator of $\log Z_1$ is then given by:

$$\sigma_{bridge}^2 \approx \frac{\text{Var}[u_n]}{[\sum_n u_n]^2} + \frac{\text{Var}[v_n]}{[\sum_n v_n]^2} \tag{2.42}$$

This formulation is much more forgiving and allows for $\text{KL}(p_M \| p_1)$ to be much larger than required by normal importance sampling, or even AIS when computing importance weights across neighboring chains. This is because p_* is chosen in some optimal fashion to have large support both with p_M and p_1 . One such optimal distribution is $p_*^{(opt)}(x) \propto \frac{\tilde{p}_M(x)\tilde{p}_1(x)}{r\tilde{p}_M(x)+\tilde{p}_1(x)}$ where $r = Z_1/Z_M$. This definition appears

circular however: to estimate Z_1 we should use the optimal bridge distribution $p_*^{(opt)}$, which is itself a function of Z_1 . Fortunately, Neal (2005) showed that it is possible to start with a coarse estimate of r and use the resulting bridge distribution to refine the estimate of Z_1 recursively.

2.8 Non-probabilistic Generative Models

Maximum likelihood learning of probabilistic models is a central pillar of this thesis. Our focus on the ML estimator (MLE) stems both from efficiency guaranteesⁱ, as well as empirical results (Marlin et al., 2010). In practice however, the intractability of the partition function requires us to use (stochastic) approximations to MLE, which invalidates any theoretical guarantees of efficiency. As such, this motivates exploring alternative *inductive principles* which bypass issues linked to the partition function. Alternatively, we may also use entirely new model families, which make no inherent assumption of normalization.

2.8.1 Score Matching

Score Matching (Hyvärinen, 2005) is one such alternative inductive principle. It defines a score function $\psi_i(p, x)$, which does not depend on the partition function of pdf p , and then minimizes a cost J_θ such that $\psi_i(p, x) \approx \psi_i(\pi, x), \forall x \in \mathcal{D}$. In particular, Hyvärinen (2005) defines:

$$\psi_i(p, v) = \frac{\partial \log p(v)}{\partial v_i} \quad (2.43)$$

$$J_{SM'}(\theta) = \mathbb{E}_{\pi(v)} \left[\sum_{i=1}^{n_v} \frac{1}{2} (\psi_i(\pi, v) - \psi_i(p, v))^2 \right]. \quad (2.44)$$

Since we cannot compute $\psi_i(\pi, v)$, $J_{SM'}(\theta)$ is not very useful in practice. Through partial integration however, it can be shown that

i. MLE achieves the Cramer-Rao bound, meaning that as the sample size goes to infinity, it obtains the lowest possible mean squared error of any unbiased estimator. Note that this property does not necessarily hold for stochastic ML used throughout this thesis.

$\arg \min_{\theta} J_{SM'}(\theta) \equiv \arg \min_{\theta} J_{SM}(\theta)$, where J_{SM} is defined as:

$$J_{SM}(\theta) = \mathbb{E}_{\pi(v)} \left[\sum_{i=1}^{n_v} \frac{1}{2} \psi_i(p_{\theta}, v)^2 + \frac{\partial \psi_i(p_{\theta}, v)}{\partial v_i} \right]. \quad (2.45)$$

One can thus minimize the mean squared error between the score of the empirical and model distributions by minimizing J_{SM} . Minimizing Eq. 2.45 with a sample average in lieu of an expectation, aims to learn model configurations which locally maximize the log-probability around training points. While penalizing the square of the score at training points could lead to either local minima or maxima of the log-probability, the second term penalizes positive values for the second derivative hence promoting critical points which correspond to local maxima of log-probability.

Score matching has been shown to be a consistent estimatorⁱ much like maximum likelihood (Hyvärinen, 2005). Score Matching has also been extended in a number of ways. Ratio matching is the analogous principle when working with binary data (Hyvärinen, 2007), and has been successfully applied to RBMs in Dauphin and Bengio (2013) when training on sparse high-dimensional inputs. Note however, that these methods remain somewhat limited for applications to deep networks, as they require an analytically tractable free-energy function, which is usually not the case for deep networks. Other methods in this family including Noise Contrastive Estimation (NCE) (Gutmann and Hyvarinen, 2010), Generalized NCE (Gutmann and Hirayama, 2011) and Margin Learning (Weston et al., 2008).

2.8.2 Auto-Encoders

Autoencoders on the other hand are deterministic neural networks, which are trained via backpropagation to reconstruct the input itself. The function $f_{\theta}(x)$ implemented by an autoencoder is typically broken down into two steps: an *encoding function* $enc(x)$ which maps the input from \mathbb{R}^D to \mathbb{R}^{N_h} , and a *decoding function* $dec(h)$ which maps back into input space, such that $r(x) = dec(enc(x)) \in \mathbb{R}^D$. While encoders and decoders may have an arbitrary rich parametrization in prac-

i. Assuming the data was generated by model p_{θ^*} , the consistency claims that $\theta = \theta^* = \arg \min_{\theta} J_{SM}(\theta)$.

tice (i.e. deep MLPs), the prototypical “shallow” auto-encoder is defined as:

$$\begin{aligned} h &:= \text{enc}(x) = f_e(x^T W_e + b) \\ \hat{x} &:= \text{dec}(h) = f_d(h^T W_d + c), \end{aligned} \tag{2.46}$$

where $W_e \in \mathbb{R}^{D \times N_h}$, $W_d \in \mathbb{R}^D$ are the encoding and decoding weight matrices (respectively), b and c are offset vectors and f_e and f_d are activations functions. Typically, encoder and decoder share the same weight matrix with $W_d = W_e^T$.

As is typically done with neural networks, activation and loss functions are matched to deal with the input distribution: for continuous data, one typically optimizes the mean-squared error with f_d the identity function, while for binary data, we use the cross-entropy loss with $f_d(x) = \text{sigmoid}(x)$. The encoding function $f_e(x)$ admits a variety of non-linearities, but is most commonly chosen to be sigmoidal or of the rectified linear variety. *Deep auto-encoders* extend encoders and decoders to have multiple levels of non-linearities, allowing them to capture high-level abstract features of the input. Autoencoders have been used with great success in a multitude of settings: for classification using the features extracted by the encoder as input to a classifier, as pretraining for deep MLPs (Bengio et al., 2007), as a dimensionality reduction method (Hinton and Salakhutdinov, 2006) and finally for image denoising applications (including structured noise) (Hinton and Salakhutdinov, 2006; Cho, 2013).

Sparsity Regularization . When working with auto-encoders, special care must be taken to regularize the reconstruction function $r(x)$. Indeed if $N_h > D$ and f_e is linear, then the training loss can be minimized trivially by setting $W_e = I$. Aside from $L1$ and $L2$ regularization, one popular method for regularizing autoencoders is to enforce h to be high-dimensional but sparse, so as to model v as a linear combination of a limited subset of basis filters (i.e. columns of the W_e matrix). This is similar to sparse coding and can be achieved by introducing the following regularization term to the loss function, $\lambda \sum_j |h_j|$.

Denoising Auto-Encoders (DAE) (Vincent et al., 2008) prevent learning the identity function by asking the autoencoder to reconstruct the input from a noisy or corrupted version of itself. Formally, an autoencoder introduces a lossy corruption

process $\mathcal{C}(\tilde{x} | x)$ and minimizes the loss $\mathcal{L}(r(\tilde{x}), x)$, where \mathcal{L} can be the mean-square or cross-entropy loss. For binary inputs, popular choices for \mathcal{C} include *masking noise* (which sets each bit x_i to 0 with some probability), *salt & pepper noise* (which randomly sets bits to either 1 or 0) and additive Gaussian noise when working with continuous inputs.

Contractive Auto-Encoders (CAE) (Rifai et al., 2011) introduce a regularization term which penalizes, for similar inputs, deviations in hidden unit activations. Concretely, this is achieved by penalizing the Frobenius norm of the encoder function’s Jacobian, i.e. by minimizing the loss function below:

$$\mathcal{L}_{CAE} = \mathcal{L}(r(x), x) + \lambda \sum_{i,j} \left(\frac{\partial h_j}{\partial x_i} \right)^2. \quad (2.47)$$

The above trade-off between reconstruction error and invariance in the encoding yields encoding functions whose Jacobians have fewer large singular values. This led the authors to hypothesize that the contraction penalty encourages CAEs to learn the leading directions of variation at any point along the data manifold, i.e. the tangent vectors of the manifold.

2.8.3 Auto-Encoders as Generative Models

While auto-encoders have been very successful at performing feature extraction or as pretraining for deep MLPs, one commonly perceived flaw is that they fail to learn a proper density model of the empirical distribution. Score matching provided the first piece of evidence linking auto-encoders to energy-based models. Vincent (2011) showed that optimizing the DAE objective was equivalent (under some conditions) to applying a regularized form of score matching to a Gaussian RBMs. Independently, score matching of the GRBM energy function was shown in Swersky et al. (2011), to lead to a loss function almost identical to Eq.2.47, when using \mathcal{L}_{MCE} and a linear reconstruction. The only difference is the lack of hyper-parameter λ , and a discrepancy in the squaring of the $h_j(1 - h_j)$ term. ⁱ

Progress in this direction is moving dramatically. Alain and Bengio (2013)

i. $h_j(1 - h_j)$ stems from computing $\frac{\partial h_j}{\partial x}$ when h has a sigmoidal activation function.

proved that when minimizing the DAE loss under some conditions ⁱ, one could recover the score of an underlying pdf by computing $(r(x) - x)/\sigma^2$ (where σ is the standard-deviation of the noise). In the general setting of $r(x)$ being parametric or when $r(x)$ does not correspond to the derivative of an energy function, one can still recover the underlying pdf via sampling. In this earlier work, [Alain and Bengio \(2013\)](#) proposed using Metropolis-Hastings with a local Gaussian kernel and using the vector field $r(x) - x$ to compute an approximate acceptance ratio.

Generalized DAEs ([Bengio et al., 2013](#)) represent a new framework linking auto-encoders to density models (which holds for general input and noise distributions) and constitute a new inductive principle for learning probabilistic density models. Given a noise or corruption process $\mathcal{C}(\tilde{x} | x)$, an auto-encoder can be thought as learning the conditional probability $p_\theta(x | \tilde{x})$. One can then recover the underlying pdf by simulating a Markov chain, which alternates sampling $\tilde{x}_t \sim \mathcal{C}(\tilde{x} | x = x_{t-1})$ and $x_t \sim p_\theta(x | \tilde{x} = \tilde{x}_t)$. The ergodicity of this chain is proven under mild assumptions of the corruption process: if $p_\theta(x|\tilde{x})$ is a consistent estimator of the true conditional $p(x|\tilde{x})$, then the stationary distribution is a consistent estimator of $p(x)$. [Bengio et al. \(2013\)](#) further generalize this concept by augmenting the state of the Markov chain to include latent variables. A GSN is defined by the following two conditionals: $h_t \sim p(h|h = h_{t-1}, x = x_{t-1}) := f(x_{t-1}, h_{t-1}, \epsilon)$ and $x_t \sim p(x|h = h_{t-1})$. Here f is an arbitrary non-linear function, e.g. an MLP, and σ is a noise random variable. The latent variables affords GSNs the ability to learn complex multi-layered distributions, while enabling faster mixing of the above Markov chain. Being stochastic feed-forward models, GSNs also side-step issues of intractable inference typically encountered with probabilistic models having multiple layers of latent variables, like the Deep Boltzmann machine.

2.8.4 Predictive Sparse Decomposition (PSD)

Inference in RBMs is both exact and trivial, but comes at the expense of an intractable partition function. In contrast, directed models like sparse coding have a tractable gradient but require a computationally expensive inference process, which makes it ill-suited for feature extraction.

ⁱ. Mean-squared error, continuous inputs, non-parametric reconstruction function $r(x)$ (i.e. having infinite capacity) and in the limit of zero-noise.

Predictive Sparse Decomposition (PSD) (Kavukcuoglu et al., 2008) represents an interesting compromise between both approaches. PSD starts from the loss function optimized by sparse-codingⁱ and adds an additional term which penalizes configurations of latent variables which are not well predicted by an encoding function $enc(x, \theta_e)$. The PSD loss function is given by:

$$\mathcal{L}_{PSD}(x, h, W, \theta_e) = \|x - Wh\|^2 + \alpha \|Z - enc(x, \theta_e)\|^2 + \lambda|h|. \quad (2.48)$$

While training still requires an expensive iterative process, at test time, one can simply use the learnt encoder to perform approximate inference. The resulting features were found to yield better classification accuracy when fed to an object recognition pipeline (compared to exact inference) and at a much lower computational cost. While Kavukcuoglu et al. (2008) proposed parametrizing $enc(x)$ as an MLP, Gregor and LeCun (2010) proposed using a (truncated) recursive neural network whose particular structure mimics the inference process of sparse coding.

i. Sparse coding using hard-EM and a Laplace prior on h can be seen as alternatively minimizing the loss $\mathcal{L}_{SC}(x, h, W) = \|x - Wh\|^2 + \lambda|h|$: once for h given fixed x and W , and once for W given fixed x and h .

3

Prologue to First Article

3.1 Article Details

Adaptive Parallel Tempering for Stochastic Maximum Likelihood Learning of RBMs. Guillaume Desjardins, Aaron Courville and Yoshua Bengio.

Presented at the *Deep Learning and Unsupervised Feature Learning Workshop*, of the *24th Annual Conference on Neural Information Processing Systems (NIPS 2010)*.

Personal Contribution. This workshop paper is the result of a close collaboration with my co-authors. The theoretical contributions of the adaptive tempering scheme were developed jointly with Aaron Courville. I implemented the algorithm, datasets and performed all of the experiments reported in the paper. I also contributed heavily to the writing (especially sections regarding algorithmic details and the experiments section), with Aaron Courville and Yoshua Bengio providing the introductory and background materials.

3.2 Context

This paper is part of larger corpus of work aimed at using tempering methods, to more efficiently and accurately estimate the sufficient statistics of BMs.

This paper is a follow-up to [Desjardins et al. \(2010b\)](#), which was first to propose using Parallel Tempering (PT) to estimate the negative phase statistics of RBMs. This work showed how the increased ergodicity of the PT sampler could yield a more robust training procedure along with faster (per update) convergence.

Concurrently, [Salakhutdinov \(2010b\)](#) proposed using the Tempered Transitions (TT) algorithm of [Neal \(1994\)](#) to draw samples of the BM. This can be considered

as the serial implementation of our PT approach: while PT simulates M parallel chains in parallel, TT simulates a single chain whose temperature is adjusted gradually. Each iteration is therefore cheaper, in terms of both memory and runtime, however more iterations are required before obtained a new sample at the nominal temperature. Both methods also suffer from similar drawbacks: if the set of temperatures is not chosen properly, PT can suffer from low swap rates between chains at neighboring temperatures, while TT can suffer from a high rejection ratio.

The PT approach was also concurrently developed in [Cho et al. \(2010\)](#). In contrast to our earlier work which showed improved per-update convergence, this paper also showed that the method was efficient from a computational perspective.

The paper presented in [Chapter 4](#) addresses some of the main shortcomings of these methods: automating the choice of temperatures to simulate and reducing the computational cost of the method.

3.3 Contributions

The success of the above methods relies crucially on the choice of temperatures \mathcal{T} to simulate. Unfortunately, no closed form solution exists for selecting the optimal set of temperatures in the general BM setting. As such, the choice of \mathcal{T} is left as a hyper-parameter, which must be tuned through trial and error. The difficulty of this process is exacerbated in the context of learning, where we expect the optimal setting \mathcal{T}^* to change as the model parameters are updated.

The paper presented in [Chapter 4](#) provides a method for automatically tuning the set of temperatures based on the principle of *return-time minimization* ([Katzgraber et al., 2006](#)). Starting from a uniform spacing, temperatures are adapted in order to minimize the amount of time required for a sample to perform a round-trip between the lower and highest temperatures. Compared to the naive approach where temperatures are adapted to optimize local swap statistics, this procedure optimizes the temperatures using a global criterion, which increases the ergodicity of the sampler.

Our algorithm, dubbed SML with Adaptive Parallel Tempering (SML-APT) also introduced a method for dynamically adjusting the number of chains under simulation. This process gradually increases the computational complexity of the sampling algorithm, as the complexity of the model distribution increases.

3.4 Recent Developments

While our temperature adaptation scheme works well in practice, its underlying principle remains a heuristic. While the return-time minimization is intuitively attractive, it does not prevent a degenerate scenario where samples constantly “return” to the same mode, thus preventing the sampler from exploring the modes of the distribution. While this issue was not observed in practice, it is a rather fundamental issue which stems from using a fixed set of temperatures \mathcal{T} , which is independent of the state of the sampler (i.e. the current set of samples). As such, the issue of adaptive tempering methods is far from solved.

Recently, Salakhutdinov (2010a) proposed an alternative adaptive tempering method, which combines Simulated Tempering (Marinari and Parisi, 1992) with the Wang-Landau (WL) algorithm (Wang and Landau, 2001). Adaptive Simulated Tempering (AST) samples from the joint distribution $p(x, k) = w_k \exp(-\beta_k E(x))$, where k is a temperature index. The sampler is updated in two steps: (1) for a fixed value of k , x is updated via Gibbs sampling (2) for fixed x , a Metropolis-Hastings (MH) move is then proposed to increase or decrease the temperature, as a function of the weights w . Crucially, these are adapted so as to guarantee that each sample spends “equal time” on average at each temperature, a method known as the *flat-histogram* method. The method is thus similar in spirit to our approach: samples are encouraged to burn-in at high temperatures, while the WL weights or temperatures (in SML-APT) are adapted to guarantee that these fast-mixing samples regularly visit the nominal temperature.

4

Adaptive Parallel Tempering for Stochastic Maximum Likelihood Learning of RBMs

RESTRICTED BOLTZMANN MACHINES (RBM) have attracted a lot of attention of late, as one of the principle building blocks of deep networks. Training RBMs remains problematic however, because of the intractability of their partition function. The maximum likelihood gradient requires a very robust sampler which can accurately sample from the model despite the loss of ergodicity often incurred during learning. While using Parallel Tempering in the negative phase of Stochastic Maximum Likelihood (SML-PT) helps address the issue, it imposes a trade-off between computational complexity and high ergodicity, and requires careful hand-tuning of the temperatures. In this paper, we show that this trade-off is unnecessary. The choice of optimal temperatures can be automated by minimizing average return time (a concept first proposed by [Katzgraber et al. \(2006\)](#)) while chains can be spawned dynamically, as needed, thus minimizing the computational overhead. We show on a synthetic dataset, that this results in better likelihood scores.

4.1 Introduction

Restricted Boltzmann Machines (RBM) ([Freund and Haussler, 1994](#); [Welling et al., 2005](#); [Hinton et al., 2006](#)) have become a model of choice for learning unsupervised features for use in deep feed-forward architectures ([Hinton et al., 2006](#); [Bengio, 2009](#)) as well as for modeling complex, high-dimensional distributions ([Welling et al., 2005](#); [Taylor and Hinton, 2009](#); [Larochelle et al., 2010](#)). Their success can be explained in part through the bipartite structure of their graphical model. Units are grouped into a visible layer \mathbf{v} and a hidden layer $h^{(1)}$, prohibiting connections within the same layer. The use of latent variables affords RBMs a rich modeling capacity, while the conditional independence property yields a trivial inference

procedure.

RBM's are parametrized by an energy function $\mathbf{E}(\mathbf{v}, h^{(1)})$ which is converted to probability through the Boltzmann distribution, after marginalizing out the hidden units. The probability of a given configuration $p(\mathbf{v})$ is thus given by $p(\mathbf{v}) = \frac{1}{Z} \sum_{h^{(1)}} \exp(-\mathbf{E}(\mathbf{v}, h^{(1)}))$, where Z is the partition function defined as $Z = \sum_{\mathbf{v}, h^{(1)}} \exp(-\mathbf{E}(\mathbf{v}, h^{(1)}))$.

Despite their popularity, direct learning of these models through maximum likelihood remains problematic. The maximum likelihood gradient with respect to the parameters θ of the model is:

$$\frac{\partial \log p(\mathbf{v})}{\partial \theta} = - \sum_{h^{(1)}} p(h^{(1)}|\mathbf{v}) \frac{\partial \mathbf{E}(\mathbf{v}, h^{(1)})}{\partial \theta} + \sum_{\mathbf{v}^-, \mathbf{h}^-} p(\mathbf{v}^-, \mathbf{h}^-) \frac{\partial \mathbf{E}(\mathbf{v}^-, \mathbf{h}^-)}{\partial \theta} \quad (4.1)$$

The first term is trivial to calculate and is referred to as the **positive phase**, as it raises the probability of training data. The second term or **negative phase** is intractable in most applications of interest, as it involves an expectation over $p(\mathbf{v}, h^{(1)})$. Many learning algorithms have been proposed in the literature to address this issue:

- Contrastive Divergence (CD) (Hinton, 1999, 2002) replaces the expectation with a finite set of negative samples, which are obtained by running a short Markov chain initialized at positive training examples. This yields a biased, but low-variance gradient which has been shown to work well as a feature extractor for deep networks such as the Deep Belief Network (Hinton et al., 2006).
- Stochastic Maximum Likelihood (SML) or Persistent Contrastive Divergence (PCD) (Younes, 1998; Tieleman, 2008) on the other hand, relies on a persistent Markov chain to sample the negative particles. The chain is run for a small number of steps between consecutive model updates, with the assumption that the Markov chain will stay close to its equilibrium distribution as the parameters evolve. Learning actually encourages this process, in what is called the “fast-weight effect” (Tieleman and Hinton, 2009).
- Ratio Matching and Score Matching (Hyvärinen, 2005, 2007) avoid the issue of the partition function altogether by replacing maximum likelihood by another learning principle, based on matching the change in likelihood to that implied by the empirical distribution.

Marlin et al. (2010) recently compared these algorithms on a variety of tasks and found SML to be the most attractive method when taking computational complexity into account. Unfortunately, these results fail to address the main shortcomings of SML. First, it relies on Gibbs sampling to extract negative samples: a poor choice when sampling from multi-modal distributions. Second, to guarantee convergence, the learning rate must be annealed throughout learning in order to offset the loss of ergodicityⁱ incurred by the Markov chain due to parameter updates (Younes, 1998; Desjardins et al., 2010b). Using tempering in the negative phase of SML (Desjardins et al., 2010b; Cho et al., 2010; Salakhutdinov, 2010b,a) appears to address these issues to some extent. By performing a random walk in the joint (configuration, temperature) space, negative particles can escape regions of high probability and travel between disconnected modes. Also, since high temperature chains are inherently more ergodic, the sampler as a whole exhibits better mixing and results in better convergence properties than traditional SML.

Tempering is still no panacea however. Great care must be taken to select the set of temperatures $\mathcal{T} = \{T_1, \dots, T_M; T_1 < T_i < T_M \forall i \in [1, M], M \in \mathbb{N}\}$ over which to run the simulation. Having too few or incorrectly spaced chains can result in high rejection ratios (tempered transition), low return rates (simulated tempering) or low swap rates between neighboring chains (parallel tempering), which all undermine the usefulness of the method. In this work, we show that the choice of \mathcal{T} can be automated for parallel tempering, both in terms of optimal temperature spacing, as well as the number of chains to simulate. Our algorithm relies heavily on the work of Katzgraber et al. (2006), who were the first to show that optimal temperature spacing can be obtained by minimizing the average **return time** of particles under simulation.

The paper is organized as follows. We start with a brief review of SML, then explore the details behind SML with Parallel Tempering (SML-PT) as described in Desjardins et al. (2010b). Following this, we show how the algorithm of Katzgraber et al. can be adapted to the online gradient setting for use with SML-PT and show how chains can be created dynamically, so as to maintain a given level of ergodicity throughout training. We then proceed to show various results on a complex synthetic dataset.

i. We use the term “ergodicity” rather loosely, to reflect the amount of time required for the states sampled by the Markov chain, to reflect the true expectation we wish to measure.

4.2 SML with Optimized Parallel Tempering

4.2.1 Parallel Tempered SML (SML-PT)

We start with a very brief review of SML, which will serve mostly to anchor our notation. For details on the actual algorithm, we refer the interested reader to Tieleman and Hinton (2009); Marlin et al. (2010). RBMs are parametrized by $\theta = \{W^{(1)}, \mathbf{b}, \mathbf{c}\}$, where \mathbf{b}_i is the i -th hidden bias, \mathbf{c}_j the j -th visible bias and W_{ij} is the weight connecting units h_i to v_j . They belong to the family of log-linear models whose energy function is given by $\mathbf{E}(\mathbf{x}) = -\sum_k \theta_k \phi_k(\mathbf{x})$, where ϕ_k are functions associated with each parameter θ_k . In the case of RBMs, $\mathbf{x} = (\mathbf{v}, h^{(1)})$ and $\phi(\mathbf{v}, h^{(1)}) = (h^{(1)}\mathbf{v}^T, h^{(1)}, \mathbf{v})$. For this family of model, the gradient of Equation 4.1 simplifies to:

$$\frac{\partial \log p(\mathbf{v})}{\partial \theta} = \mathbb{E}_{p(h^{(1)}|\mathbf{v})}[\phi(\mathbf{v}, h^{(1)})] - \mathbb{E}_{p(\mathbf{v}, h^{(1)})}[\phi(\mathbf{v}, h^{(1)})]. \quad (4.2)$$

As was mentioned previously, SML approximates the gradient by drawing negative phase samples (i.e. to estimate the second expectation) from a persistent Markov chain, which attempts to track changes in the model. If we denote the state of this chain at time step t as \mathbf{v}_t^- and the i -th training example as $\mathbf{v}^{(i)}$, then the stochastic gradient update follows $\phi(\mathbf{v}^{(i)}, h^{(1)}) - \phi(\tilde{\mathbf{v}}_{t+k}^-, h^{(1)}_{t+k})$, where $h^{(1)} = \mathbb{E}[h^{(1)}|\mathbf{v} = \mathbf{v}^{(i)}]$, and $\tilde{\mathbf{v}}_{t+k}^-$ is obtained after k steps of alternating Gibbs starting from state \mathbf{v}_t^- and $h^{(1)}_{t+k} = \mathbb{E}[h^{(1)}|\mathbf{v} = \mathbf{v}_{t+k}^-]$.

Training an RBM using SML-PT maintains the positive phase as is. During the negative phase however, we create and sample from an extended set of M persistent chains, $\{p_{\beta_i}(\mathbf{v}, h^{(1)}) | i \in [1, M], \beta_i \geq \beta_j \iff i < j\}$. Here each $p_{\beta_i}(\mathbf{v}, h^{(1)}) = \frac{\exp(-\beta_i \mathbf{E}(\mathbf{x}))}{Z(\beta_i)}$ represents a smoothed version of the distribution we wish to sample from, with the inverse temperature $\beta_i = 1/T_i \in [0, 1]$ controlling the degree of smoothing. Distributions with small β values are easier to sample from as they exhibit greater ergodicity.

After performing k Gibbs steps for each of the M intermediate distributions, cross-temperature state swaps are proposed between neighboring chains using a Metropolis-Hastings-based swap acceptance criterion. If we denote by \mathbf{x}_i the joint state (visible and hidden) of the i -th chain, the swap acceptance ratio r_i for swap-

ping chains $(i, i + 1)$ is given by:

$$r_i = \max\left(1, \frac{p_{\beta_i}(\mathbf{x}_{i+1})p_{\beta_{i+1}}(\mathbf{x}_i)}{p_{\beta_i}(\mathbf{x}_i)p_{\beta_{i+1}}(\mathbf{x}_{i+1})}\right) \quad (4.3)$$

Although one might reduce variance by using free-energies to compute swap ratios, we prefer using energies as the above factorizes nicely into the following expression:

$$r_i = \exp((\beta_i - \beta_{i+1}) \cdot (\mathbf{E}(\mathbf{x}_i) - \mathbf{E}(\mathbf{x}_{i+1}))), \quad (4.4)$$

While many swapping schedules are possible, we use the Deterministic Even Odd algorithm (DEO) (Lingenheil et al., 2009), described below.

4.2.2 Return Time and Optimal Temperatures

Conventional wisdom for choosing the optimal set \mathcal{T} has relied on the “flat histogram” method which selects the parameters β_i such that the pair-wise swap ratio r_i is constant and independent of the index i . Under certain conditions (such as when sampling from multi-variate Gaussian distributions), this can lead to a geometric spacing of the temperature parameters (Neal, 1994). Behrens et al. (2010) has recently shown that geometric spacing is actually optimal for a wider family of distributions characterized by $\mathbb{E}_\beta(\mathbf{E}(x)) = K_1/\beta + K_2$, where \mathbb{E}_β denotes the expectation over inverse temperature and K_1, K_2 are arbitrary constants.

Since this is clearly not the case for RBMs, we turn to the work of Katzgraber et al. (2006) who propose a novel measure for optimizing \mathcal{T} . Their algorithm directly maximizes the ergodicity of the sampler by minimizing the time taken for a particle to perform a round-trip between β_1 and β_M . This is defined as the average “return time” τ_{rt} . The benefit of their method is striking: temperatures automatically pool around phase transitions, causing spikes in local exchange rates and maximizing the “flow” of particles in temperature space.

The algorithm works as follows. For N_s sampling updates:

- assign a label to each particle: those swapped into β_1 are labeled as “up” particles. Similarly, any “up” particle swapped into β_M becomes a “down” particle.

- after each swap proposal, update the histograms $n_u(i), n_d(i)$, counting the number of “up” and “down” particles for the Markov chain associated with β_i .
- define $f_{up}(i) = \frac{n_u(i)}{n_u(i)+n_d(i)}$, the fraction of “up”-moving particles at β_i . By construction, notice that $f_{up}(\beta_1) = 1$ and $f_{up}(\beta_M) = 0$. f_{up} thus defines a probability distribution of “up” particles in the range $[\beta_1, \beta_M]$.
- The new inverse temperature parameters β' are chosen as the ordered set which assigns equal probability mass to each chain. This yields an f_{up} curve which is linear in the chain index.

The above procedure is applied iteratively, each time increasing N_s so as to fine-tune the β_i 's. To monitor return time, we can simply maintain a counter τ_i for each particle \mathbf{x}_i , which is (1) incremented at every sampling iteration and (2) reset to 0 whenever \mathbf{x}_i has label “down” and is swapped into β_1 . A lower-bound for return time is then given by $\hat{\tau}_{rt} = \sum_{i=0}^M \tau_i$.

4.2.3 Optimizing \mathcal{T} while Learning

Online Beta Adaptation

While the above algorithm exhibits the right properties, it is not very well suited to the context of learning. When training an RBM, the distribution we are sampling from is continuously changing. As such, one would expect the optimal set \mathcal{T} to evolve over time. We also do not have the luxury of performing N_s sampling steps after each gradient update.

Our solution is simple: the histograms n_u and n_d are updated using an exponential moving average, whose time constant is in the order of the return time $\hat{\tau}_{rt}$. Using $\hat{\tau}_{rt}$ as the time constant is crucial as it allows us to maintain flow statistics at the proper timescale. If an “up” particle reaches the i -th chain, we update $n_u(i)$ as follows:

$$n_u^{t+1}(i) = n_u^t(i)(1 - 1/\hat{\tau}_{rt}^t) + 1/\hat{\tau}_{rt}^t, \quad (4.5)$$

where $\hat{\tau}_{rt}^t$ is the estimated return time at time t .

Using the above, we can estimate the set of optimal inverse temperatures β'_i . Beta values are updated by performing a step in the direction of the optimal value: $\beta_i^{t+1} = \beta_i^t + \mu(\beta'_i - \beta_i^t)$, where μ is a learning rate on β . The properties of [Katzgraber](#)

et al. (2006) naturally enforce the ordering constraint on the β_i 's.

Choosing M and β_M

Another important point is that Katzgraber et al. (2006) optimizes the set \mathcal{T} while keeping the bounds β_1 and β_M fixed. While $\beta_1 = 1$ is a natural choice, we expect the optimal β_M to vary during learning. For this reason, we err on the side of caution and use $\beta_M = 0$, relying on a **chain spawning process** to maintain sufficiently high swap rates between neighboring parallel chains. Spawning chains as required by the sampler should therefore result in increased stability, as well as computational savings.

Lingenheil et al. (2009) performed an interesting study where they compared the round trip rate $1/\tau_{rt}$ to the average swap rate measured across all chains. They found that the DEO algorithm, which alternates between proposing swaps between chains $\{(i, i + 1); \forall \text{ even } i\}$ followed by $\{(i, i + 1); \forall \text{ odd } i\}$, gave rise to a concave function with a broad maximum around an average swap rate of $\bar{r} = \sum_i r_i \approx 0.4$

Our temperature adaptation therefore works in two phases:

1. The algorithm of Katzgraber et. al is used to optimize $\{\beta_i; 1 < i < M\}$, for a fixed M .
2. Periodically, a chain is spawned whenever $\bar{r} < \bar{r}_{min}$, a hyper-parameter of the algorithm.

Empirically, we have observed increased stability when the index j of the new chain is selected such that $j = \operatorname{argmax}_i (|f_{up}(i) - f_{up}(i+1)|)$, $i \in [1, M-1]$. To avoid a long burn-in period, we initialize the new chain with the state of the $(j + 1)$ -th chain and choose its inverse temperature as the mean $(\beta_j + \beta_{j+1})/2$. A small but fixed burn-in period allows the system to adapt to the new configuration.

4.3 Results and Discussion

We evaluate our adaptive SML-PT algorithm (SML-APT) on a complex, synthetic dataset. This dataset is heavily inspired from the one used in Desjardins et al. (2010b) and was specifically crafted to push the limits of the algorithm.

It is an online dataset of 28x28 binary images, where each example is sampled from a mixture model with probability density function $f_X(x) = \sum_{m=1}^5 w_m f_{Y_m}(x)$. Our dataset thus consists of 5 mixture components whose weights w_m are sampled uniformly in the unit interval and normalized to one. Each mixture component Y_m is itself a random 28x28 binary image, whose pixels are independent random variables having a probability p_m of being flipped. From the point of view of a sampler performing a random walk in image space, p_m is inversely proportional to the difficulty of finding the mode in question. The complexity of our synthetic dataset comes from our particular choice of w_m and p_m .ⁱ Large w_m and small p_m lead to modes which are difficult to sample and in which a Gibbs sampler would tend to get trapped. Large p_m values on the other hand will tend to intercept "down" moving particles and thus present a challenge for parallel tempering.

Figure 4.1(a) compares the results of training a 10 hidden unit RBM, using standard SML, SML-PT with $\{10, 20, 50\}$ parallel chains and our new SML-APT algorithm. We performed 10^5 updates (followed by $2 \cdot 10^4$ steps of sampling) with mini-batches of size 5 and tested learning rates in $\{10^{-3}, 10^{-4}\}$, β learning rates in $\{10^{-3}, 10^{-4}, 10^{-5}\}$. For each algorithm, we show the results for the best performing hyper-parameters, averaging over 5 different runs. Results are plotted with respect to computation time to show the relative computational cost of each algorithm.

As we can see, standard SML fails to learn anything meaningful: the Gibbs sampler is unable to cope with the loss in ergodicity and the model diverges. SML-PT on the other hand performs much better. Using more parallel chains in SML-PT consistently yields a better likelihood score, as well as reduced variance. This seems to confirm that using more parallel chains in SML-PT increases the ergodicity of the sampler. Finally, SML-APT outperforms all other methods. As we will see in Figure 4.2, it does so using only 20 parallel chains. Unfortunately, the computational cost seems similar to 50 parallel chains. We hope this can be reduced to the same cost as SML-PT with 20 chains in the near future. Also interesting to note, while the variance of all methods increase with training time, SML-APT seems immune to this issue.

We now compare the various metrics being optimized by our adaptive algorithm. Figure 4.1(b) shows the average return time for each of the algorithms. We can see that SML-APT achieves a return time which is comparable to SML-PT with

i. $w = [0.3314, 0.2262, 0.0812, 0.0254, 0.3358]$ and $p = [0.0001, 0.0137, 0.0215, 0.0223, 0.0544]$

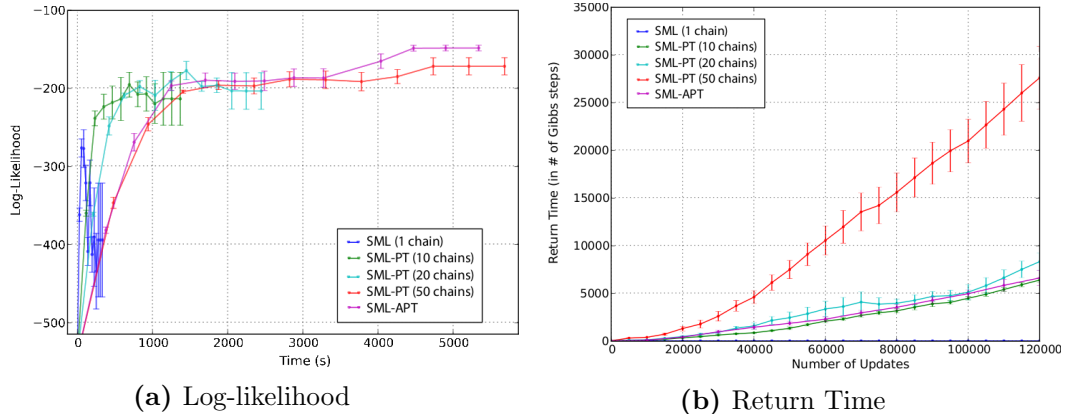


Figure 4.1: (a) Comparison of training likelihood as a function of time for standard SML, SML-PT with 10/20/50 chains and the proposed SML-APT (initialized with 10 chains). SML-APT adapts the temperature set $\mathcal{T} = \{T_1, \dots, T_M; T_1 < T_i < T_M\}$ to minimize round trip time between chains T_1 and T_M , and modifies the number of chains M to maintain a minimal average swap rate. The resulting sampler exhibits greater ergodicity and yields better likelihood scores than standard SML and SML-PT, without requiring a careful hand-tuning of \mathcal{T} . (b) Average return time of each algorithm. SML-APT successfully minimizes this metric resulting in a return time similar to SML-PT 10, while still outperforming SML-PT 50 in terms of likelihood. Errors bars represent standard error on the mean.

10 chains, while achieving a better likelihood score than SML-PT 50.

We now select the best performing seeds for SML-PT with 50 chains and SML-APT, and show in Figure 4.2, the resulting $f_{up}(i)$ curves obtained at the end of training.

The blue curve plots f_{up} as a function of beta index, while the red curves plots f_{up} as a function of β . We can see that SML-APT results in a more or less linear curve for $f_{up}(i)$, which is not the case for SML-PT. In Figure 4.3(a) we can see the effect on the pair-wise swap statistics r_i . As reported in Katzgraber et al. (2006), optimizing \mathcal{T} to maintain a linear f_{up} leads to temperatures pooling around the bottleneck. In comparison, SML-PT fails to capture this phenomenon regardless of whether it uses 20 or 50 parallel chains (figures 4.3(b)-4.3(c)).

Finally, Figure 4.4 shows the evolution of the inverse temperature parameters throughout learning. We can see that the position of the bottleneck in temperature space changes with learning. As such, a manual tuning of temperatures would be hopeless in achieving optimal return times.

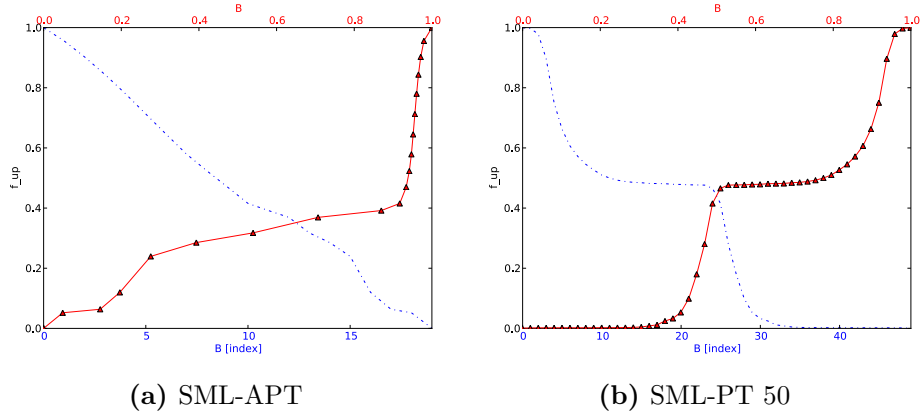


Figure 4.2: Return time is minimized by tagging each particle with a label: “up” if the particle visited T_1 more recently than T_M and “down” otherwise. Histograms $n_u(i)$ and $n_d(i)$ track the number of up/down particles at each temperature T_i . Temperatures are modified such that the ratio $f_{up}(i) = n_u(i)/(n_u(i)+n_d(i))$ is linear in the index i . (a) f_{up} curve obtained with SML-APT, as a function of temperature index (blue) and inverse temperature (red). SML-APT achieves a linear f_{up} in the temperature index i . (b) Typical f_{up} curve obtained with SML-PT (here using 50 chains). f_{up} is not linear in the index i , which translates to larger return times as shown in Fig. ??.

4.4 Conclusion

We have introduced a new adaptive training algorithm for RBMs, which we call Stochastic Maximum Likelihood with Adaptive Parallel Tempering (SML-APT). It leverages the benefits of PT in the negative phase of SML, but adapts and spawns new temperatures so as to minimize return time. The resulting negative phase sampler thus exhibits greater ergodicity. Using a synthetic dataset, we have shown that this can directly translate to a better and more stable likelihood score. In the process, SML-APT also greatly reduces the number of hyper-parameters to tune: temperature set selection is not only automated, but optimal. The end-user is left with very few dials: a standard learning rate on β_i and a minimum average swap rate \bar{r}_{min} below which to spawn.

Much work still remains. In terms of computational cost, we would like a model trained with SML-APT and resulting in M chains, to always be upper-bounded by SML-PT initialized with M chains. Obviously, the above experiments should also be repeated with larger RBMs on natural datasets, such as MNIST or Caltech

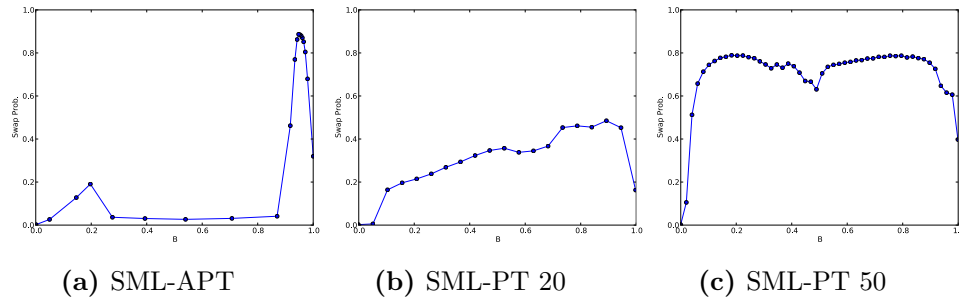


Figure 4.3: Pairwise swap statistics obtained after 10^5 updates. Minimizing return time causes SML-APT to pool temperatures around bottlenecks, achieving large swap rates (0.9) around bottlenecks with relatively few chains. SML-PT on the other hand results in a much flatter distribution, requiring around 50 chains to reach swap rates close to 0.8.

Silhouettes.ⁱ

Acknowledgments

The authors acknowledge the support of the following agencies for research funding and computing support: NSERC, Compute Canada and CIFAR. We would also like to thank the developers of Theanoⁱⁱ, for developing such a powerful tool for scientific computing, especially gradient-based learning.

i. <http://people.cs.ubc.ca/~bmarlin/data/index.shtml>

ii. <http://deeplearning.net/software/theano/>

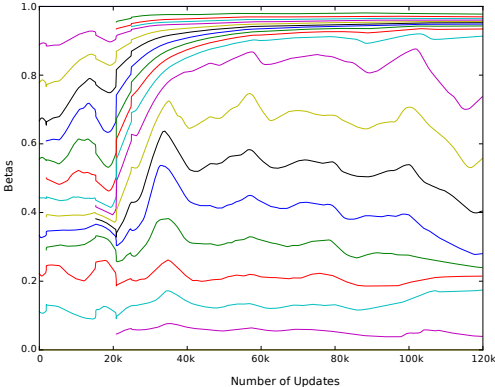


Figure 4.4: Adaptive temperatures during SML-APT training
Graphical depiction of the set $\{\beta_i; i \in [1, M]\}$, of inverse temperature parameters used by SML-APT during learning. Temperatures pool around a bottleneck to minimize return time, while new chains are spawned to maintain a given average swap rate. Note that the last 20k updates actually correspond to a pure sampling phase (i.e. a learning rate of 0).

5

Prologue to Second Article

5.1 Article Details

On Tracking the Partition Function. Guillaume Desjardins, Aaron Courville and Yoshua Bengio. Proceedings of the *25th Annual Conference on Neural Information Processing Systems (NIPS 2011)*.

Personal Contribution. The idea that one could leverage gradient descent to track the partition function during learning, came from my personal experimentation with SML-APT and AIS. I implemented the algorithm and performed all of the experiments reported in the paper. The inference equations were developed jointly with Aaron Courville. I also contributed heavily to the writing of the paper, with Aaron Courville writing most of the Introduction and Section 6.5.

5.2 Context

While the partition function is not required for estimating the SML gradient nor for feature extraction, it remains an important quantity which is crucial in performing model comparison, early-stopping, Bayesian learning of MRFs and simply evaluating probabilities. To this day, the method of choice for estimating the partition function of MRFs remains AIS (Salakhutdinov and Murray, 2008), presented in Section 6.2.1. While AIS can be made highly accurate by scaling the number of particles (mini-batch size) and the number of intermediate temperatures, it comes at a large computational cost.

5.3 Contributions

The contribution of this paper is a novel algorithm which exploits our previously developed Adaptive Parallel Tempering (APT) algorithm and the smoothness of gradient descent training to obtain an online estimate of the partition function. The computational cost of our method is similar to that of training alone. Our method relies on estimating both the change in log-partition function incurred by each parameter update, and the delta in log-partition function between neighboring chains simulated by SML-APT. These observations are integrated into a Kalman-like filter to obtain an online estimate.

5.4 Recent Developments

To the authors' knowledge, since the publication of this paper, no new algorithms have been proposed which tackle the direct problem of estimating $\log Z$ in an online fashion. There are however several recent developments in partition function estimation which are worth mentioning.

In Coupled Adaptive Simulated Tempering (CAST), [Salakhutdinov \(2010a\)](#) used the Wang-Landau (WL) algorithm to ensure that Simulated Tempering evenly distributed its simulation time across all rungs of the temperature ladder. The concept can be more generally applied to untempered MRFs however, by clustering the state-space directly through a binning of the energy function. The WL transition operator then exploits the density of states function ⁱ to ensure that the Markov chain spends equal time in each energy bin. [Stefano Ermon and Selman \(2011\)](#) proposed two improvements to this general formulation. Observing that the partition function is dominated by low-energy configurations, they propose saturating the energy function before performing the binning process. This prevents the Markov chain from spending time in configurations having little to no impact on the partition function estimate. Second, the WL transition operator is modified to incorporate a focused random walk which gives higher probability to

i. The density of states function $n(E)$ counts the number of input configuration yielding the same energy level. One commonly used estimator for $n(E)$ is a basic histogram, which bins the energy function into a fixed set of levels.

lower energy configurations. The resulting algorithm MCMCFocusedFlatSAT was shown to yield faster and better estimates of the partition function for Markov Logic Networks and Ising Models, compared to standard Gibbs sampling or variational approaches. Recently, [Stefano Ermon and Selman \(2012\)](#) introduced a novel message passing algorithm called *Density Propagation* to estimate the density of states function, whose efficiency stems from exploiting the structure of the graphical model. To date, neither of these methods have been applied to general BMs with latent variables.

[Jianzhu Ma and Xu \(2013\)](#) recently introduced Langevin Importance Sampling (LIS) for estimating the partition function of MRFs. Similar to Simulated Tempering and CAST, LIS samples from the joint distribution $p(x, \beta) \propto \exp(-E(x, \beta))$. Instead of using WL or a random walk to explore temperature space however, LIS updates the temperature parameter following the Langevin equation, which combines gradient information of $\log p$ with Gaussian noise. This allows the method to efficiently explore the energy landscape while allowing samples to escape from local minima of the energy function by raising the temperature. This is contrary to AIS which deterministically anneals the temperature during simulation. Despite similarities to CAST, LIS has so far only been applied to the problem of partition function estimation and does not yet constitute a viable training algorithm for MRFs.

The focus on adaptive tempering methods (whether for learning or partition function estimation) highlights the importance of selecting a proper temperature set \mathcal{T} . Recently, [Roger Grosse and Salakhutdinov \(2013\)](#) made a significant discovery in the context of AIS. When considering a fixed set of temperatures, one typically chooses a geometric spacing for the interpolating distributions such that $p_k = p_A^{(1-\beta_k)} p_B^{\beta_k}$. For models in the exponential family, this is analogous to interpolating between model parameters θ_A and θ_B . In this work, the authors propose instead to interpolate between the expected sufficient statistics of the model, i.e. the moments. This results in a more efficient estimate of the partition function, both in the number of chains required and number of burn-in steps required at each temperature.

6

On Tracking the Partition Function

MARKOV RANDOM FIELDS (MRFs) have proven very powerful both as density estimators and feature extractors for classification. However, their use is often limited by an inability to estimate the partition function Z . In this paper, we exploit the gradient descent training procedure of Restricted Boltzmann Machines (a type of MRF) to **track** the partition function during learning. Our method relies on two distinct sources of information: (1) estimating the change ΔZ incurred by each gradient update, (2) estimating the difference in Z over a small set of tempered distributions using bridge sampling. The two sources of information are then combined using an inference procedure similar to Kalman filtering. Learning MRFs through Tempered Stochastic Maximum Likelihood, we can estimate Z using no more temperatures than are required for learning. Comparing to both exact values and estimates using Annealed Importance Sampling (AIS), we show on several datasets that our method is able to accurately track the partition function. In contrast to AIS, our method provides this estimate at each time-step, at a computational cost similar to that required for training alone.

6.1 Introduction

In many domains of application, problems are naturally expressed as a Gibbs measure, where the distribution over the domain is given by:

$$P(x) = \frac{1}{Z(\beta)} \exp\{-\beta E(x)\}, \text{ with } Z(\beta) = \sum_x \exp\{-\beta E(x)\}. \quad (6.1)$$

$E(x)$ is referred to as the “energy” of configuration x , and β is a free parameter known as the inverse temperature. $Z(\beta)$ is the normalization factor commonly referred to as the partition function.

Models of this kind include the Markov Random Field (MRF), which has been very popular within the vision and natural language processing communities. MRFs with latent variables – in particular Restricted Boltzmann Machines (RBMs) (Hinton et al., 2006) – are among the most well-studied building block for deep architectures (Bengio, 2009), e.g. being used in the unsupervised initialization of both Deep Belief Networks (Hinton et al., 2006) and Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009b).

As illustrated in Eq. 6.1, the partition function is computed by summing over all variable configurations. Since the number of configurations summed over in Eq. 6.1 scales exponentially with the number of variables, exact calculation of the partition function is generally computationally intractable. Without the partition function, probabilities under the model can only be determined up to a multiplicative constant, which seriously limits the model’s utility. One method recently proposed for estimating $Z(\beta)$ is Annealed Importance Sampling (AIS) (Neal, 2001; Salakhutdinov and Murray, 2008). In AIS, $Z(\beta)$ is approximated by the sum of a set of importance-weighted samples drawn from the model distribution. With a large number of variables, drawing a set of importance-weighted samples is generally subject to extreme variance in the importance weights. AIS alleviates this issue by annealing the model distribution through a series of slowly changing distributions that link the target model distribution to one where the partition function is tractable. While AIS is quite successful, it generally requires the use of tens of thousands of annealing distributions in order to achieve accurate results. This computationally intensive requirement renders AIS inappropriate as a means of maintaining a running estimate of the partition function throughout training.

Having ready access to the partition function throughout learning opens the door to a range of possibilities. Likelihood could be used as a basis for model comparison throughout training; early-stopping could be accomplished by monitoring an estimate of the likelihood of a validation set. Another important application is in Bayesian inference in MRFs (Murray and Ghahramani, 2004) where we require the partition function for each value of the parameters in the region of support. Tracking the partition function would also enable simultaneous estimation of all the parameters of a heterogeneous model, for example an extended directed graphical model with Gibbs distributions forming some of the model components. Another example application is the case where an MRF is combined with

other graphical models (e.g., a mixture model) and where learning proceeds *online*, i.e., updating the parameters after each new example is shown. In this work, we consider a method of *tracking* the partition function throughout learning. We exploit the basic observation that typically the parameters tend to change slowly during training; consequently, $Z(\beta)$ also tends to evolve slowly. This is true, for example, with models trained by simple stochastic gradient descent. Stochastic gradient descent is one of the most popular methods for training MRFs precisely because second order optimization methods typically require a deterministic gradient, whereas sampling-based estimators are the only practical option for models with an intractable partition function.

Our proposed method is based on a variation of the well-known Kalman filter: it combines a *bridge* estimator of the partition function (Bennett, 1976) and a version of AIS that is applied to estimate the change in $Z(\beta)$ from one training iteration to the next. With a limited computational budget, each method on its own tends not to provide a reliable estimate. However, by combining these estimates with a Kalman filter, we are able to reliably and accurately track the evolution of the partition function throughout learning. Our method is built on the parallel tempering framework similar to the method established in Desjardins et al. (2010b) for use in training Restricted Boltzmann Machines (RBMs). In doing so, our partition function estimator can make use of the samples generated in the course of training to provide an online estimate of the partition function while incurring relatively little additional computation. In this work we concentrate on tracking the partition functions of RBMs. However, the method we propose is readily applicable to tracking the partition function of other MRFs.

6.2 Previous Work

Estimating the partition function of an undirected graphical model has long been an important goal. Unsurprisingly, there are a wide range of methods which attempt to address this issue. Here we will focus on the two methods that are among the most established and successful partition function estimation strategies, namely Annealed Importance Sampling (AIS) (Neal, 2001) and Bridge Sampling (Bennett, 1976).

6.2.1 Annealed Importance Sampling

Given an unnormalized probability density function $\tilde{p}(x)$ whose normalization constant Z_p is unknown and $q(x) = \tilde{q}(x)/Z_q$, simple importance sampling allows one to estimate $\log Z_p$ as $\log Z_q + \log \frac{1}{K} \sum_{x \in \mathcal{X}} \frac{\tilde{p}(x)}{\tilde{q}(x)}$, where $\mathcal{X} = \{x^{(k)}; x^{(k)} \sim q(x), 1 \leq k \leq K\}$. However, if the KL divergence of $p = \tilde{p}/Z_p$ and q is large, the variance of the importance weights (ratio of the two unnormalized probabilities) will be large and the estimate of Z_p will be poor.

AIS addresses this issue by introducing a set of intermediate distributions $\{p_1(x), \dots, p_{\tau-1}(x)\}$, such that $KL(p_i||q) < KL(p_j||q)$ and $KL(p_i||p) > KL(p_j||p)$ for $i < j$. Relabeling distribution q as p_0 and p as p_τ , it can be shown that $Z_\tau/Z_0 \approx \frac{1}{K} \sum_{k=1}^K w^{(k)}$, with:

$$w^{(k)} = \frac{p_1(x_0^{(k)}) p_2(x_1^{(k)}) \dots p_{\tau-1}(x_{\tau-2}^{(k)}) p_\tau(x_{\tau-1}^{(k)})}{p_0(x_0^{(k)}) p_1(x_1^{(k)}) \dots p_{\tau-2}(x_{\tau-2}^{(k)}) p_{\tau-1}(x_{\tau-1}^{(k)})} \quad \sigma_{AIS}^2 \approx \frac{\text{Var}[w^{(k)}]}{[\sum_k w^{(k)}]^2}. \quad (6.2)$$

where $x_0^{(k)}$ is the k -th sample from p_0 and the $x_i^{(k)}$'s are either drawn independently from p_i , or sampled from a Markov chain with transition operator $T_i(x_i; x_{i-1})$ (converging to fixed point p_i). This has been shown to be unbiased, with a variance σ_{AIS}^2 defined above.

This variance can be mitigated in two ways: either by increasing the number of intermediate distributions τ or by using a larger number K of importance weights (Neal, 2001).

6.2.2 Bridge Sampling

Bridge sampling (Bennett, 1976) addresses the shortcomings of importance sampling in a slightly different manner. It relies on a single distribution p_* , interpolating between p_0 (for which partition function Z_0 is known) and p_1 (for which we want to estimate Z_1), and estimates Z_1/Z_0 as the ratio of the expected importance weights between each distribution and the bridge, with variance σ_{bridge}^2 .

$$\frac{Z_1}{Z_0} \approx \frac{\sum_{k=1}^K \frac{p_*(x_0^{(k)})}{p_0(x_0^{(k)})}}{\sum_{k=1}^K \frac{p_*(x_1^{(k)})}{p_1(x_1^{(k)})}} = \frac{\sum_{k=1}^K u_k}{\sum_{k=1}^K v_k} \quad \sigma_{bridge}^2 \approx \frac{\text{Var}[u_k]}{[\sum_k u_k]^2} + \frac{\text{Var}[v_k]}{[\sum_k v_k]^2} \quad (6.3)$$

This formulation is much more forgiving, and allows for $KL(p_0||p_1)$ to be much larger than required by normal importance sampling or AIS when computing importance weights across neighboring chains. This is because p_* is chosen in some optimal fashion to have large support both with p_0 and p_1 . One such optimal distribution is $p_*^{(opt)}(x) \propto \frac{p_0(x)p_1(x)}{rp_0(x)+p_1(x)}$ where $r = Z_1/Z_0$. While this appears to be a chicken and egg issue, it is possible to start with a coarse estimate of r and use the resulting bridge distribution to refine our estimate recursively (Neal, 2005), a trick which our tracking algorithm exploits with great success.

6.3 Training Restricted Boltzmann Machines

Our proposed partition function tracking strategy is potentially applicable to any Gibbs distribution model that is undergoing relatively smooth changes in the partition function. Here, we concentrate on its application to the Restricted Boltzmann Machine, since it has become the model of choice for learning unsupervised features for use in deep feed-forward architectures (Hinton et al., 2006; Bengio, 2009) as well as for modeling complex, high-dimensional distributions (Welling et al., 2005; Taylor and Hinton, 2009; Larochelle et al., 2010).

While several algorithms have been proposed for training RBMs, our method builds upon Stochastic Maximum Likelihood (SML) (Tieleman, 2008) which approximates the maximum likelihood gradient by drawing negative phase samples from a persistent Markov chain, which attempts to track changes in the model. If we denote the state of this chain at time step t as \mathbf{v}_t^- and the i -th training example as $\mathbf{v}^{(i)}$, then the stochastic gradient update follows $\phi(\mathbf{v}^{(i)}, h^{(1)}) - \phi(\tilde{\mathbf{v}}_{t+k}^-, h_{t+k}^{(1)-})$, where $h^{(1)} = \mathbb{E}[h^{(1)}|\mathbf{v} = \mathbf{v}^{(i)}]$ and ϕ is the potential function. $\tilde{\mathbf{v}}_{t+k}^-$ is obtained after k steps of alternating Gibbs starting from state \mathbf{v}_t^- and $h_{t+k}^{(1)-} = \mathbb{E}[h^{(1)}|\mathbf{v} = \mathbf{v}_{t+k}^-]$. We refer the reader to Bengio (2009) for a more complete overview.

6.4 Parallel Tempering

In Desjardins et al. (2010b), a parallel tempering strategy was employed to improve mixing of the SML negative-phase Markov chain in an algorithm referred to as SML-PT (Parallel Tempering). Better mixing results in a lower variance estimate of the negative gradient. Other effective strategies exist for improving the mixing of the SML negative phase Markov chain (Tieleman and Hinton, 2009; Salakhutdinov, 2010b,a). From the perspective of tracking the partition function however, we will see in Section 6.2.2 the parallel tempering scheme of Desjardins et al. (2010b) presents a rather unique advantage.

At a given time step t of the learning process, we sample from an extended system composed of multiple RBM models: $\mathcal{M}_t = \{\mathcal{M}_{1,t}, \mathcal{M}_{2,t}, \dots, \mathcal{M}_{M,t}\}$ using a set of M parallel Markov chains, with the i th chain drawing samples from the associated RBM, $\mathcal{M}_{i,t}$. Each $\mathcal{M}_{i,t}$ is defined by its joint probability distribution: $p_{i,t}(\mathbf{v}, h^{(1)}) = \exp(-\beta_i \mathbf{E}(\mathbf{x})) / Z_{i,t}$ represents a smoothed version of the distribution we wish to sample from, with the inverse temperature $\beta_i = 1/T_i \in [0, 1]$ (with $T_M = 0$) controlling the degree of smoothing. Distributions with small β values are easier to sample from as they exhibit greater ergodicity. Each RBM in the extended system has its own unique partition function $Z_{i,t}$.

After performing k Gibbs steps for each of the M Model distributions, cross-temperature state swaps are proposed between neighboring chains using a Metropolis-Hastings-based swap acceptance criterion. If we denote by $\mathbf{x}_{i,t}$ the joint state (visible and hidden) of the i -th chain at learning iteration t , the swap acceptance ratio r_i for swapping chains $(i, i + 1)$ is given by:

$$r_i = \max \left(1, \frac{p_{i,t}(\mathbf{x}_{i+1,t}) p_{i+1,t}(\mathbf{x}_{i,t})}{p_{i,t}(\mathbf{x}_{i,t}) p_{i+1,t}(\mathbf{x}_{i+1,t})} \right) \quad (6.4)$$

While many swapping schedules are possible, we use the Deterministic Even Odd algorithm (Lingenheil et al., 2009).

6.5 Tracking the Partition Function

Consider again the Parallel Tempering extended system construction. For each learning time step t we have the set of RBMs $\mathcal{M}_t = \{\mathcal{M}_{1,t}, \mathcal{M}_{2,t}, \dots, \mathcal{M}_{M,t}\}$. Associated with each of the RBM is a partition function $Z_{i,t}$. Unrolling the extended system in time (learning iterations), we can envision a 2-dimensional lattice of RBMs indexed by (i, t) (although only one temporal slice needs to be stored).

6.5.1 AIS between learning iterations

First, consider the sequence of models for just one temperature i , up to learning iteration t , with associated partition function: $Z_{i,0}, Z_{i,1}, \dots, Z_{i,t}$. SML training of the RBM model parameters is a stochastic gradient descent algorithm (typically over a minibatch of $N \in [1, 100]$ examples) where the parameters change by small increments following the SML approximation of the gradient. This implies that both the model distribution and the partition function change relatively slowly over learning increments with a rate of change being a function of the SML learning rate; i.e. we expect $p_{i,t}(\mathbf{v}, h^{(1)})$ and $Z_{i,t}$ to be close to $p_{i,t-1}(\mathbf{v}, h^{(1)})$ and $Z_{i,t-1}$ respectively.

This sequence of slowly changing distributions is reminiscent of the sequence of annealing distributions used in AIS. Specifically, we can estimate the ratio of partition functions between consecutive learning iterations as $Z_{i,t}/Z_{i,t-1} \approx \frac{1}{N} \sum_{n=1}^N w^{(n)}$ where $w^{(n)}$ is defined in Eq. 6.2. In the log domain, the estimate of the difference in log partition functions is given by:

$$O_{i,t}^{\Delta t} = \log \left\{ \frac{1}{N} \sum_{n=1}^N w^{(n)} \right\} \approx \log Z_{i,t} - \log Z_{i,t-1} \quad (6.5)$$

If we also choose the model at learning iteration $t = 0$ to be a model whose partition function can be computed tractably, i.e. the model parameters connecting \mathbf{v} to $h^{(1)}$ are zero, then we can, in principle, use the sequence of models that emerge from the learning process to provide an “online” estimate of the partition function.

In practice, the number of intermediate distributions connecting $p_{i,t}$ to $p_{i,t-1}$ is going to be a function of the learning rate, with larger learning rates requiring more chains. However typically $p_{i,t}$ is quite close to $p_{i,t-1}$ and we require only very few intermediate distributions. For reasons of computational efficiency, we use no

intermediate distributions in our experiments, as it allows us to reuse the particles sampled during learning.

6.5.2 Bridging the parallel tempering temperature gaps

Consider now the other dimension of our parallel tempered lattice of RBMs: temperature. As described in Sec. 6.4, in parallel tempering, the distributions at neighboring temperatures are designed to have significant overlap in their densities in order to permit particle swaps. However, $p_{i,t}(\mathbf{v}, h^{(1)})$ and $p_{i+1,t}(\mathbf{v}, h^{(1)})$ are not so close that we can use them as the intermediate distributions of AIS. AIS typically requires thousands of intermediate chains, and maintaining that number of parallel chains would carry a prohibitive computational burden. On the other hand, the Parallel Tempering strategy of spacing the temperature to ensure moderately frequent swapping nicely matches the ideal operating regime of Bridge Sampling.

In the log domain, we can estimate the difference of the log partition functions between neighboring chains ($i - 1$) and i by:

$$O_{i,t}^{\Delta\beta} = \log \sum_{n=1}^N \frac{p_*(x_{i-1,t}^{(n)})}{p_{i-1,t}(x_{i-1,t}^{(n)})} - \log \sum_{n=1}^N \frac{p_*(x_{i,t}^{(n)})}{p_{i,t}(x_{i,t}^{(n)})} \approx \log Z_{i,t} - \log Z_{i-1,t} \quad (6.6)$$

where $x_{i,t}^{(n)}$ is the n -th sample from $p_{i,t}$ and $p_*(x_{i,t}^{(n)})$ is the bridging distribution described in Sec. 6.2.2. Finally, similarly to the initial distribution at time-step $t = 0$, we specify that the highest temperature chain corresponds to a model $\mathcal{M}_{M,t}$, for all t , with $\beta_M = 0$, whose partition function is analytically tractable. Again, in the case of the RBM this is trivially satisfied by ensuring that the parameters tying the \mathbf{v} to the $h^{(1)}$ are zeroed out.

In principle, the Bridge Sampler alone would be capable of arriving at an accurate estimate of the partition function for each model in the extended system at every learning iteration by repeated application of Bridge Sampling. Unfortunately, reducing the variance sufficiently to provide useful estimates of the partition function would require using a relatively large number of samples at each temperature. Within the context of RBM training, the required number of samples at *each* of the parallel chains would have an excessive computational cost. Fortunately, as described in the next section, it is possible to combine multiple high variance

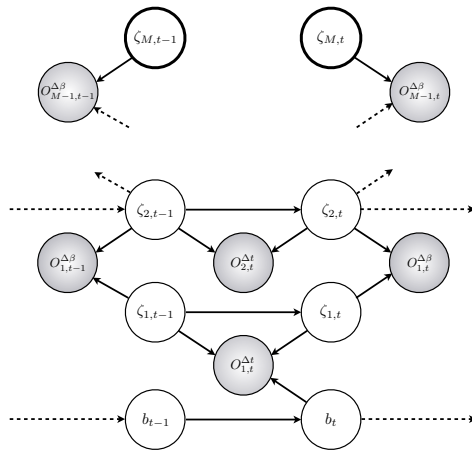
partition function estimates to achieve an accurate estimate.

6.5.3 Kalman Filtering the Log Partition Function

In the above we have described two paths to arrive at an estimate of the partition function for each the RBMs $\mathcal{M}_{t,i}$ in the lattice. In this section we describe a method to fuse all available information regarding each partition function to improve the overall accuracy of the estimate of every partition function while not sacrificing computational tractability.

Our strategy is to formally treat the unknown partition functions as latent variables and consider statistical estimates of the difference between neighboring log partition functions on the lattice, $O_t^{\Delta t}$ and $O_t^{\Delta \beta}$ as observed quantities, used to refine the joint distribution over the log partition functions. We follow the treatment of Neal (2001) in characterizing our uncertainty regarding $\log Z_{i,t}$ as a Gaussian distribution. We also characterize the evolution of the partition functions as independent Gaussian processes. Our tracking algorithm is a variation on the famous Kalman filter.

Let $\zeta_t = [\log Z_{1,t}, \dots, \log Z_{M,t}, b_t]$, where b_t is a random process introduced to account for the systematic bias of the “learning path” AIS when the same samples used to compute $O_t^{(\Delta t)}$ are also used in gradient estimation. Our model for the evolution of the system $\{\zeta_t, O_t^{(\Delta t)}, O_t^{(\Delta \beta)}\}$ is described in Fig. B.1. We use the notation $\mathcal{N}(\mu, \Sigma)$ to refer to a Gaussian distribution with mean μ and variance Σ . The model parameters μ_0 and Σ_0 specify the initial mean and covariance matrix of the latent state ζ_0 : $\mu_0 = [\log Z_{1,0}, \dots, \log Z_{M,0}, 0]$, where each $Z_{i,0}$ is tractable to compute exactly and the initial bias distribution has mean zero, and $\Sigma_0 = \text{Diag}[0, \dots, 0, \sigma_{b_0}^2]$ is a diagonal covariance matrix. The model dynamics are simple and represent our assumption that the partition function is slowly changing. The conditional expectation of ζ_t is ζ_{t-1} with fixed diagonal covariance $\Sigma_\zeta = \text{Diag}[\sigma_Z^2, \dots, \sigma_Z^2, \sigma_b^2]$. See Section 12.5 for the heuristics used to set σ_{b_0} , σ_b and σ_Z . The observation distributions in given in Fig. B.1 over $O_t^{(\Delta t)}$ and $O_t^{(\Delta \beta)}$ encode the relationship between the latent log partition function values and the statistical measurements given by AIS over learning iterations, $O_t^{(\Delta t)} = [O_{1,t}^{(\Delta t)}, \dots, O_{M-1,t}^{(\Delta t)}]^T$, and by the Bridge Sampler over the parallel chains, $O_t^{(\Delta \beta)} = [O_{1,t}^{(\Delta \beta)}, \dots, O_{M-1,t}^{(\Delta \beta)}]^T$, respectively. The matrix C encodes the relationship between the elements of ζ_t and ζ_{t-1} and $O_t^{(\Delta t)}$, while H



System Equations:

$$p(\zeta_0) = \mathcal{N}(\mu_0, \Sigma_0)$$

$$p(\zeta_t | \zeta_{t-1}) = \mathcal{N}(\zeta_{t-1}, \Sigma_\zeta)$$

$$p(O_t^{(\Delta t)} | \zeta_t, \zeta_{t-1}) = \mathcal{N}(C[\zeta_t, \zeta_{t-1}]^T, \Sigma_{\Delta t})$$

$$p(O_t^{(\Delta\beta)} | \zeta_t) = \mathcal{N}(H\zeta_t, \Sigma_{\Delta\beta})$$

$$C = \begin{bmatrix} & & & 1 \\ I_{M-1} & -I_{M-1} & & 0 \\ & & & \vdots \\ & & & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} -1 & +1 & 0 & 0 & 0 \\ 0 & -1 & +1 & 0 & \vdots & 0 \\ & & \dots & & & 0 \\ 0 & 0 & 0 & -1 & +1 & 0 \end{bmatrix}$$

Figure 6.1: Graphical model capturing the evolution of the log-partition function. A directed graphical model for partition function tracking. The shaded nodes represent observed variables, and the double-walled nodes represent the analytically tractable $\log Z_{M,t}$, with $\beta = 0$ (infinite temperature). For clarity of presentation, the graph shows the bias term as distinct from the other $\zeta_{i,t}$, recall $b_t = \zeta_{M+1,t}$

encodes the relationship between the elements of ζ_t and $O_t^{(\Delta\beta)}$. The elements of the diagonal matrices $\Sigma_{\Delta t}$ and $\Sigma_{\Delta\beta}$ are updated online from the estimated variances of AIS (Eq. 6.2) and of the Bridge Sampler (Eq. 6.3) respectively.

6.5.4 Inference over the latent state

We now consider the steps involved in the inference process in moving from an estimate of the posterior over the latent state at time $t - 1$ to an estimate of the posterior at time t . We begin by assuming we know the posterior $p(\zeta_{t-1} \mid O_{t-1:0}^{(\Delta t)}, O_{t-1:0}^{(\Delta\beta)})$, where $O_{t-1:0}^{(\cdot)} = [O_1^{(\cdot)}, \dots, O_{t-1}^{(\cdot)}]$. We define $p(\zeta_{t-1} \mid O_{t-1:0}^{(\Delta t)}, O_{t-1:0}^{(\Delta\beta)}) \sim \mathcal{N}(\mu_{t-1,t-1}, P_{t-1,t-1})$, that is $\mu_{t-1,t-1}$ and $P_{t-1,t-1}$ are the posterior mean and covariance at time $t - 1$. The two indices indicate which is the latest observation being conditioned upon for each of the two types of observations. For example, $\mu_{t,t-1}$ represents the posterior mean given $O_{t:0}^{(\Delta t)}$ and $O_{t-1:0}^{(\Delta\beta)}$.

Departing from the typical Kalman filter setting, $O_t^{(\Delta t)}$ depends on both ζ_t and ζ_{t-1} . So in order to incorporate this observation into our estimate of the latent state, we need to specify the prior joint distribution:

$$p(\zeta_{t-1}, \zeta_t \mid O_{t-1:0}^{(\Delta t)}, O_{t-1:0}^{(\Delta\beta)}) = \mathcal{N}(\zeta_{t-1}, \Sigma_\zeta) \mathcal{N}(\mu_{t-1,t-1}, P_{t-1,t-1}) = \mathcal{N}(\eta_{t-1,t-1}, V_{t-1,t-1})$$

where $\eta_{t-1,t-1} = \begin{bmatrix} \mu_{t-1,t-1} \\ \mu_{t-1,t-1} \end{bmatrix}$ and $V_{t-1,t-1} = \begin{bmatrix} P_{t-1,t-1} & P_{t-1,t-1} \\ P_{t-1,t-1} & \Sigma_\zeta + P_{t-1,t-1} \end{bmatrix}$

Incorporating the observation $O_t^{(\Delta t)}$ into the joint distribution over ζ_{t-1} and ζ_t yields, through Bayes rule, $p(\zeta_{t-1}, \zeta_t \mid O_{t:0}^{(\Delta t)}, O_{t-1:0}^{(\Delta\beta)}) = \mathcal{N}(\eta_{t,t-1}, V_{t,t-1})$ where $V_{t,t-1} = (V_{t,t-1}^{-1} + C^T \Sigma_{\Delta t}^{-1} C)^{-1}$ and $\eta_{t,t-1} = V_{t,t-1} (C^T \Sigma_{\Delta t} O_t^{(\Delta t)} + V_{t-1,t-1}^{-1} \eta_{t-1,t-1})$. Now that the AIS estimate is incorporated into the model, we no longer require the ζ_{t-1} component of the joint conditional distribution. Marginalizing over ζ_{t-1} , and we define

$$p(\zeta_t \mid O_{t:0}^{(\Delta t)}, O_{t-1:0}^{(\Delta\beta)}) = \int p(\zeta_{t-1}, \zeta_t \mid O_{t:0}^{(\Delta t)}, O_{t-1:0}^{(\Delta\beta)}) d\zeta_{t-1} = \mathcal{N}(\mu_{t,t-1}, P_{t,t-1}) \quad (6.7)$$

where $\mu_{t,t-1} = [\eta_{t,t-1}]_2$ is vector with the lower $M + 1$ elements and $P_{t,t-1} = [V_{t,t-1}]_{2,2}$ is the lower right-hand quadrant of $V_{t,t-1}$. It remains only to incorporate the Bridge Sampler estimate $O_t^{(\Delta\beta)}$ by a second application of Bayes rule: $p(\zeta_t \mid$

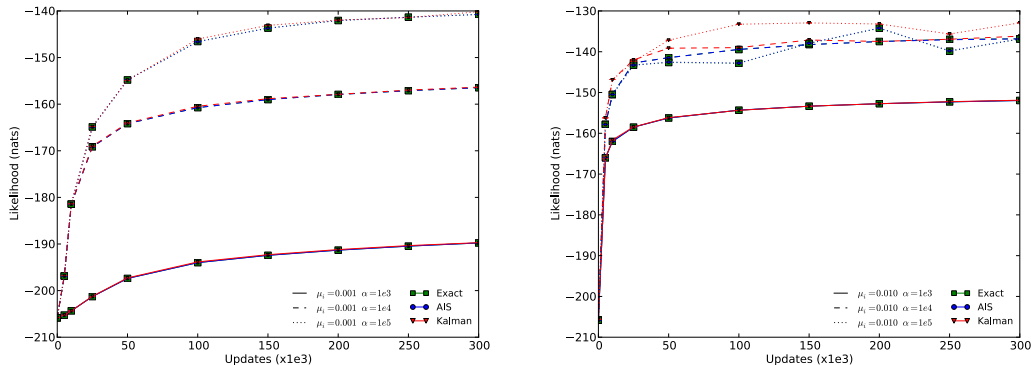


Figure 6.2: Comparison of exact test-set likelihood and estimated likelihood as given by AIS and our tracking algorithm. We trained a 25-hidden unit RBM for 300k updates using SML, with a learning rate schedule $\mu_t = \min(\mu_i \frac{\alpha}{t+1}, \mu_i)$, with (left) $\mu_i = 0.001$ and (right) $\mu_i = 0.01$ varying $\alpha \in \{10^3, 10^4, 10^5\}$.

$$O_{t:0}^{(\Delta t)}, O_{t:0}^{(\Delta \beta)}, O_t^{(\Delta \beta)} = \mathcal{N}(\mu_{t,t}, P_{t,t}), \text{ where } P_{t,t} = (P_{t,t-1}^{-1} + H^T \Sigma_{\Delta \beta}^{-1} H)^{-1} \text{ and } \mu_{t,t} = P_{t,t} (H^T \Sigma_{\Delta \beta} O_t^{(\Delta \beta)} + P_{t,t-1}^{-1} \mu_{t,t-1})$$

6.6 Experimental Results

6.6.1 Comparing to Exact Likelihood

In this section, we compare the performance of our tracking algorithm to the exact likelihood, obtained by marginalizing over the hidden units. We chose 25 hidden units and trained on the ubiquitous MNIST dataset (LeCun et al., 1998) for 300k updates, using both fixed and adaptive learning rates. The main results are shown in Figure 6.2.

In Figure 6.2 (left), we can see that our tracker provides a very good fit to the likelihood for a base learning rate of $\mu_i = 0.001$ and decrease constants α in $\{10^3, 10^4, 10^5\}$. Increasing the base learning rate to $\mu_i = 0.01$ in Figure 6.2 (right), we maintain a good fit up to $\alpha = 10^4$, with a small dip in performance at 50k updates. Our tracker fails however to capture the oscillatory behavior engendered by too high of a learning rate ($\mu_i = 0.01, \alpha = 10^5$). It is interesting to note that

the failure mode of our algorithm seems to coincide with an unstable optimization process.

6.6.2 Comparing to AIS for Large-scale Models

We now test the performance of our tracking algorithm on larger models, where exact computation of the likelihood is no longer possible. Our baseline for comparison is thus AISⁱ. Our models consisted of RBMs with 500 hidden units, trained using SML-APT (Desjardins et al., 2010b) on the MNIST and Caltech Silhouettes (Marlin et al., 2010) datasets. We performed 200k updates, with learning rate parameters $\mu_i \in \{.01, .001\}$ and $\alpha \in \{10^3, 10^4, 10^5\}$.

On MNIST, AIS estimated the test-likelihood of our best model at -94.34 ± 3.08 (where \pm indicates the 3σ confidence interval), while our tracking algorithm reported a value -89.96 . On Caltech Silhouettes, our model reached -134.23 ± 21.14 according to AIS, while our tracker reported -114.31 . To put these numbers in perspective, Salakhutdinov and Murray (2008) reports values of -125.53 , -105.50 and -86.34 for 500 hidden unit RBMs trained with CD-1, CD-3 and CD-25 respectively. Marlin et al. (2010) seem to report around -120 for Caltech Silhouettes, again using 500 hidden units.

Figure 6.3 (left) shows a detailed view of the Kalman filter measurements and its output, for the best performing MNIST model. We can see that the variance on $O_t^{(\Delta\beta)}$ (bridge sampling estimate) grows slowly over time, which is mitigated by a decreasing variance on $O_t^{(\Delta t)}$. As the model converges and the learning rate is decreasing, \mathcal{M}_{t-1} and \mathcal{M}_t become progressively closer and the “learning path” AIS estimate becomes more robust. An important point to note is that a naive linear-spacing of temperatures yielded low exchange rates between neighboring temperatures, with adverse effects on the quality of our Bridge Sampling estimates. As a result, we observed a drop in performance, both in likelihood as well as tracking performance. Adaptive Tempering (Desjardins et al., 2010b) proved crucial in getting good tracking. Temperatures were thus automatically tuned to maximize the flow of particles between extremal temperatures. This has the effect of increasing the overlap between adjacent distributions $p_{i,t}, p_{i+1,t}$, yielding better bridge sam-

i. Our base AIS configuration was 10^3 chains spaced linearly between $\beta = [0, 0.5]$, 10^4 chains for the interval $[0.5, 0.9]$ and 10^4 between $[0.9, 1.0]$. We report the estimated $\log Z$ by averaging over 100 annealed importance weights.

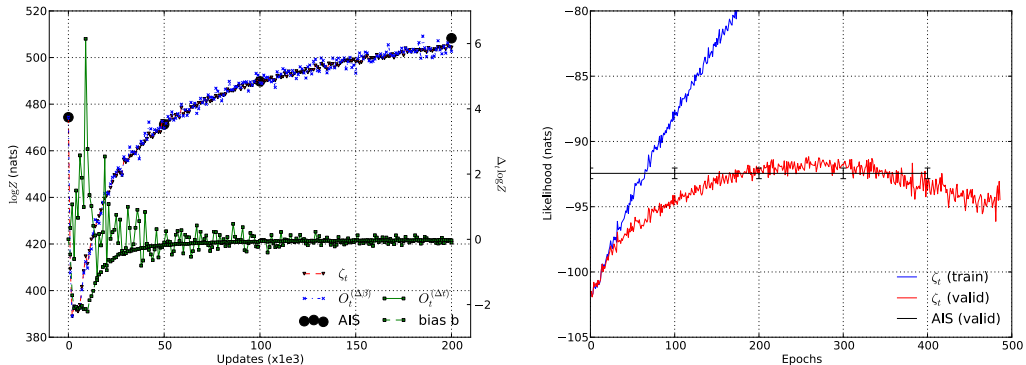


Figure 6.3: (left) Kalman filter measurements $O_t^{(\Delta t)}$, $O_t^{(\Delta \beta)}$ and estimate ζ_t during training. Note that $O_t^{(\Delta t)}$ represents the estimated likelihood change between consecutive gradient updates. “bias b” shows the estimated bias on $O_t^{(\Delta t)}$, induced by the gradient updates. Both green curves correspond to the right-hand axis. Point estimate of the partition function as given by AIS. The confidence interval at three standard-deviation of the AIS value at 200k updates was measured to be 3.08. (right) Example of early-stopping on dna dataset.

pling estimates.

6.6.3 Early-Stopping Experiments

Our final set of experiments highlights the performance of our algorithm on a wide-variety of datasets. Their use was motivated by their recent inclusion in [Larochelle and Murray \(2011\)](#). Model selection was done using grid-search, based on validation set performance. Because of tracking, our algorithm afforded us the ability to perform early-stopping, the advantages of which are highlighted in Fig. 6.3 (right).

We tested parameters in the following range: number of hidden units in $\{100, 200, 500, 1000\}$ (depending on dataset size), learning rates in $\{10^{-2}, 10^{-3}, 10^{-4}\}$ either held constant during training or annealed with constants $\alpha \in \{10^3, 10^4, 10^5\}$. For tempering, we used 10 fixed temperatures, spaced linearly between $\beta = [0, 1]$. SGD was performed using mini-batches of size $\{10, 100\}$ when estimating the gradient, and mini-batches of size $\{10, 20\}$ for our set of tempered-chains (we thus simulate $10 \times \{10, 20\}$ tempered chains in total). As can be seen in Table 6.1, our tracker performs very well compared to the AIS estimates and across all datasets. Efforts

Dataset	RBM		RBM-25	NADE
	Kalman	AIS		
adult	-15.24	-15.70 (± 0.50)	-16.29	-13.19
connect4	-15.77	-16.81 (± 0.67)	-22.66	-11.99
dna	-87.97	-88.51 (± 0.97)	-96.90	-84.81
mushrooms	-10.49	-14.68 (± 30.75)	-15.15	-9.81
nips	-270.10	-271.23 (± 0.58)	-277.37	-273.08
ocr_letters	-33.87	-31.45 (± 2.70)	-43.05	-27.22
rcv1	-46.89	-48.61 (± 0.69)	-48.88	-46.66
web	-28.95	-29.91 (± 0.74)	-29.38	-28.39

Table 6.1: Estimated likelihoods of misc. UCI datasets, obtained via tracking Test set likelihood on various datasets. Models were trained using SML-PT.

Early-stopping was performed by monitoring likelihood on a hold-out validation set, using our running estimate of the partition function. Best models (i.e. the choice of hyper-parameters) were then chosen according to the AIS likelihood estimate. Results for 25-hidden unit RBMs and NADE are taken from [Larochelle and Murray \(2011\)](#), and appear simply to provide context. \pm indicates a confidence interval of three standard deviations.

to lower the variance of the AIS estimate proved unsuccessful, even going as far as 10^5 intermediate distributions.

6.7 Conclusion and Discussion

In this paper, we have shown that while exact calculation of the partition function of RBMs may be intractable, one can exploit the smoothness of gradient descent learning, in order to approximately track the evolution of the log-partition function during learning. Our method exploits the Parallel Tempering framework. At each time-step t , Bridge Sampling allows us to estimate the ΔZ_i between adjacent chains providing a path to a known partition function Z_M . AIS can then be applied with very few interpolating chains between models at nearby learning iterations. For small enough learning rates, this can even be reduced to zero, as was the case in our experiment, and results in large computational gains. Treating the log Z_i 's as unknown variables, the formalism of the Gaussian graphical model

of Figure B.1 allowed us to combine multiple sources of information, smooth out the estimates and achieve good tracking of the partition function.

The method presented in the paper is also computationally attractive, with only a small computational overhead relative to SML-PT training. The added computational cost lies in the computation of the importance weights for AIS and Bridge Sampling. However, this boils down to computing free-energies which are mostly pre-computed in the course of gradient updates with the sole exception being the computation of an extra $p_{i,t}(x_{i,t-1})$ term in the “learning path” AIS.

In comparison to AIS, our method allows us to fairly accurately track the partition function, and at a per-point estimate cost well below that of AIS. Having a reliable and accurate online estimate of the partition function opens the doors to interesting research avenues.

7

Prologue to Third Article

7.1 Article Details

Metric-Free Natural Gradient for Joint-Training of Boltzmann Machines. Guillaume Desjardins, Razvan Pascanu, Aaron Courville and Yoshua Bengio. International Conference on Learning Representations (ICLR), 2013.

Personal Contribution. The particular form of the natural gradient for BMs was derived by myself, with the generic derivation of the natural gradient being joint work with Razvan Pascanu and Aaron Courville. I was the main author for the DBM code, and developed the MFNG algorithm with Razvan Pascanu, who also provided the crucial code for Conjugate Gradient and MinRes. The experiments and writing are my own, with the exception of the results found in Figure 8.2.

7.2 Context

The Hessian-Free (HF) optimization method of [Martens \(2010\)](#); [Martens and Sutskever \(2011\)](#) represents an important breakthrough in training deep and recurrent neural networks. Until its publication, pretraining was the method of choice for successfully training deep multi-layer perceptrons ([Hinton et al., 2006](#); [Bengio et al., 2007](#); [Lee et al., 2009](#)). Using Hessian-Free, the authors were able to outperform pre-trained auto-encoders of [Hinton and Salakhutdinov \(2006\)](#) using pure supervised learning. Later, this same method was adapted to learn long-term dependencies in Recurrent Neural Networks (RNN), seemingly bypassing issues of exploding and vanishing gradients ([Bengio et al., 1994](#)). HF belongs to the family of truncated Newton methods explored in Section 1.2.2. In addition to using CG to compute the Newton update direction $H^{-1}g$ (where H is the Hessian matrix and g the estimated gradient), HF bypasses the need for computing or storing the Hessian

matrix explicitly by using the R-operator, an efficient mechanism for computing Hessian-vector products.

At the time of publication however, it was not clear how the above could be adapted to the probabilistic setting of BMs.

Subsequently, [Montavon and Muller \(2012\)](#) showed how centering (see Section 2.5.2) could enable joint-training of DBMs. This paper provided convincing evidence that our previous reliance on greedy layer-wise pretraining ([Salakhutdinov and Hinton, 2009a](#)) stemmed from issues of optimization. This motivated us to revisit second-order optimization methods for BMs, to determine if they could not only subsume the centering trick but improve training altogether.

7.3 Contributions

To the authors' knowledge, this paper introduced the first practical algorithm for applying the natural gradient to large Boltzmann Machines. As with HF, our method uses a linear solver to invert the Fisher Information Matrix (FIM) and precludes the need to compute or store it explicitly through an efficient matrix-vector operation.

Our paper shows that the Metric-Free Natural Gradient (MFNG) algorithm (and its diagonal approximation) improves convergence speed when training DBMs via variational SML. Unfortunately, the method as presented in the paper is not yet computationally efficient. Surprisingly, we also found that the natural gradient was not a replacement for proper centering of the energy function.

7.4 Recent Developments

Concurrent to our work, [Pascanu and Bengio \(2013\)](#) showed that using Newton's method with the extended Gauss-Newton (EGN) matrix in lieu of the Hessian, was directly equivalent to the natural gradient algorithm. Since this approximation was found to work best in [Martens \(2010\)](#), HF and MFNG are essentially equivalent at a high-level: both are efficient implementations of the natural gradient, HF

being tailored to the optimization of deterministic functions and MFNG to the optimization of MRFs.

Since publication, it was brought to our attention that [Byrd et al. \(2011\)](#) may provide the key to making our method computationally efficient. In the context of a truncated Newton method, they found that it was preferable to use a much smaller batch size for estimating the Hessian and allocate more capacity towards a careful estimation of the gradient (via a larger batch size). [Pascanu and Bengio \(2013\)](#) also observed improved performance when using a separate set of samples for estimating the FIM, than to estimate the gradient. With regards to the need for centering, we are currently exploring two hypotheses. (1) The benefits of centering might stem from its global re-parametrization of the energy, whereas the natural gradient is only locally invariant to re-parametrizations of the model. (2) Alternatively, our treatment of latent variables in the derivation of the FIM might be to blame for our inability to perform joint-training without centering. In particular, we have observed that for a fixed setting of the parameters, centering the BM energy can greatly reduce the number of iterations required for inference. Given a maximal number of inference iterations, the failure of MFNG alone to perform joint-training might therefore stem from failures of the inference process. This suggests taking into account the manifold structure of the posterior in the context of inference.

Finally, our paper also failed to cite the Information Geometry Optimization (IGO) algorithm of [Arnold et al. \(2011\)](#), which applied the natural gradient to a toy RBM and predates our work. In addition to the (local) invariance properties of the natural gradient to re-parametrization of the model, their method also incorporates invariances to re-parametrizations of the input and monotonic transformations of the function undergoing optimization.

Metric-Free Natural Gradient for Joint-Training of Boltzmann Machines

THIS PAPER introduces the Metric-Free Natural Gradient (MFNG) algorithm for training Boltzmann Machines. Similar in spirit to the Hessian-Free method of [Martens \(2010\)](#), our algorithm belongs to the family of truncated Newton methods and exploits an efficient matrix-vector product to avoid explicitly storing the natural gradient metric L . This metric is shown to be the expected second derivative of the log-partition function (under the model distribution), or equivalently, the covariance of the vector of partial derivatives of the energy function. We evaluate our method on the task of joint-training a 3-layer Deep Boltzmann Machine and show that MFNG does indeed have faster per-epoch convergence compared to Stochastic Maximum Likelihood with centering, though wall-clock performance is currently not competitive.

8.1 Introduction

Boltzmann Machines (BM) have become a popular method in Deep Learning for performing feature extraction and probability modeling. The emergence of these models as practical learning algorithms stems from the development of efficient training algorithms, which estimate the negative log-likelihood gradient by either contrastive ([Carreira-Perpiñan and Hinton, 2005](#)) or stochastic ([Tieleman, 2008](#); [Younes, 1998](#)) approximations. However, the success of these models has for the most part been limited to the Restricted Boltzmann Machine (RBM) ([Freund and Haussler, 1992](#)), whose architecture allows for efficient exact inference. Unfortunately, this comes at the cost of the model’s representational capacity, which is limited to a single layer of latent variables. The Deep Boltzmann Machine (DBM) ([Salakhutdinov and Hinton, 2009a](#)) addresses this by defining a joint energy function over multiple disjoint layers of latent variables, where interactions within a

layer are prohibited. While this affords the model a rich inference scheme incorporating top-down feedback, it also makes training much more difficult, requiring until recently an initial greedy layer-wise pretraining scheme. Since, [Montavon and Muller \(2012\)](#) have shown that this difficulty stems from an ill-conditioning of the Hessian matrix, which can be addressed by a simple reparameterization of the DBM energy function, a trick called *centering* (an analogue to centering and skip-connections found in the deterministic neural network literature ([Schraudolph, 1998](#); [Raiko et al., 2012](#))). As the barrier to joint-trainingⁱ is overcoming a challenging optimization problem, it is apparent that second-order gradient methods might prove to be more effective than simple stochastic gradient methods. This should prove especially important as we consider models with increasingly complex posteriors or higher-order interactions between latent variables.

To this end, we explore the use of the Natural Gradient ([Amari, 1998](#)), which seems ideally suited to the stochastic nature of Boltzmann Machines. Our paper is structured as follows. Section 8.2 provides a detailed derivation of the natural gradient, including its specific form for BMs. While most of these equations have previously appeared in [Amari et al. \(1992\)](#), our derivation aims to be more accessible as it attempts to derive the natural gradient from basic principles, while minimizing references to Information Geometry. Section 8.3 represents the true contribution of the paper: a practical natural gradient algorithm for BMs which exploits the persistent Markov chains of Stochastic Maximum Likelihood (SML) ([Tieleman, 2008](#)), with a Hessian-Free (HF) like algorithm ([Martens, 2010](#)). The method, named Metric-Free Natural Gradient (MFNG) (in recognition of the similarities of our method to HF), avoids explicitly storing the natural gradient metric L and uses a linear solver to perform the required matrix-vector product $L^{-1}\mathbb{E}_q[\nabla \log p_\theta]$. Preliminary experimental results on DBMs are presented in Section 8.4, with the discussion appearing in Section 8.5.

i. Joint-training refers to the act of jointly optimizing θ (the concatenation of all model parameters, across all layers of the DBM) through maximum likelihood. This is in contrast to [Salakhutdinov and Hinton \(2009a\)](#), where joint-training is preceded by a greedy layer-wise pretraining strategy.

8.2 The Natural Gradient

8.2.1 Motivation and Derivation

The main insight behind the natural gradient is that the space of all probability distributions $\mathcal{P} = \{p_\theta(x); \theta \in \Theta, x \in \chi\}$ forms a Riemannian manifold. Learning, which typically proceeds by iteratively adapting the parameters θ to fit an empirical distribution q , thus traces out a path along this manifold. An immediate consequence is that following the direction of steepest descent in the original Euclidean parameter space does not correspond to the direction of steepest descent along \mathcal{P} . To do so, one needs to account for the metric describing the local geometry of the manifold, which is given by the Fisher Information matrix (Amari, 1985), shown in Equation 8.4. While this metric is typically derived from Information Geometry, a derivation more accessible to a machine learning audience can be obtained as follows.

The natural gradient aims to find the search direction $\Delta\theta$ which minimizes a given objective function, such that the Kullback–Leibler divergence $KL(p_\theta \parallel p_{\theta+\Delta\theta})$ remains constant throughout optimization. This constraint ensures that we make constant progress regardless of the curvature of the manifold \mathcal{P} and enforces an *invariance to the parameterization of the model*. The natural gradient for maximum likelihood can thus be formalized as:

$$\begin{aligned} \nabla_N := \Delta\theta^* &\leftarrow \operatorname{argmin}_{\Delta\theta} \mathbb{E}_q[-\log p_{\theta+\Delta\theta}(x)] \\ \text{s.t. } &KL(p_\theta \parallel p_{\theta+\Delta\theta}) = \text{const.} \end{aligned} \quad (8.1)$$

In order to derive a useful parameter update rule, we will consider the KL divergence under the assumption $\Delta\theta \rightarrow 0$. We also assume we have a discrete and bounded domain χ over which we define the probability mass functionⁱ p_θ . Taking the Taylor series expansion of $\log p_{\theta+\Delta\theta}$ around θ , and denoting ∇f as the column vector of partial derivatives with $\frac{\partial f}{\partial \theta_i}$ as the i -th entry, and $\nabla^2 f$ the Hessian matrix

i. When clear from context, we will drop the argument of p_θ to save space.

with $\frac{\partial^2 f}{\partial \theta_i \partial \theta_j}$ in position (i, j) , we have:

$$\begin{aligned} KL(p_\theta \parallel p_{\theta+\Delta\theta}) &\approx \sum_x p_\theta \log p_\theta - \sum_x p_\theta \left[\log p_\theta + (\nabla \log p_\theta)^T \Delta\theta + \frac{1}{2} \Delta\theta^T (\nabla^2 \log p_\theta) \Delta\theta \right] \\ &= \frac{1}{2} \Delta\theta^T \mathbb{E}_{p_\theta} [-\nabla^2 \log p_\theta] \Delta\theta \end{aligned} \quad (8.2)$$

with the transition stemming from the fact that $\sum_x p_\theta \frac{\partial \log p_\theta}{\partial \theta_i} = \frac{\partial}{\partial \theta_i} \sum_{x \in \mathcal{X}} p_\theta(x) = 0$. Replacing the objective function of Equation 8.1 by its first-order Taylor expansion and rewriting the constraint as a Lagrangian, we arrive at the following formulation for $\mathcal{L}(\theta, \Delta\theta)$, the loss function which the natural gradient seeks to minimize.

$$\mathcal{L}(\theta, \Delta\theta) = \mathbb{E}_q [-\log p_\theta] + \mathbb{E}_q [-\nabla \log p_\theta]^T \Delta\theta + \frac{\lambda}{2} \Delta\theta^T \mathbb{E}_{p_\theta} [-\nabla^2 \log p_\theta] \Delta\theta.$$

Setting $\frac{\partial \mathcal{L}}{\partial \Delta\theta}$ to zero yields the natural gradient direction ∇_N :

$$\nabla_N = L^{-1} \mathbb{E}_q [\nabla \log p_\theta] \quad \text{with } L = \mathbb{E}_{p_\theta} [-\nabla^2 \log p_\theta] \quad (8.3)$$

$$\text{or equivalently } L = \mathbb{E}_{p_\theta} [\nabla \log p_\theta \nabla^T \log p_\theta] \quad (8.4)$$

While its form is reminiscent of the Newton direction, the natural gradient multiplies the estimated gradient by the inverse of the expected Hessian of $\log p_\theta$ (Equation 8.3) or equivalently by the Fisher Information matrix (FIM, Equation 8.4). The equivalence between both expressions can be shown trivially, with the details appearing in the Appendix. We stress that both of these expectations are computed with respect to the *model distribution*, and thus computing the metric L does not involve the empirical distribution in any way. The FIM for Boltzmann Machines is thus *not equal* to the uncentered covariance of the maximum likelihood gradients. In the following, we pursue our derivation from the form given in Equation 8.4.

8.2.2 Natural Gradient for Boltzmann Machines

Derivation. Boltzmann machines define a joint distribution over a vector of binary random variables $x \in \{0, 1\}^N$ by way of an energy function $E(x) = -\sum_{k < l} W_{kl} x_k x_l - \sum_k b_k x_k$, with weight matrix $W \in \mathbb{R}^{N \times N}$ and bias vector $b \in \mathbb{R}^N$. Energy and probability are related by the Boltzmann distribution, such that $p(x) = \frac{1}{Z} \exp(-E(x))$,

with Z the partition function defined by $Z = \sum_x \exp(-E(x))$.

Starting from the expression of L found in Equation 8.3, we can derive the natural gradient metric for Boltzmann Machines.

$$L^{(BM)} = \mathbb{E}_{p_\theta} [\nabla^2 E(x) + \nabla^2 \log Z] = \mathbb{E}_{p_\theta} [\nabla^2 \log Z]$$

The natural gradient metric for first-order BMs takes on a surprisingly simple form: it is the expected Hessian of the log-partition function. With a few lines of algebra (whose details are presented in the Appendix), we can rewrite it as follows:

$$L^{(BM)} = \mathbb{E}_{p_\theta} \left[(\nabla E(x) - \mathbb{E}_{p_\theta} [\nabla E(x)])^T (\nabla E(x) - \mathbb{E}_{p_\theta} [\nabla E(x)]) \right] \quad (8.5)$$

$L^{(BM)}$ is thus given by the covariance of ∇E , measured under the model distribution p_θ . Concretely, if we denote W_{kl} and W_{mn} as the i and j -th parameters of the model respectively, the entry L_{ij} will take on the value $-\mathbb{E}[x_k x_l x_m x_n] + \mathbb{E}[x_k x_l] \mathbb{E}[x_m x_n]$.

Discussion. When computing the Taylor expansion of the KL divergence in Equation 8.2, we glossed over an important detail. Namely, how to handle latent variables in $p_\theta(x)$, a topic first discussed in Amari et al. (1992). If $x = [v, h]$, we could just as easily have derived the natural gradient by considering the constraint $KL(\sum_h p_\theta(v, h) \parallel \sum_h p_{\theta+\Delta\theta}(v, h)) = \text{const}$. Alternatively, since the distinction between visible and hidden units is entirely artificial (since the KL divergence does not involve the empirical distribution), we may simply wish to consider the distribution obtained by analytically integrating out a maximal number of random variables. In a DBM, this would entail marginalizing over all odd or even layers, a strategy employed with great success in the context of AIS (Salakhutdinov and Hinton, 2009a). In this work however, we only consider the metric obtained by considering the KL divergence between the full joint distributions p_θ and $p_{\theta+\Delta\theta}$.

8.3 Metric-Free Natural Gradient Implementation

We can compute the natural gradient ∇_N by first replacing the expectations of Equation 8.5 by a finite sample approximation. We can do this efficiently by reusing the model samples generated by the persistent Markov chains of SML. Given the size of the matrix being estimated however, we expect this method to require a larger number of chains than is typically used. The rest of the method is similar to the Hessian-Free (HF) algorithm of Martens (2010): we exploit an efficient matrix-vector implementation combined with a linear-solver, such as Conjugate Gradient or MinRes (Paige and Saunders, 1975), to solve the system $Ly = \mathbb{E}_q[\nabla \log p_\theta]$ for $y \in \mathbb{R}^N$. Additionally, we replace the expectation on the rhs. of this previous equation by an average computed over a mini-batch of training examples (sampled from the empirical distribution q), as is typically done in the stochastic learning setting.

For Boltzmann Machines, the matrix-vector product Ly can be computed in a straightforward manner, without recourse to Pearlmutter’s R-operator (Pearlmutter, 1994). Starting from a sampling approximation to Equation 8.5, we simply push the dot product inside of the expectation as follows:

$$\begin{aligned}
 L^{(BM)}y &\approx (S - \bar{S})^T [(S - \bar{S})y] & (8.6) \\
 \text{with } S &\in \mathbb{R}^{M \times N}, \text{ the matrix with entries } s_{mj} = \frac{\partial E(x_m)}{\partial \theta_j} \\
 \text{and } \bar{S} &\in \mathbb{R}^N, \text{ the vector with entries } s_j = \frac{1}{M} \sum_m s_{mj} \\
 \text{and } x_m &\sim p_\theta(x), m \in [1, M].
 \end{aligned}$$

By first computing the matrix-vector product $(S - \bar{S})y$, we can easily avoid computing the full $N \times N$ matrix L . Indeed, the result of this operation is a vector of length M , which is then left-multiplied by a matrix of dimension $N \times M$, yielding the matrix-vector product $Ly \in \mathbb{R}^N$. A single iteration of the MFNG is presented in Algorithm 8. A full open-source implementation is also available online.ⁱ

i. <https://github.com/gdesjardins/MFNG>

Algorithm 8 MFNG_iteration($\theta, \mathcal{X}^+, \mathcal{Z}_{old}^-$) θ : parameters of the model. $N := |\theta|$. \mathcal{X}^+ : mini-batch of training examples, with $\mathcal{X}^+ = \{x_m; m \in [1, M]\}$. \mathcal{Z}_{old}^- : previous state of Markov chains, with $\mathcal{Z} = \{z_m := (v_m, h_m^{(1)}, h_m^{(2)}); m \in [0, M]\}$.

-
- Generate positive phase samples
 - Initializing M Markov chains from state \mathcal{Z}_{old}^- , generate negative phase samples \mathcal{Z}_{new}^- .
 - Compute the vectors $s_m^+ = \frac{\partial E(z_m^+)}{\partial \theta}$ and $s_m^- = \frac{\partial E(z_m^-)}{\partial \theta}$, $\forall m$.
 - Compute negative log-likelihood gradient as $g = \frac{1}{M} \sum_m (s_m^+ - s_m^-)$.
 - Denote $S \in \mathbb{R}^{M \times N}$ as the matrix with rows s_m^- and $\bar{S} = \frac{1}{M} \sum_m s_m^-$.
- # Solve the system “ $Ly = g$ ” for y , given $L = (S - \bar{S})^T(S - \bar{S})$ and an initial zero vector.*
- # computeLy is a function which performs equation 8.6, without instantiating L .*
- $\nabla_N \theta \leftarrow \text{CGSolve}(\text{computeLy}, S, g, \text{zeros}(N))$
-

8.4 Experiments

We performed a proof-of-concept experiment to determine whether our Metric-Free Natural Gradient (MFNG) algorithm is suitable for joint-training of complex Boltzmann Machines. To this end, we compared our method to Stochastic Maximum Likelihood and a diagonal approximation of MFNG on a 3-layer Deep Boltzmann Machine trained on MNIST (LeCun et al., 1998). All algorithms were run in conjunction with the centering strategy of Montavon and Muller (2012), which proved crucial to successfully joint-train all layers of the DBM (even when using MFNG) ⁱ. We chose a small 3-layer DBM with 784-400-100 units at the first, second and third layers respectively, to be comparable to Montavon and Müller (2012). Hyper-parameters were varied as follows. For inference, we ran 5 iterations of either mean-field as implemented in Salakhutdinov and Hinton (2009a) or Gibbs sampling. The learning rate was kept fixed during training and chosen from the set $\{5 \cdot 10^{-3}, 10^{-3}, 10^{-4}\}$. For MinRes, we set the damping coefficient to 0.1 and used a fixed tolerance of 10^{-5} (used to determine convergence). Finally, we tested

ⁱ. The centering coefficients were initialized as in Montavon and Muller (2012), but were otherwise held fixed during training.

all algorithms on minibatch sizes of either 25, 128 or 256 elements ⁱ. Finally, since we are comparing optimization algorithms, hyper-parameters were chosen based on the training set likelihood (though we still report the associated test errors). All experiments used the MinRes linear solver, both for its speed and its ability to return pseudo-inverses when faced with ill-conditioning.

Figure 8.1 (left) shows the likelihood as estimated by Annealed Importance Sampling (Salakhutdinov and Hinton, 2009a; Neal, 2001) as a function of the number of epochs ⁱⁱ. Under this metric, MFNG achieves the fastest convergence, obtaining a training/test set likelihood of $-71.26/-72.84$ nats after 94 epochs. In comparison, MFNG-diag obtains $-73.22/-74.05$ nats and SML $-80.12/-79.71$ nats in 100 epochs. The picture changes however when plotting likelihood as a function of CPU-time, as shown in Figure 8.1 (right). Given a wall-time of 8000s for MFNG and SML, and 5000s for MFNG-diag ⁱⁱⁱ, SML is able to perform upwards of 1550 epochs, resulting in an impressive likelihood score of $-64.94 / -67.73$. Note that these results were obtained on the binary-version of MNIST (thresholded at 0.5) in order to compare to Montavon and Müller (2012). These results are therefore not directly comparable to Salakhutdinov and Hinton (2009a), which binarizes the dataset through sampling (by treating each pixel activation as the probability p of a Bernoulli distribution).

Figure 8.2 shows a breakdown of the algorithm runtime, for various components of the algorithm. These statistics were collected in the early stages of training, but are generally representative of the bigger picture. While the linear solver clearly dominates the runtime, there are a few interesting observations to make. For small models and batch sizes greater than 256, a single evaluation of Ly appears to be of the same order of magnitude as a gradient evaluation. In all cases, this cost is smaller than that of sampling, which represents a non-negligible part of the total computational budget. This suggests that MFNG could become especially attrac-

i. We expect larger minibatch sizes to be preferable, however simulating this number of Markov chains in parallel (on top of all other memory requirements) was sufficient to hit the memory bottlenecks of GPUs.

ii. While we do not report error margins for AIS likelihood estimates, the numbers proved robust to changes in the number of particles and temperatures being simulated. To obtain such robust estimates, we implemented all the tricks described in Salakhutdinov and Hinton (2009a) and Salakhutdinov and Murray (2008): p_A a zero-weight base-rate model whose biases are set by maximum likelihood; interpolating distributions $p_i \propto p_A^{(1-\beta_i)} p_B^{(\beta_i)}$, with p_B the target distribution; and finally analytical integration of all odd-layers.

iii. This discrepancy will be resolved in the next revision.

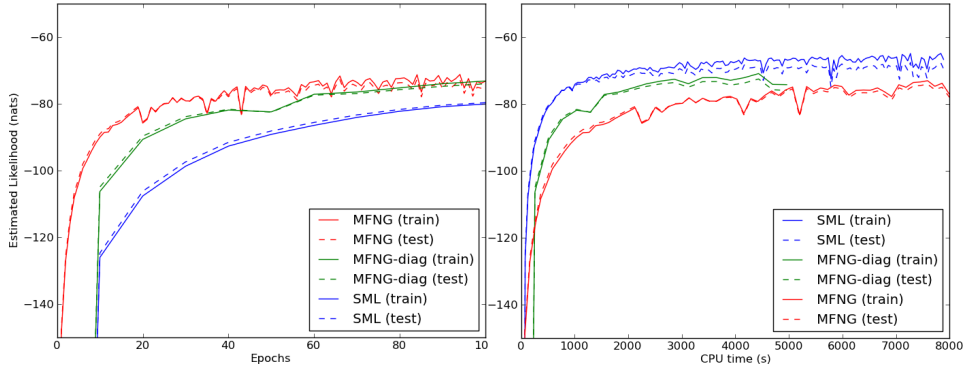


Figure 8.1: Estimated model likelihood as a function of (left) epochs and (right) CPU-time for MFNG, its diagonal approximation (MFNG-diag) and SML. All methods were run in conjunction with the DBM centering trick (Montavon and Muller, 2012), with centering coefficients held fixed during training. Our grid search yielded the following hyper-parameters: batch size of 256/128 for MFNG(-diag)/SGD; 5 steps of mean-field / sampling-based inference for MFNG(-diag)/SGD and a learning rate of $5 \cdot 10^{-3}$.

tive for models which are expensive to sample from. Overall however, restricting the number of CG/MinRes iterations appears key to computational performance, which can be achieved by increasing the damping factor α . How this affects convergence in terms of likelihood is left for future work.

8.5 Discussion and Future Work

While the wall-clock performance of MFNG is not currently competitive with SML, we believe there are still many avenues to explore to improve computational efficiency. Firstly, we performed almost no optimization of the various MinRes hyper-parameters. In particular, we ran the algorithm to convergence with a fixed tolerance of 10^{-5} . While this typically resulted in relatively few iterations (around 15), this level of precision might not be required (especially given the stochastic nature of the algorithm). Additionally, it could be worth exploiting the same strategy as HF where the linear solver is initialized by the solution found in the previous iteration. This may prove much more efficient than the current approach of initializing the solver with a zero vector. Pre-conditioning is also a well-known method for accelerating the convergence speed of linear solvers (Chapelle and Erhan, 2011).

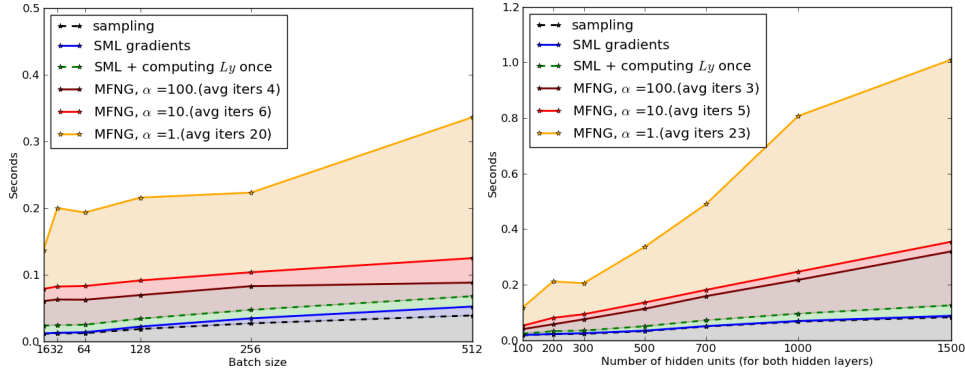


Figure 8.2: Breakdown of algorithm runtime, when we vary (left) the batch size (with fixed model architecture $784 - 400 - 100$) and (right) the model size (with fixed batch size of 256). Runtime is additive in the order given by the labels (top to bottom). Dotted lines denote intermediate steps, while continuous lines denote full steps. Data was collected on a Nvidia GTX 480 card.

Our implementation used a simple diagonal regularization of L . The Jacobi preconditioner could be implemented easily however by computing the diagonal of L in a first-pass.

Finally, while our single experiment offers little evidence in support of either conclusion, it may very well be possible that MFNG is simply not computationally efficient for DBMs, compared to SML with centering. In this case, it would be worth applying the method to either (i) models with known ill-conditioning, such as factored 3-rd order Boltzmann Machines or (ii) models and distributions exhibiting complex posterior distributions. In such scenarios, we may wish to maximize the use of the positive phase statistics (which were obtained at a high computational cost) by performing larger jumps in parameter space. It remains to be seen how this would interact with SML, where the burn-in period of the persistent chains is directly tied to the magnitude of $\Delta\theta$.

9

Prologue to Fourth Article

9.1 Article Details

The Spike-and-Slab RBM and Extensions to Discrete and Sparse Data Distributions. Aaron Courville, Guillaume Desjardins, James Bergstra and Yoshua Bengio. To appear in IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2014.

Personal Contribution. My contributions to this paper were in the extensions of the ssRBM to binary and spike & slab input data, along with the development of the subspace ssRBM, the insights of which were gleaned from my work on the bilinear ssRBM of Chapter 12. In particular, I implemented the code and ran all of the experiments of Sections 10.7.2, 10.8.2 and 10.9, and wrote the associated sections of the paper. I also contributed heavily to solving the instability issues of the subspace ssRBM which led to us modeling the visible units as truncated Gaussians.

9.2 Context

Higher-order statistics play a crucial role in natural image statistics (Simoncelli and Olshausen, 2001). As such, they should be a focus of representation learning. While hierarchical representations offer a powerful framework for doing so, it may be more efficient (statistically speaking) to capture or learn to be invariant to simple concepts, like local pixel correlations, in lower layers of the hierarchy or even within a single stage of feature extraction.

Earlier work on this topic in the areas of feed-forward neural networks include Minsky and Papert (1969); Giles and Maxwell (1987) and Bergstra et al. (2009) to name a few. In the context of unsupervised learning via energy-based models,

these include the mean-covariance (Ranzato and Hinton, 2010) and mean-“Product of Student-t” (mPoT) RBM (Ranzato et al., 2010), which modified the RBM energy function to include squares of a (linear) data projection term and as such belong to the more generally family of 3-rd order BMs (see Section 2.6.1). Courville et al. (2011a,b) provided an alternative solution to capturing higher-order statistics: augmenting each binary random variable with a continuous slab component.

The subsequent chapter is a journal submission dedicated to this model family. It is a superset of the results found in these earlier papers, along with novel extensions of the model.

9.3 Contributions

The following paper offers a unified view of the spike & slab RBM family of models. The ssRBM offers a flexible framework for density modeling, in which the learnt features are sensitive to both first and second order statistics of the input. This framework is shown to be flexible in that it can be adapted to cover various input distributions: Bernoulli, multinomial, spike & slab or normal. This is in stark contrast to the mean-covariance RBM (mcRBM) which has been limited to modeling continuous data, due to its reliance on Hybrid Monte Carlo (HMC) for sampling.

In addition to previously published results, our contributions highlight the importance of learning *rich invariant features* for classification. To this end, we relied exclusively on the ssRBM to perform feature extraction, followed by a simple linear SVM for classification (and hence no fine-tuning). This allows us to independently assess the quality of the learnt features.

Our results are clear. By capturing properties of both the mean and covariance of input pixels, the binary input ssRBM is able to significantly outperform a vanilla RBM, while a 2-layer spike & slab DBN is able to match the results of Hinton et al. (2006) *without the need for fine-tuning*. Similarly, we show how the subspace ssRBM can lead to increased invariance of the latent representations, which directly translates to improved performance in the low-labeled data regime.

These results paint a coherent picture. Rich, invariant representations can be learnt in an unsupervised manner. These can then serve as input to classifiers having a reduced model complexity, a clear advantage when working with small labeled datasets. While this is in stark contrast to the current trend of learning large feed-forward classification networks via loads of supervised data ([Krizhevsky et al., 2012](#); [Goodfellow et al., 2013](#)), we believe it is important to keep the above conclusion in mind moving forward, as limited access to labeled data (even in the limit of one-shot learning) might prove to be more representative of the general AI problem.

The Spike-and-Slab RBM and Extensions to Discrete and Sparse Data Distributions

THE *spike-and-slab* RESTRICTED BOLTZMANN MACHINE (ssRBM) is defined by having both a real-valued “slab” variable and a binary “spike” variable associated with each unit in the hidden layer. The ssRBM uses its slab variables to model the conditional covariance of the observation – thought to be important in capturing the statistical properties of natural images. In this paper, we present the canonical ssRBM framework together with some extensions of the model. These extensions highlight the flexibility of the spike-and-slab RBM as a platform for exploring more sophisticated probabilistic models of high dimensional data in general and natural image data in particular. Here, we introduce the subspace ssRBM focused on the task of learning invariant features. We highlight the behaviour of the ssRBM and its extensions through experiments with the MNIST digit recognition task and the CIFAR-10 object classification task.

10.1 Introduction

Unsupervised feature learning for natural images is presently the subject of intense research. Approaches to object recognition (Coates et al., 2011; Coates and Ng, 2011a,b), scene analysis (Socher et al., 2011) and activity recognition (Le et al., 2011) have largely converged on a classification pipeline that begins with at least one feature extraction phase. While standard feature extraction schemes such as SIFT (Lowe, 1999) are popular, superior performance has been demonstrated by incorporating *learned features*.

A large variety of modeling paradigms have been applied to the problem, including autoencoders (Vincent et al., 2008; Rifai et al., 2011), sparse coding (Olshausen and Field, 1996), and energy-based models. One of the most popular energy-based modeling paradigms for unsupervised feature learning is the restricted Boltzmann

machine (RBM). An RBM is a Markov random field with a bipartite graph structure consisting of a visible layer and a hidden layer. The bipartite structure excludes connections between the variables within each layer so that the latent variables are conditionally independent given the visible variables and *vice versa*. The factorial nature of these conditional distributions enables efficient Gibbs sampling which forms the basis of the most popular RBM learning algorithms such as contrastive divergence (Hinton, 2002) and stochastic maximum likelihood (Tieleman, 2008).

Both as a feature learning scheme (Ranzato and Hinton, 2010) and especially as a generative model of natural images (Ranzato et al., 2010, 2011; Kivinen and Williams, 2012), RBM-based methods have shown considerable promise. As the canonical energy model for real-valued data, the Gaussian RBM has long been popular as a means of extracting features from natural image data. However, recently Ranzato et al. (2010) have argued that the Gaussian RBM inductive bias is not well suited to the statistical variations present in natural image data. In response to these insights, several alternative models have been proposed to better account for the kinds of variation we see in natural images. These include the mean and covariance RBM (mcRBM) and the mean-product of t-distribution (mPoT) model. Unlike the Gaussian RBM which uses its hidden units to encode the conditional mean of pixels, these models use their hidden units to encode the conditional covariance of the pixels. One drawback of both of these models is that, unlike the standard RBM, the conditional distribution over the observation given the hidden units is not factorial. As a result, the usual (and efficient) inference and training strategies are not compatible with these models.

In this paper, we develop the spike-and-slab RBM (ssRBM). The ssRBM is defined as having each hidden unit associated with the product of a binary *spike* latent variable and a real-valued *slab* latent variable. The name *spike-and-slab* is inspired from terminology in the statistics literature (Mitchell and Beauchamp, 1988), where the term refers to a prior consisting of a mixture between two components: the spike, a discrete probability mass at zero; and the slab, a density (typically uniformly distributed) over a continuous domain.

Spike-and-slab models have previously been explored in the context of factor analyzer-like directed graphical models (Garrigues and Olshausen, 2008; Zhou et al., 2009; Lücke and Sheikh, 2011; Mohamed et al., 2011; Titsias and Lázaro-Gredilla, 2011; Goodfellow et al., 2012) as well as in a hierarchical extension of such mod-

els (Hinton et al., 1998). The primary advantage that our ssRBM offers over these directed spike-and-slab models is its comparative ease of inference. The ssRBM shares the RBM’s well-known efficient posterior computation, making it an appropriate basis for scalable representation learning.

As a model of natural images, the ssRBM is interesting in that, like the mcRBM and the mPoT model, its binary hidden units encode the conditional covariance of the pixels while simultaneously maintaining the simple conditional independence structure that underlies efficient learning and inference in the traditional RBM. In the ssRBM, this is accomplished by exploiting real-valued latent slab variables, as was done in Martens and Sutskever (2010). Marginalizing over these variables results in the conditional covariance being parametrized by the binary hidden units. However, conditioning on the slab variables recovers the traditional RBM conditional independence structure.

In this paper, we present the canonical ssRBM framework together with several extensions of the model. These extensions demonstrate the flexibility of the spike-and-slab RBM as a platform for exploring more sophisticated probabilistic models of high dimensional data. In particular, we present variations of the ssRBM model to binary data and to sparse real-valued data that can be represented as spike-and-slab data (either real-valued or exactly zero). Finally, we also present an extension of the ssRBM termed the subspace ssRBM that ties single binary “spike” variables to subspaces of the observation space. These subspaces are defined through sets of feature vectors, each associated with a slab variable, that span the subspaces. We demonstrate how learned feature subspaces can improve the performance of the extracted features, particularly when the number of training examples is low.

10.1.1 The Gaussian RBM as a Model of Images

The Gaussian RBM is the simplest RBM that models real-valued data. Taking the number of hidden units to be N and dimensionality of the input to be D , we let $h_i \in \{0, 1\}$ for i ranging from 1 to N denote the binary hidden units and $x \in \mathcal{R}^D$ denote a single real-valued observation vector. Assuming that x is drawn from a centered distribution, the Gaussian RBM is specified by the energy function:

$$E(x, h) = \frac{1}{2}x^T \Lambda x - \sum_{i=1}^N x^T W_i h_i - \sum_{i=1}^N b_i h_i, \quad (10.1)$$

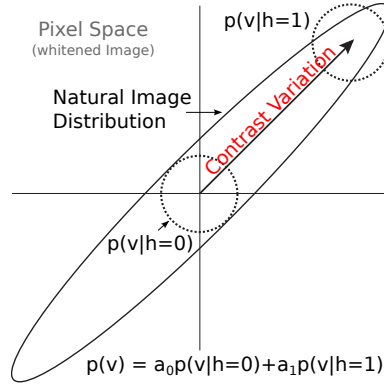


Figure 10.1: GRBM inductive bias

An illustration of the Gaussian RBM inductive bias (with one hidden unit) and the statistical structure of natural images. The Gaussian RBM exhibits undesirable sensitivity to local variation in contrast.

where W_i is the weight vector of the i th hidden unit, Λ is the diagonal precision matrix on the visible units, and b are the hidden unit biases. From $E(x, h)$ we can derive the conditional distribution over the inputs given the hidden units:

$$p(x | h) = \mathcal{N}(Wh, \sigma \mathbf{I}), \quad (10.2)$$

where $\mathcal{N}(\mu, \Sigma)$ denotes a Gaussian distribution with mean μ and covariance Σ . We can interpret the Gaussian RBM marginal as a Gaussian mixture model with each setting of the hidden units specifying the position of a mixture component. While the number of mixture components scales exponentially with N , these components share a set of parameters that only scales linearly with N .

Ranzato and Hinton (2010) suggest that the Gaussian RBM is unsatisfactory as a model of natural images because of the model's constant conditional covariance. They argue that the relevant statistics of natural images are captured by the covariance of pixel values rather than absolute pixel values. This point is supported by the widespread use of preprocessing methods that standardize the global scaling of pixel values across each image in a dataset. Figure 10.1 illustrates the mismatch between the Gaussian RBM inductive bias and a simplified view of the kind of variation we would expect to see in natural images.

The inductive bias of the Gaussian RBM has real consequences on its ability to model natural images. Figure 10.1 illustrates the effect of this bias on an inpainting

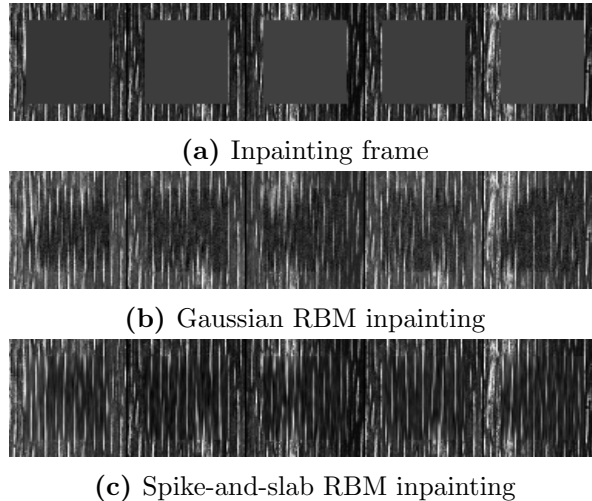


Figure 10.2: Inpainting using the GRBM vs. spike & slab RBM

(a) Image of Brodatz texture D68, the top half of which was used for training, and inpainting frames. (b) Inpainting with the Gaussian RBM (trained tile-convolutionally – see [Luo et al. \(2013\)](#) for details). (c) Inpainting with the ssRBM model in the identical training and text configuration as the Gaussian RBM.

task. The model was trained on a natural texture, shown in Figure 10.2(a), in a tiled-convolutional configuration (see [Luo et al. \(2013\)](#) for details). Figure 10.2(b) shows the results of inpainting with the Gaussian RBM. The inpainted regions exhibit a clearly visible mismatch in contrast at the border of the inpainted region of the image. The spike-and-slab RBM, on the other hand, is better able to match the contrast at the boundary of the inpainted region, as shown in 10.2(c).

Despite its disadvantages, the Gaussian RBM has the significant advantage that it preserves the classic RBM property that the two conditionals, $p(x | h)$ and $P(h | x)$ are factorized. Alternative energy formulations such as [Ranzato et al. \(2010\)](#) and [Ranzato and Hinton \(2010\)](#) trade this important property away in order to model more interesting (at least, non-diagonal) covariance between pixel values. Consequently, the Gaussian RBM remains a standard approach to modeling images and other continuous data in the RBM framework.

10.2 The Spike-and-Slab RBM

In this section we present the spike-and-slab restricted Boltzmann machine (ssRBM): an extension of the RBM framework designed to improve on the Gaussian RBM as a model of natural images. The ssRBM specifies interactions between three random vectors: the vector x representing the observed data, the latent binary “spike” variables h and the latent real-valued “slab” variables s . The i th hidden unit is associated with the product of the spike element h_i and the slab element s_i of the real-valued variable. Combined, their product gives each hidden unit a sparse value, i.e. either real-valued or zero. Let there be N hidden units: $h \in [0, 1]^N$, $s \in \mathbb{R}^N$ and an observation vector of dimension D : $x \in \mathbb{R}^D$. The ssRBM model is defined via the energy function:

$$\begin{aligned}
 E(x, s, h) = & - \sum_{i=1}^N x^T W_i s_i h_i + \frac{1}{2} x^T \left(\Lambda + \sum_{i=1}^N \Phi_i h_i \right) x \\
 & + \frac{1}{2} \sum_{i=1}^N \alpha_i s_i^2 - \sum_{i=1}^N \alpha_i \mu_i s_i h_i - \sum_{i=1}^N b_i h_i + \sum_{i=1}^N \alpha_i \mu_i^2 h_i, \quad (10.3)
 \end{aligned}$$

where, as with the Gaussian RBM W_i denotes the i th weight vector ($W_i \in \mathbb{R}^D$), b_i is the bias of spike h_i , and Λ is a diagonal precision matrix on the observations x . The ssRBM, though, introduces additional parameters beyond those in the Gaussian RBM. Namely, each $\alpha_i > 0$ is a scalar precision parameter for the real-valued slab variable s_i ; each Φ_i is a non-negative diagonal matrix that defines an h -dependent quadratic penalty on x ; and each μ_i is a mean parameter for the slab variable s_i . With the energy function thus defined, the joint probability distribution over the model variables is given by:

$$p(x, s, h) = \frac{1}{Z} \exp \{-E(x, s, h)\} \quad (10.4)$$

where Z is the normalizing partition function. The ssRBM joint distribution has the very important property that it corresponds to the standard RBM bipartite graph structure with the distinction that the hidden units are considered to form N cliques consisting of paired spike and slab variables h_i and s_i .

One way to understand the ssRBM model is to consider the form of its various conditional distributions. We will begin by comparing the conditional form of

$p(x | h)$ as it arises in the Gaussian RBM (recall Eq. 10.2) and the ssRBM models. In the ssRBM, we recover this conditional by marginalizing out the slab variables s :

$$\begin{aligned} p(x | h) &= \frac{1}{P(h)} \frac{1}{Z} \int \exp \{-E(x, s, h)\} ds \\ &= \mathcal{N} \left(C_{x|h} \sum_{i=1}^N W_i \mu_i h_i, C_{x|h} \right) \end{aligned} \quad (10.5)$$

where $C_{x|h} = \left(\Lambda + \sum_{i=1}^N \Phi_i h_i - \sum_{i=1}^N \alpha_i^{-1} h_i W_i W_i^T \right)^{-1}$. The last equality holds only if the covariance matrix $C_{x|h}$ is positive definite, which is not guaranteed from the parametrization. In Section 10.3, we will discuss a few strategies, via constraints on Λ and Φ , to ensure positive definiteness of $C_{x|h}$.

From Eq. 10.5 we see that, like the Gaussian RBM, the conditional $p(x | h)$ is Gaussian-distributed, however unlike the Gaussian RBM, the hidden units not only encode the conditional mean but also specify the conditional covariance $C_{x|h}$. Delving a bit deeper into the parametrization of the conditional $p(x | h)$, we see that the conditional mean of x given h and principal axis of conditional covariance are related: active hidden variables ($h_i = 1$) for which μ_i is relatively large will tend to align the mean with the principal axes of variance, whereas hidden variables for which μ_i is close to zero will only affect the directions of conditional variance. This flexibility for the model to *adaptively* assign capacity to model either the conditional mean or the conditional variance represents an innovation of the ssRBM over previous work.

While the conditional $p(x | h)$ demonstrates that the ssRBM is appropriately parametrized for natural image modeling in that its conditional covariance is fully general (i.e. not restricted to be diagonal as in the Gaussian RBM), the non-diagonal covariance has another immediate consequence for learning: unlike the Gaussian RBM, the elements of the ssRBM conditional $p(x | h)$ are not independent. This implies we cannot sample easily and efficiently from this conditional using block Gibbs sampling. It might seem we have gained modeling power at the expense of a more challenging sampling scenario (options include Hybrid Monte Carlo (Neal, 1993), HMC). However, in the case of the ssRBM, another option is available.

In addition to marginalizing out the slab variables s , it is also enlightening to

condition on them, so that:

$$\begin{aligned} p(x | s, h) &= \frac{1}{p(s, h)} \frac{1}{Z} \exp \{-E(x, s, h)\} \\ &= \mathcal{N} \left(C_{x|s,h} \sum_{i=1}^N W_i s_i h_i, C_{x|s,h} \right) \end{aligned} \quad (10.6)$$

where $C_{x|s,h} = \left(\Lambda + \sum_{i=1}^N \Phi_i h_i \right)^{-1}$. That is, the conditional distribution of x given both s and h is, once again, Gaussian distributed. Critically though, with diagonal Λ and Φ_i ($\forall i \in [1, N]$), the conditional covariance $C_{x|s,h}$ is also diagonal. The diagonal covariance allows us to sample from $p(x | s, h)$ using block Gibbs sampling. Equation 10.16 also shows the role played by Φ_i in augmenting the precision with the activation of h_i . Indeed, hidden unit i contributes a component not only to the mean proportional to $W_i s_i$, but also to the global scaling of the conditional mean.

The next conditional distribution we consider is the conditional over the slabs s given the spikes h and the observation x :

$$p(s | x, h) = \prod_{i=1}^N \mathcal{N} \left((\alpha_i^{-1} x^T W_i + \mu_i) h_i, \alpha_i^{-1} \right). \quad (10.7)$$

As was the case with $p(x | s, h)$, the conditional $p(s | x, h)$ is once again Gaussian distributed with diagonal covariance. Equation 10.7 also shows how the mean of the slab variable s_i , given $h_i = 1$, is linearly dependent on x , and as the precision $\alpha_i \rightarrow \infty$, s_i converges in probability to μ_i .

Finally we consider the conditional distribution over the latent spike variables h given the observations. When we marginalize over s , this time we find that the conditional does factorize, i.e. $P(h | x) = \prod_i^N P(h_i | x)$ with

$$\begin{aligned} P(h_i = 1 | x) &= \frac{1}{p(x)} \frac{1}{Z_i} \int \exp \{-E(x, s, h)\} ds \\ &= \text{sigm} \left(\frac{1}{2} \alpha_i^{-1} (x^T W_i)^2 + x^T W_i \mu_i - \frac{1}{2} x^T \Phi_i x + b_i \right), \end{aligned} \quad (10.8)$$

where sigm indicates a logistic sigmoid. Equation 10.8 shows the interaction between three data-dependent terms. The first term, $\frac{1}{2} \alpha_i^{-1} (x^T W_i)^2$, is the contribution due to the variance in s about its mean (note the scaling with α_i^{-1}) and

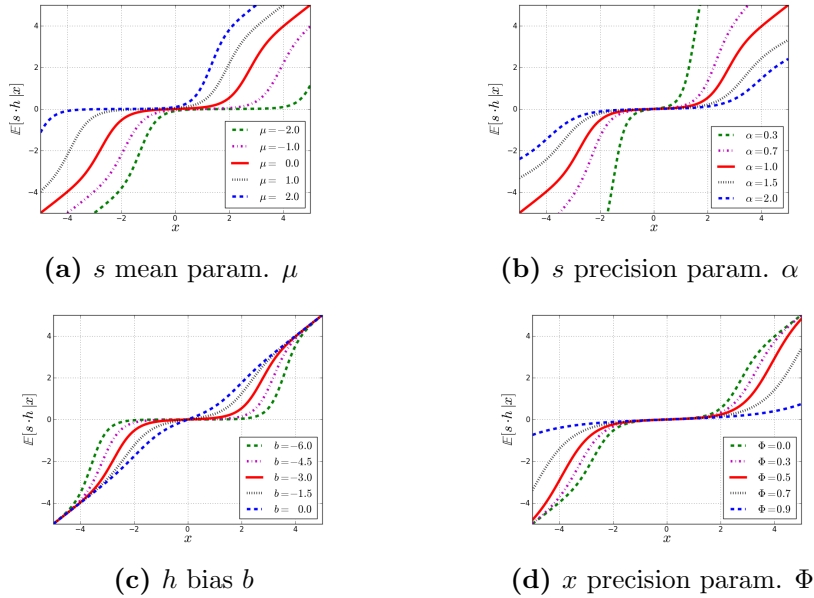


Figure 10.3: Sensitivity analysis of the ssRBM activation function.

The sensitivity of the spike-and-slab RBM feature activation curves, given by $\mathbb{E}[s \cdot h | x]$, to changes to model parameters (in the case of a one dimensional input v). Default values for the model parameters are: $\mu = 0.0$, $\alpha = 1.0$, $b = -3.0$ and $\Phi = 0.0$.

Unless specified by the legend, all parameter are at their default values.

appears in the sigmoid as a result of marginalizing out s . This term is always non-negative, so it always acts to increase $P(h_i | x)$. Countering this tendency to activate h_i is the other term quadratic in x , $-\frac{1}{2}x^T\Phi_i x$, that is always a non-positive contribution to the sigmoid argument. In addition to these two quadratic terms, there is the term $x^T W_i \mu_i$ whose behaviour mimics the data-dependent term in the analogous Gaussian RBM version of the conditional distribution over h : $P_{\text{GRBM}}(h_i | x) = \text{sigm}(x^T W_i + b_i)$.

One of the interesting aspects of the model is that, when using the model as a feature learning mechanism, the ssRBM offers a choice in the feature representation of the data. One natural choice is the expected product of the spike and slab variables, $\mathbb{E}[s \cdot h | x]$. This choice has the potential advantage that, like the RBM with rectified linear hidden units (Nair and Hinton, 2010), the feature activations scale with the intensity of activation, through the action of the slab variable s . In this way, the feature response can be said to be *equivariant* with respect to changes

in intensity. Figure 10.3 shows how the quantity $\mathbb{E}[s \cdot h \mid x]$ changes as a function of the model parameters. On the other hand, sometimes it is desirable to be *invariant* with respect to changes in the intensity of activation. By taking $P(h \mid x)$ as the feature representation, the model can encode correlation patterns while remaining relatively insensitive to local changes in intensity. In the case of natural images, intensity often reflects illumination conditions and contrast levels – factors that are often irrelevant to tasks of interest such as object classification (Bergstra et al., 2011).

10.3 Positive Definite Constraints

The conditional $p(x \mid h)$ is only a well-defined Gaussian distribution if the covariance matrix $C_{x|h}$ is positive definite (PD). However, as previously noted the form of $C_{x|h}$ (in Eq. 10.5) is not parametrized to guarantee that this condition is met. In particular, if there exists an x such that $x^T C_{x|h} x \leq 0$, then the covariance matrix is not PD.

One way to deal with this issue is to restrict the domain of x to a finite box or ball that encompasses all training data. In the case of the box constraint, this would imply replacing the conditional Gaussian distribution of the visible variables in eqn. 10.16 with a truncated Gaussian distribution. In some cases (including the case of pixel arrays from a CCD sensor) this restriction may be natural, because there are physical reasons why observation vectors must necessarily be limited in magnitude. Generally though, it would be preferable not to fix the modeling domain *a priori*, and instead permit the training algorithm free reign over all of \mathbb{R}^D . To that end, this section examines several techniques for constraining the basic ssRBM so that the conditional covariance $C_{x|h}$, or equivalently the conditional precision matrix, remains PD.

Specifically, we wish to constrain:

$$x^T C_{x|h}^{-1} x > 0 \quad \forall x \neq \mathbf{0}$$

That is, we need to ensure that $\Lambda + \sum_{i=1}^N \Phi_i h_i$ is, in some sense, large enough to offset $\sum_{i=1}^N \alpha_i^{-1} W_i W_i^T h_i$. Here we consider two basic strategies: (1) define Λ to be

large enough to offset a worst-case setting of the h ; and (2) define the Φ_i to ensure that the contribution of each active h_i is itself PD.

10.3.1 Constraining Λ

One option to ensure that $C_{x|h}$ remains PD for all patterns of h activation is to constrain Λ to be large enough. In setting a constraint on Λ , we will ignore the contribution of the Φ_i terms (which leads to non-tightness of the constraint). Since the contribution of every $\alpha_i^{-1}W_iW_i^Th_i$ term is negative semi-definite, the worst case setting of the h would be to have $h_i = 1$ for all $i \in [1, N]$. This implies that Λ must be constrained such that:

$$x^T \left(\Lambda - \sum_{i=1}^N \alpha_i^{-1} W_i W_i^T \right) x > 0 \quad \forall x \neq \mathbf{0}. \quad (10.9)$$

If we constrain Λ to be a scalar matrix, i.e. $\Lambda = \lambda I$, then the problem of enforcing a PD precision matrix reduces to ensuring that λ is greater than the maximum eigenvalue, ρ , of $\sum_{i=1}^N \alpha_i^{-1} W_i W_i^T$. In practice we use the power iteration method to quickly estimate an upper bound on the maximum eigenvalue, and then constrain $\lambda > \rho$ throughout training.

10.3.2 Constraining Φ_i

Another option to ensure that $C_{x|h}$ remains PD for all patterns of h activation is to constrain Φ_i to be large enough to ensure that the contribution of each h_i is PD. Let W_{ij} be the j th element of the filter W_i (or equivalently, the ij th element of the weight matrix W) and let Φ_{ij} denote the jj th element of the diagonal Φ_i matrix. We can ensure that $C_{x|h}$ is positive definite if we constrain Φ_{ij} either as¹:

$$\Phi_{ij} = \zeta_{ij} + \alpha^{-1} \sum_j W_{ij}^2 I, \quad (10.10)$$

where Φ_{ij} takes the form of a scalar matrix, or as

$$\Phi_{ij} = \zeta_{ij} + \alpha^{-1} D W_{ij}^2, \quad (10.11)$$

i. Courville et al. (2011b) present the details of the derivation of this constraint.

where the j th elements on the diagonal of Φ_i is scaled with W_{ij}^2 (recall D is the dimension of the observation vector). In both cases, the parameter $\zeta_{ij} > 0$ provides an extra degree of freedom to Φ_{ij} to be estimated through maximum likelihood learning. These are of course not the only option for parametrization of Φ_i . However they are particularly simple constraints that offer complimentary behaviour. In the case of Φ_{ij} parametrized as in Eq. 10.10, the presence of the $\sum_i^N \Phi_i h_i$ as a scaling term implies that the activation of any h_i will have an effect on the scaling of the mean across the entire observation vector irrespective of how localized is the corresponding filter W_i . Unsurprisingly, use of this parametrization tends to encourage both sparse activation of h_i and W_i having relatively large receptive fields. The parametrization of Φ_{ij} as in Eq. 10.11 has the property that the Φ_i receptive fields are steered in the direction of W_i . Where W_i is near zero, Φ_i is relatively unconstrained. This is an appealing property for modeling images or other data that give rise to sparse receptive fields W_i . We will empirically explore these constraints and their effect on the ssRBM as a feature learning and extraction algorithm.

10.4 Learning in the ssRBM

As is typical of RBM-styled models, learning in the ssRBM is rooted in the ability to efficiently draw samples from the model via block Gibbs sampling. As previously discussed, the conditionals $P(h | x)$, $p(x | h)$, $p(s | x, h)$ and $p(x | s, h)$ possess some important properties with regard to sampling. First, consider the standard RBM Gibbs sampling scheme of iteratively sampling from $P(h | x)$ and $p(x | h)$ with s marginalized out. Sampling from $P(h | x)$ is straightforward, as Equation 10.8 indicates that the h_i are all conditionally independent given x . Under the assumption of a positive definite covariance matrix, the conditional distribution $p(x | h)$ is multivariate Gaussian with non-diagonal covariance $C_{x|h}$. As previously discussed, sampling from $p(x | h)$ would require the calculation of the covariance matrix (via matrix inverse) with every weight update. Fortunately, in the case of the ssRBM, rather than sampling directly from $p(x | h)$, we can sample the slab vector from $p(s | h, x)$, which is Gaussian distributed with diagonal covariance. Then, given both s and h , we can sample x from the conditional $p(x | s, h)$,

which is also Gaussian distributed with diagonal covariance. Taken all together the triplet $P(h | x)$, $p(s | x, h)$ and $p(x | s, h)$ forms a three-phase block-Gibbs sampling scheme that allows us to sample efficiently from the ssRBM.

In training, we use the stochastic maximum likelihood algorithm (SML, also known as persistent contrastive divergence) (Tieleman, 2008). We follow the data log likelihood gradient $\frac{\partial}{\partial \theta_i} \left(\sum_{t=1}^T \log p(x_t) \right)$, is:

$$-\sum_{t=1}^T \left\langle \frac{\partial}{\partial \theta_i} E(x_t, s, h) \right\rangle_{p(s, h | x_t)} + T \left\langle \frac{\partial}{\partial \theta_i} E(x, s, h) \right\rangle_{p(x, s, h)}$$

The log likelihood gradient takes the form of a difference between two expectations, over $p(s, h | x_t)$ in the “clamped” condition, and over $p(x, s, h)$ in the “unclamped” condition. As with the standard RBM, the expectations over $p(s, h | x_t)$ are amenable to analytic evaluation. The expectations over the model distribution $p(x, s, h)$ is approximated by samples drawn from the ssRBM three-phase Block Gibbs sampler. Typically in SML, only one or a few Markov Chain (Gibbs) simulations are performed between each parameter update.

10.5 Related Models of Conditional Covariance

As discussed in the introduction, there are other Boltzmann Machine-based models with the goal of modeling the kind of statistical structure found in natural images. For instance, as previously discussed, RBMs with rectified linear hidden units (Nair and Hinton, 2010) possess a similar equivariance of intensity as the ssRBM slab variables. However, unlike the model of Nair and Hinton (2010), the ssRBM is expressed naturally as a simple energy function. This allows us to consider simple extensions of the model that we consider in later sections. The models that are most closely related to the ssRBM are the mcRBM (Ranzato and Hinton, 2010) and the mPoT model (Ranzato et al., 2010). Here we briefly review these models and compare them to the spike-and-slab RBM.

10.5.1 The Mean and Covariance RBM

Similar to the ssRBM, mean and covariance RBM (mcRBM) is a Boltzmann machine that models the observation x as a multivariate Gaussian distributed quantity with general covariance structure. However it does so via a very different mechanism. The mcRBM uses its hidden layer to independently parametrize both the mean and covariance of the data through two sets of binary hidden units. The model combines the covariance RBM (cRBM) (Ranzato et al., 2010) with the Gaussian RBM. The cRBM components model the conditional covariance structure, with the Gaussian RBM capturing the conditional mean. With N_c covariance units: $h^c \in \{0, 1\}^{N_c}$, and N_m mean units: $h^m \in \{0, 1\}^{N_m}$, the mcRBM model is defined via the energy function:

$$E_{\text{mc}}(x, h^c, h^m) = -\frac{1}{2} \sum_{j=1}^{N_c} h_j^c (x^T C_j)^2 - \sum_{j=1}^{N_c} b_j^c h_j^c + E_m(x, h^m),$$

where C_j is the weight vector associated with covariance unit h_j^c and b^c is a vector of covariance unit biases. The energy function defines a conditional distribution over the observations given h^m and h^c with a fully general multivariate Gaussian distribution:

$$p_{\text{mc}}(x | h^m, h^c) = \mathcal{N} \left(\Sigma \left(\sum_{j=1}^{N_m} W_j h_j^m \right), \Sigma \right), \quad (10.12)$$

with covariance matrix $\Sigma = \left(\sum_{j=1}^{N_c} h_j C_j C_j^T + \mathbf{I} \right)^{-1}$. The conditional distributions over the binary hidden units h_i^m and h_i^c form the basis for the feature representation in the mcRBM and are given by:

$$P_{\text{mc}}(h_i^m = 1 | x) = \text{sigm} \left(\sum_{i=1}^{N_m} W_i h_i^c - b_i^m \right),$$

$$P_{\text{mc}}(h_j^c = 1 | x) = \text{sigm} \left(\frac{1}{2} \sum_{j=1}^{N_c} h_j^c (x^T C_j)^2 - b_j^c \right),$$

The mcRBM can be trained using contrastive divergence or SML which require the ability to draw samples from the model. However, due to its non-diagonal conditional covariance structure, sampling from $p_{\text{mcRBM}}(x | h^m, h^c)$ would require computing the Σ^{-1} at every iteration of learning. This leads to an impractical

computational burden for even moderately sized observations. [Ranzato and Hinton \(2010\)](#) avoid direct sampling from the conditional 10.12 by sampling directly from the marginal $p(x)$ using hybrid Monte Carlo ([Neal, 1993](#)) on the the mcRBM free energy.

10.5.2 Mean - Product of Student's T-distributions

The mean-product of Student's t-distribution (mPoT) model ([Ranzato et al., 2010](#)) extends the PoT model ([Welling et al., 2003](#)) in a manner similar to how the mcRBM extends the cRBM. Specifically by include nonzero Gaussian means by the addition of Gaussian RBM-like hidden units. The PoT model is an energy-based model where the conditional distribution over the observation is a multivariate Gaussian (non-diagonal covariance) and the complementary conditional distribution over the hidden variables are a set of conditionally independent Gamma distributions. The mPoT energy function is given as:

$$E_{\text{mp}}(x, h^m, h^c) = E_m(x, h^m) + \sum_j \left(h_j^c \left(1 + \frac{1}{2} (C_j^T x)^2 \right) + (1 - \gamma_j) \log h_j^c \right)$$

where C_j is the weight vector associated with covariance unit h_j^c . The mPoT model energy function specifies a multivariate Gaussian conditional distribution over x with non-diagonal covariance. While the covariance units h^c are conditionally Gamma-distributed:

$$P_{\text{mp}}(h_j^c | x) = \mathcal{G} \left(\gamma_j, 1 + \frac{1}{2} (C_j^T x)^2 \right)$$

As both the mPoT model and mcRBM give rise to multivariate Gaussian conditional distributions over the observations with non-diagonal covariance structure, it is unsurprising that mPoT parameter learning encounters the same difficulties as encountered with the mcRBM. [Ranzato et al. \(2010\)](#) also advocate direct sampling of $p(x)$ via hybrid Monte Carlo.

10.5.3 Comparing the ssRBM to the mcRBM, mPoT

The mcRBM and the mPoT model differ from the ssRBM in a number of interesting ways. First, while both the mcRBM and mPoT models resort to hybrid

Monte Carlo, the design of the ssRBM admits a simple and efficient Gibbs sampling scheme. It remains to be determined if this difference impacts their relative feasibility though it seems likely that the ssRBM might prove a more flexible framework for further extensions. The later sections of the paper set out to highlight this aspect of the model.

Another difference between these models is how the conditional covariance of the observation is parametrized. The mcRBM and mPoT both model the covariance structure of the observation as $\left(\sum_{j=1}^{N_c} h_j^c C_j C_j^T + \mathbf{I}\right)^{-1}$, using the activation of the hidden units $h_j > 0$ to enforce constraints on the conditional covariance in the direction C_j . In contrast, the ssRBM specifies the conditional covariance of the observations as $\left(\Lambda + \sum_{i=1}^N \Phi_i h_i - \sum_{i=1}^N \alpha_i^{-1} h_i W_i W_i^T\right)^{-1}$, i.e. using the hidden spike activations $h_i = 1$ to pinch the precision matrix along the direction specified by the corresponding weight vector. In fact, the covariance structure of the ssRBM conditional $p(x | h)$ (Eq. 10.5) is very similar to the product of probabilistic principal components analysis (PoPPCA) model (Williams and Agakov, 2002) with components corresponding to the μ -ssRBM weight vectors associated with the active hidden units ($h_i = 1$). In the over-complete setting, sparse activation with the ssRBM parametrization permits significant variance (above the nominal variance given by Λ^{-1}) only in the select directions of the sparsely activated h_i . In the case of the mPoT model or the mcRBM, an over-complete set of constraints on the covariance implies that capturing arbitrary covariance along a particular direction of the input requires removing potentially all constraints with positive projection in that direction. This would suggest that these models are less well suited in the overcomplete setting.

10.6 Exp. I: ssRBM on Natural Images

In this section, we demonstrate the utility of the ssRBM on the CIFAR-10 dataset (Krizhevsky and Hinton, 2009) by classifying images and by sampling from the model. Our experiments explore the roles of μ and Φ and the effects of the Λ and Φ PD constraints. In this first set of experiments we use the CIFAR-10 image classification dataset consisting of 40K training images, 10K validation images, and 10K test images. The images are 32-by-32 pixel RGB images. Each image is

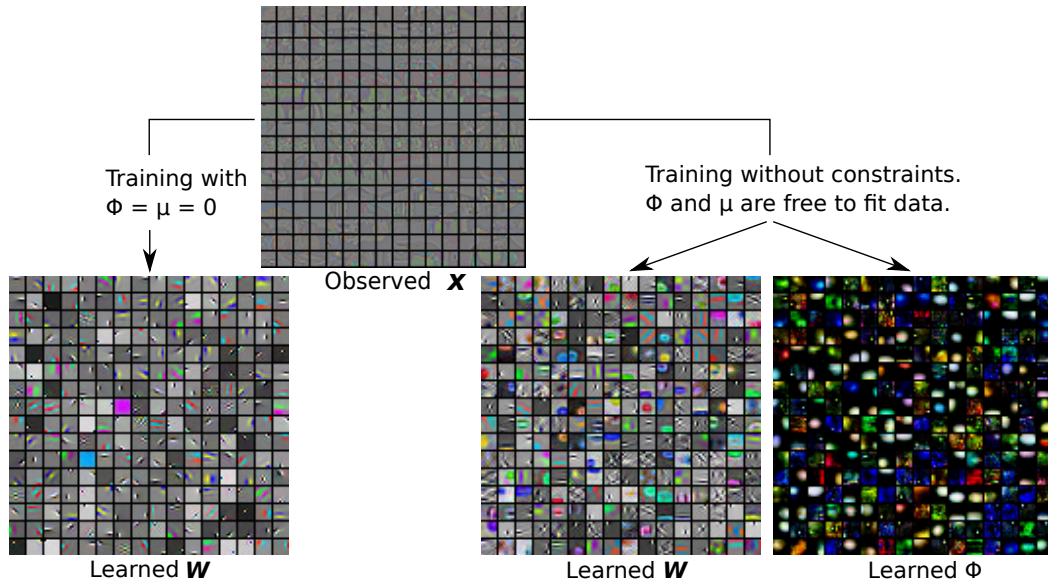


Figure 10.4: ZCA-whitened training data and parameters learnt by the ssRBM reveal filters that neatly separate luminance and color oriented edges. The combination of W and Φ gives individual units more representational flexibility, and yields a wider variety of features.

labelled with one of ten object categories (aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) according to the most prominent object in the image.

10.6.1 Classification

We evaluate the ssRBM as a feature-extraction scheme by plugging it into the classification pipeline developed by Coates et al. (2011). Broadly, the ssRBM is fit to (192-dimensional) 8×8 RGB image patches, and then applied convolutionally to the 32×32 images. The image patches (starting from pixels between 0 and 255) on which the ssRBM was trained were centered, and then normalized by dividing by the square root of their variance plus a noise-cancelling constant ($c = 10$). The normalized patches were whitened by ZCA (Hyvärinen et al., 2001) with a small positive constant (0.1) added to all eigenvalues. The resulting patches (Figure 12.4, top left) are mostly grey with high spatial frequencies amplified, and lower spatial frequencies attenuated. Our models were trained from the 16 non-overlapping 8×8 patches from each of the first 10K training set images in CIFAR-10 (for a total of 160K training examples).

Table 10.1: The performance of ssRBM variants with 256 hidden units in CIFAR-10 image classification ($\pm 95\%$ confidence intervals). “no PD” = without PD constraint.

Model	Accuracy (%)
no PD, μ free, Φ free	73.1 ± 0.9
no PD, μ free, $\Phi = \mathbf{0}$	71.43 ± 0.9
no PD, $\mu = \mathbf{0}$, Φ free	71.19 ± 0.9
no PD, $\mu = \mathbf{0}$, $\Phi = \mathbf{0}$	68.92 ± 0.9
PD by Diag. W (Eq. 10.11)	69.1 ± 0.9
PD by Λ (Eq. 10.9)	68.3 ± 0.9
PD by scal. mat. (Eq. 10.10)	67.1 ± 0.9

Models were trained for one hundred thousand minibatches of 100 patches. On an NVIDIA GTX 285 GPU this training took on the order of 15 minutes for most models. We used SML training (Younes, 1998; Tieleman, 2008). Classification was done with an ℓ_2 -regularized SVM. The SVM was applied to the conditional mean value of latent spike (h) variables, extracted from every 8x8 image patch in the 32x32 CIFAR-10 image. Prior to classification, our conditional h values were spatially pooled into 9 regions, analogous to the 4 quadrants employed in Coates et al. (2011). For a model with N hidden units, the classifier operated on a feature vector of $9N$ elements.

Figure 12.4 shows the filters W and Φ for the trained ssRBM. When $\Phi = \mathbf{0}$, the ssRBM filters display the characteristic Gabor-like edge detectors and look similar to filters learned using a variety of methods such as sparse coding. When Φ is free to be estimated via approximate likelihood maximization, they tend to form localized receptive fields that match those of the corresponding filters W . Comparing the filters W , between the $\Phi = \mathbf{0}$ and Φ free, shows that Φ can have a significant impact on the evolution of the filters. When Φ is free, the filters W display significantly more variety of form.

Table 10.1 shows the results of an ablative analysis on the ssRBM model. For this comparison all variants were trained with a mild sparsity penalty aimed at maintaining 15% activity, and were configured with 256 hidden units. The sparsity penalty is a KL-divergence penalty penalizing average spike variable activity above and below 15% activity. This penalty was done to ensure that all of the hidden units are engaged by the model. The strength of regularization was picked to be just strong enough to have the desired effect, of engaging the full set of hidden units.

Table 10.2: The performance of ssRBM relative to other generative feature-learning models in the literature for CIFAR-10 ($\pm 95\%$ confidence interval).

Model	Accuracy (%)
ssRBM (4096 units)	76.7 ± 0.9
ssRBM (1024 units)	76.2 ± 0.9
ssRBM (512 units)	74.1 ± 0.9
ssRBM (256 units)	73.1 ± 0.9
Deep net, learned RFs (3200)	82.0 ± 0.9
conv. trained DBN	78.9 ± 0.9
mcRBM (225 factors)	68.2 ± 0.9
cRBM (900 factors)	64.7 ± 0.9
cRBM (225 factors)	63.6 ± 0.9
Gaussian RBM	59.7 ± 1.0

We experimented with simplifications to the energy function $\mu = \mathbf{0}$ and $\Phi = \mathbf{0}$. We found that the full model, with both μ and Φ , without any constraint to operate in a strictly PD regime, worked the best. Removing the μ or Φ term from the energy function cost about 1.5% classification accuracy, and removing both cost about 3%. The constraints that the model operate in a strictly PD regime also detracted from classification performance by between 4% and 6%.

Table 10.2 situates the performance of the ssRBM in the literature of results on CIFAR-10. The ssRBM outperforms related energy models GRBM, cRBM, and mcRBM as a feature extractor for classification on CIFAR-10. Although some of the differences in performance may almost certainly be attributable to differences in the pre-processing and classification details [Ranzato and Hinton \(2010\)](#), as we’ve argued in Section 10.5.3, since the mcRBM models the data in terms of constraints rather than directions of variance, they are less well suited to the sparse and overcomplete regime where we see the best performance for the ssRBM. In this task, it appears that the ability to model the conditional mean, exhibited by both the ssRBM and the mcRBM is important factor in improving performance over models such as the cRBM that are not able to model the conditional mean with its hidden units. The “conv. trained DBN” result is the convolutionally trained two-layer Deep Belief Network (DBN) with rectified linear units, reported in [Krizhevsky \(2010\)](#). Recently this method has been improved via a Bayesian optimization scheme ([Snoek et al., 2012b](#)) to achieve 85.0% accuracy. The “Deep net,

learned RFs” result is from [Coates and Ng \(2011b\)](#), a deep neural network is formed by greedy layer-wise unsupervised training, using vector quantization (clustering) to learn the filters and a correlation-based mechanism to learn receptive fields for higher-layer units. Earlier work by [Coates and Ng \(2011a\)](#) has shown that even simple dictionary-learning algorithms such as K-means can yield highly effective feature extractors for CIFAR-10, if only they are used to extract sufficiently many features (thousands). Training the ssRBM with thousands of hidden units yielded less marginal gain than was observed in the case of K-means, but this is possibly because we did not properly optimize the hyper-parameters for this case.

10.6.2 Model Samples

We also trained a version of the ssRBM convolutionally, following the convolutional RBM described in [Krizhevsky \(2010\)](#). Our convolutional implementation of the ssRBM included 1000 fully-connected units to capture global structure, and 64 hidden units for every image position using 9x9 RGB filters. The model was trained on the CIFAR dataset, centered and globally contrast normalized. Filters W and Φ were shared across the image, though independent scalar-parameters μ_i , α_i , and hidden unit bias b_i were allocated for each individual hidden unit.

Figure 10.5 illustrates some samples drawn from the model. The samples are taken from the negative phase at the end of training, with the learning rate annealed to near zero, ($\approx 10^{-6}$). These samples exhibit global coherence, and sharp region boundaries. Qualitatively, these samples compare favorably with samples from similar energy-based models, such as those featured in [Ranzato et al. \(2010\)](#) with samples drawn from the mPoT model. Much like we see in binary and Gaussian RBMs, the negative phase Gibbs sampler for a thoroughly trained ssRBM can mix very slowly, as it does in this convolutionally-trained version. As is the case with these other models, we can turn to established methods, such as tempering to overcome this challenge to sampling ([Salakhutdinov, 2010b](#); [Desjardins et al., 2010b](#); [Cho et al., 2010](#)).

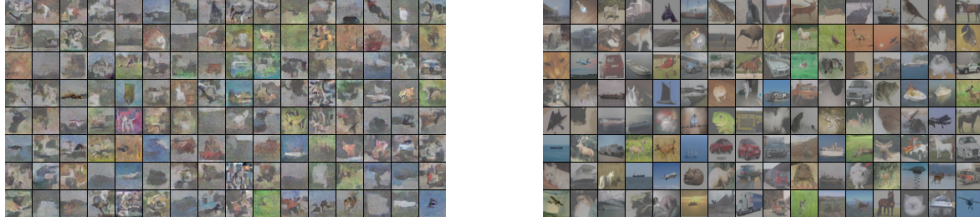


Figure 10.5: Samples of an ssRBM trained on CIFAR-10

(Left) Samples drawn from a convolutionally trained ssRBM (Left), and (Right) closest matching images from the CIFAR-10 training set (L2 distance with contrast normalized training images). Samples have the general appearance of dataset images, and the dissimilarity to corresponding training images indicates that the model has not memorized training points.

10.7 ssRBM for Discrete Data

Unlike related models such as the mcRBM and mPoT, the ssRBM provides a natural framework for capturing conditional covariance in discrete data. Here we will outline an ssRBM model for binary observations $v \in \{0, 1\}^D$. A formulation for multinomial-valued observations ($v \in \{1, \dots, c\}^D$) would be similar.

10.7.1 Model

The spike-and-slab energy function serves perfectly well when v is binary:

$$\begin{aligned}
 E(v, s, h) = & - \sum_{i=1}^N \sum_{j=1}^D v_j W_{ij} s_i h_i - \sum_{j=1}^D \rho_j v_j \\
 & + \frac{1}{2} \sum_{i=1}^N \alpha_i s_i^2 - \sum_{i=1}^N \alpha_i \mu_i s_i h_i - \sum_{i=1}^N b_i h_i
 \end{aligned} \tag{10.13}$$

except that compared with the original energy function (Eq. 10.3) the quadratic in x is redundant, and in Eq. 10.13 it collapses to the linear term $\sum_{j=1}^D \rho_j v_j$ with visible biases ρ_j , for $j \in [1, \dots, D]$. Note that we have set $\Phi_i = 0$ to simplify the parametrization of the model. This is the case for all of the extensions we will consider in the remainder of this article. Even with binary data, the real-valued slab variables s are meaningful. Their variation will capture covariance information

in the binary v , just as in the case of real-valued x .

With regards to the conditionals, the first thing to note is that changing x from a real-valued vector to the binary vector v does not affect the conditional distributions over s or h that arise from conditioning on v . The conditionals $p(s | v, h)$ and $P(h | v)$ are identical to their form in the case of real-valued x and are given by Eqs. 10.7 and 10.5 respectively. The conditional distributions over v are of course affected by its binary nature. It is straightforward to show that, given s , the conditional distribution over the binary visible vector v factorizes:

$$\begin{aligned} p(v | s, h) &= \frac{p(v, s, h)}{p(s, h)} \\ &= \frac{1}{Z'} \prod_{j=1}^D \exp \left\{ \sum_{k=1}^M \sum_{i=1}^N v_j W_{ij} s_i h_i + \sum_{k=1}^M \rho_j v_j \right\} \end{aligned}$$

so that the conditionals over the individual elements of v can be expressed by:

$$p(v_j = 1 | s, h) = \text{sigmoid} \left(\sum_{i=1}^N W_{ij} s_i h_i + \rho_j \right).$$

This conditional is similar to the standard RBM model over binary data, with the addition of real-valued s_i variable.

We can glean some insight into the role of the s_i by considering the conditional over v with s marginalized out.

$$\begin{aligned} p(v | h) &= \int p(v, s | h) ds \\ &\propto \exp \left\{ \sum_{j=1}^D \rho_j v_j + \sum_{i=1}^N \frac{1}{2} \alpha_i \left(\sum_{j=1}^D v_j \alpha_i^{-1} W_{ij} + \mu_i \right)^2 h_i \right\} \end{aligned}$$

While this distribution is less familiar than the Gaussian that emerged as $p(x | h)$ in the real-valued case (Eq. 10.5), marginalizing out s critically renders the elements of the input v conditionally *dependent*. An ssRBM on binary-valued observations is not a reparametrized vanilla RBM. Sampling $p(v | h)$ directly would be difficult, perhaps requiring sequential Gibbs sampling from the elements of v . Unlike in the real-valued setting, a discrete domain for v removes any concern regarding the potential for a non-PSD conditional covariance.

Empirical investigation of the utility of the ssRBM on binary or discrete data is left for future work. However, it is not hard to imagine that in some tasks the real-valued modeling of discrete covariance might be useful. To take document modeling as one example, it may well be desirable to learn a feature representation that captures the covariance of word counts (or the covariance of the probability of word appearance), conditional on an abstract *topic* identified by the distributed representation of the binary vector h .

10.7.2 Exp. II: Binary Visible Data

We evaluate the binary extension of the ssRBM on MNIST, the hand-written character recognition dataset (LeCun et al., 1998). As with our previous classification experiments, we first perform unsupervised learning using SML and in a second phase, use the latent representation of the RBM (conditional mean of the latent variables h) as input to an ℓ_2 -regularized linear SVM.

We compare the ssRBM to a traditional binary-binary RBM, choosing the hyper-parameters through random-search (Bergstra and Bengio, 2012). For each model family, we ran 100 different experiments varying the hyper-parameters as follows: number of hidden units in $[500, 1500]$, initial weights sampled from a zero-mean normal distribution with standard deviation in $\{10^{-3}, 10^{-2}, 10^{-1}, 0.5\}$, sparse activation targets (as described in Hinton (2010)) in $\{0.05, 0.1, 0.2\}$ and giving the sparsity regularization term a weight in $\{0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. Learning rates were chosen to have a linear decreasing schedule, with start and end points randomly sampledⁱ from $\{10^{-2}, 10^{-3}, 10^{-4}\}$ and performed up to 500k updates with a mini-batch of size 64. While the unsupervised training was performed on the entire 60k training set, the SVM was trained on the first 50k labels of the MNIST training set only, using the last 10k to select the hyper-parameters. Test set error is reported after retraining the optimal SVM on the entire training set.

The 20 best resulting models are shown in Figure 10.6. Overall, the lowest classification error obtained by the binary ssRBM is 1.39%, while the RBM achieves 1.67%. In comparison, Local Coordinate Coding achieves 1.64% error (Yu and Zhang, 2010). Interestingly, the filters obtained by the best performing ssRBM are noticeably different than those obtained with an RBM. A random subset of

i. We additionally constrain the learning rate endpoint to be smaller or equal to the starting learning rate.

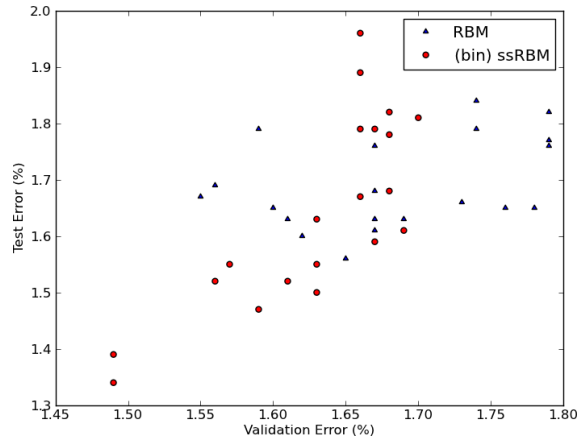


Figure 10.6: MNIST classification results using the binary-input ssRBM. Plot shows test vs. validation error for the 20-best RBM and (binary) ssRBM models.

filters is shown in Figure 10.7. The ssRBM filters appear more diverse, capturing both local (pen-strokes, letter-boxing artifacts) and some more global filters (digit outlines, as well as high-frequency circular gratings).

10.8 ssRBM for Spike-and-Slab Data

One variation of the ssRBM framework that may be of particular interest is how it can be used to model sparse, real-valued data. In particular, we consider a spike-and-slab RBM on spike-and-slab modeled data (S4RBM). That is we consider an observation pair $[x, v]$ where x is a real-valued slab vector and v are binary spike variables. Obviously this setting is interesting from the perspective of stacking the ssRBM to form a spike-and-slab deep belief net (ssDBN). We can train an ssRBM to model the hidden unit activations of the ssRBM in the layer below. However it might also be an excellent way to model sparse real-valued data where elements of the observation vector are either exactly zero or otherwise real-valued. In this setting, the observation is modeled as the element-wise product $x \circ v$. When $v_j = 0$, one may easily impute the missing x (as it is not directly observed).

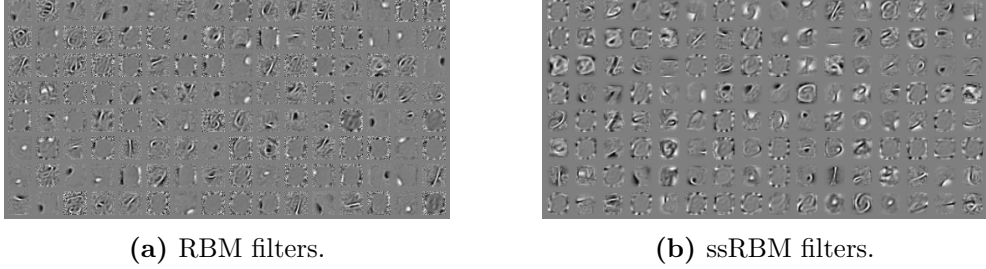


Figure 10.7: Filters learnt by the binary-input ssRBM, trained on MNIST. Random subset of filters drawn from the best performing (Left) RBM and (Right) binary-visible ssRBM models of the MNIST classification experiments of Section 10.7.2. ssRBM filters are noticeably different and appear more diverse, capturing both local and global structure in the input.

10.8.1 Model

The ssRBM with spike-and-slab observations is defined by the energy function:

$$\begin{aligned}
 E(x, v, s, h) = & \\
 & - \sum_{i=1}^N \sum_{j=1}^D x_j v_j W_{ij} s_i h_i - \sum_{j=1}^D \rho_j v_j + \frac{1}{2} \sum_{j=1}^D \lambda_j x_j^2 \\
 & - \sum_{j=1}^D \lambda_j \eta_j x_j v_j + \frac{1}{2} \sum_{j=1}^D \lambda_j \eta_j^2 v_j + \frac{1}{2} \sum_{i=1}^N \alpha_i s_i^2 \\
 & - \sum_{i=1}^N \alpha_i \mu_i s_i h_i + \frac{1}{2} \sum_{i=1}^N \alpha_i \mu_i^2 h_i - \sum_{i=1}^N b_i h_i, \tag{10.14}
 \end{aligned}$$

where the model parameters are defined as before, with ρ_j for $j \in [1, \dots, D]$ acting as a bias on v and λ_j is the precision weight for x_j . Note: while it is technically straightforward to include the terms involving the Φ_i to the S4RBM energy function, we have suppressed them to simplify the model.

As one might expect, the conditionals reveal a symmetry between the conditionals $P(v | s, h)$ and $p(x | v, s, h)$ and the conditionals $P(h | x, v)$ and $p(s | x, v, h)$,

all of which factorize with the conditional distributions over the elements given by:

$$\begin{aligned}
 P(v_j = 1 | s, h) &= \text{sigm} \left(\rho_j + \frac{1}{2} \sigma_j^{-1} \left(\sum_{i=1}^N W_{ij} s_i h_i \right)^2 + \eta_j \sum_{i=1}^N W_{ij} s_i h_i \right) \\
 p(x_j | v, s, h) &= \mathcal{N} \left(\left[\eta_j + \sigma_j^{-1} \sum_{i=1}^N W_{ij} s_i h_i \right] v_j, \sigma^{-1} \right) \\
 P(h_i = 1 | x, v) &= \text{sigm} \left(b_i + \frac{1}{2} \alpha_i^{-1} \left(\sum_{j=1}^D W_{ij} x_j v_j \right)^2 + \mu_i \sum_{j=1}^D W_{ij} x_j v_j \right) \\
 p(s_i | x, v, h) &= \mathcal{N} \left(\left[\mu_i + \alpha_i^{-1} \sum_{j=1}^D W_{ij} x_j v_j \right] h_i, \alpha_i^{-1} \right)
 \end{aligned}$$

In the case of spike-and-slab observations, the block Gibbs sampling scheme becomes a 4-phase algorithm iteratively drawing samples from the conditionals $P(h | x, v)$, $p(s | x, v, h)$, $P(v | s, h)$ and then $p(x | v, s, h)$.

As revealed in the ssRBM experiments in section 10.6, using parameter constraints to ensure that the marginal distribution of the data remains well defined (with a positive definite covariance matrix) resulted in a decrease in classification performance. For this reason, our experiments with the S4RBM used no such constraint. Provided we initialized the model in a stable (PD) regime and used a sufficiently small learning rate ($\leq 1e-3$), we did not experience any difficulty in maintaining stability.

10.8.2 Exp. III: Spike-and-Slab Data

We trained stacked S4RBMs on the output of the best 1-layer model of Section 10.7.2. We employed the typical greedy layer-wise training procedure of Deep Belief Networks and used the concatenation of all binary latent variables as input to a ℓ_2 -regularized linear SVM. MNIST results are shown in Table 10.3.

The 2-layer ssDBN achieves an impressive test error of 1.21%, which is comparable to the original Deep Belief Network results of Hinton et al. (2006), but were obtained without the need for fine-tuning. The rich latent representation learnt by the ssRBM thus seems better suited at capturing discriminative information present in the input.

Table 10.3: Classification error obtained by ssDBNs on MNIST.

Model	Validation (%)	Test (%)
ssRBM (1363 units)	1.49	1.39
ssDBN (1363-1000 units)	1.27	1.21
ssDBN (1363-1000-1000 units)	1.29	1.21

10.9 Subspace Spike-and-Slab RBM

The principle that invariant features can actually *emerge*, using only unsupervised learning, from the organization of features into subspaces was first established in the ASSOM model (Kohonen, 1996). Since then, the same basic strategy has reappeared in a number of different models and learning paradigms (Hyvärinen and Hoyer, 2000; Le et al., 2010; Kavukcuoglu et al., 2009; Ranzato and Hinton, 2010). The strategy is to group filters together by, for example, using a variable (the *pooling* feature) that *gates* the activation for all elements of the group. This gated activation mechanism causes the filters within the group to share a common window on the dataset, which in turn leads to filter groups composed of mutually complementary filters. In the end, the span of the filter vectors defines a subspace which specifies the directions in which the pooling feature is invariant. Somewhat surprisingly, this basic strategy has repeatedly demonstrated that useful invariant features can be learned in a strictly unsupervised fashion, using only the statistical structure inherent in the data.

In this section we explore how the spike-and-slab model can be straightforwardly extended to a subspace feature learning method: the subspace ssRBM. We arrive at the subspace ssRBM by simply generalizing the slab variable associated with hidden unit i to a slab vector of dimension L : $s_i \in \mathbb{R}^L$ and associating an independent weight vector W_{il} with each element of the slab vector. What this extension implies is that each binary spike variable h_i is associated with a set of L slab variables and their associated weight vectors. Modifying the original ssRBM energy function to incorporate this extension is a trivial matter of converting the relevant scalar operations to vector and matrix operations (not shown). All conditionals are equivalent except the conditional $P(h_i = 1 | x)$ which must incorporate all interactions between the observations and the weight vectors associated with h_i :

Table 10.4: MNIST classification error of a subspace (binary input) ssRBM with (a) $N = 500$ hidden units and a pooling size $L \in \{1, 3, 5\}$ and (b) $L = 1$ with $N \in \{500, 1.5K, 2.5K\}$ as a function of the number of supervised training examples.

Number of Labels	N=500	N=500		L=1	
	L=1	L=3	L=5	N=1500	N=2500
10	18.21	<i>14.51</i>	13.44	19.17	22.00
100	5.82	<i>5.22</i>	5.03	6.32	6.70
1000	2.94	2.70	<i>2.69</i>	2.64	2.99

$$P(h_i = 1 | x) = \text{sigm} \left(\frac{1}{2} \sum_{l=1}^L \alpha_{il}^{-1} (x^T W_{\cdot,il})^2 + \sum_{l=1}^L v^T W_{\cdot,il} \mu_{il} - \frac{1}{2} v^T \Phi_i v + b_i \right) \quad (10.15)$$

In contrast to the standard ssRBM or the S4RBM, when applied to real-valued data, subspace-ssRBM is more susceptible slipping outside the parameter regime in which the marginal distribution over the visible units is assured to be positive definite. Rather than enforce the parameter constraints we explored in Sec. 10.3, we opted for a simple box constraint on the visible variables that enclosed the training data. For each data points all dimensions that fell outside the interval were clipped to the interval boundary. In order to enforce the box constraint for the negative phase samples (see Sec. 10.4), for each dimension of x , we use a truncated normal (\mathcal{TN}) distribution with truncation interval $[x_L, x_H]$:

$$p(x_j | s, h) = \mathcal{TN} \left(C_{x_j|s,h} \sum_{i=1}^N \left(\sum_{l=1}^L W_{il} s_{il} \right) h_i, C_{x_j|s,h}, x_L, x_H \right), \quad (10.16)$$

$$\text{where } C_{x_j|s,h} = \left[\left(\Lambda + \sum_{i=1}^N \Phi_i h_i \right)^{-1} \right]_{jj},$$

that is the j th element along the diagonal of the covariance matrix $C_{x|s,h}$. This distribution defines a Gaussian distribution for $x_L < x_j < x_H$ and $P((x_j \geq x_H) \cup (x_j \leq x_L)) = 0$. In our experiments this constraint on the domain of the visible units worked well.

10.9.1 Exp. IV: Subspace ssRBM

We now attempt to quantify the effect of pooling on classification performance. In particular, we try to determine whether for a fixed network capacity (i.e. number of filters), it is better to use pooling ($L > 1$) or simply increase the number of hidden units to $L \times N$.

MNIST: We trained various subspace (binary input-valued) ssRBM models on MNIST, measuring classification error as a function of both pooling size and number of training examples used by the SVM. Models were chosen to have either $N = 500$ with $L \in \{1, 3, 5\}$ or $L = 1$ with $N \in \{500, 1.5\text{K}, 2.5\text{K}\}$ hidden units. The number of training labels was restricted to $\{10, 100, 1\text{K}\}$ (per class). Hyperparameters were otherwise chosen from a range similar to Section 10.7.2, with the exception of learning rates which were held constant in $\{10^{-1}, 10^{-2}, 10^{-3}\}$.

The results are presented in Table 10.4. We can see that for models having equivalent capacity, pooling is always beneficial in the low-labeled data regime (shown in *italic*). When limiting ourselves to 10 training labels, the best pooling model ($N = 500, L = 5$) achieves 13.44% classification error, a reduction of 29.9% compared to the 19.17% error achieved by the best un-pooled configuration ($N = 1500, L = 1$). Pooling remains beneficial when using 100 labels, decreasing the error of the best non-pooled model from 6.32% to 5.03% with pooling, a decrease of about 20.4%. When using 1k labels, the benefits of pooling seem to be offset by the benefits of using a larger output layer: a model with $N = 1500$ hidden units and no-pooling achieved 2.64% error, compared to 2.69% with pooling.

These results should not come as a surprise. It is a fairly well known result that large over-complete representations are best when using simple linear classifiers (Coates et al., 2011). In the low-data regime however, training a large output layer becomes problematic due to overfitting. By allowing each hidden unit to be invariant to a larger subspace of the input, pooling can yield a richer representation, while restricting the dimensionality of the output.

A random subset of filters from a competitive pooling model with $L = 3$ is shown in Figure 10.8(left). We can see that filters belonging to the same pool (consecutive groups of 3 filters, e.g. outlined in red or blue) tend to learn similar pen-strokes, often with offsets in position, curvature or phase.

Table 10.5: CIFAR-10 classification error of a fully-connected subspace ssRBM with (a) $N = 500$, $L \in \{1, 3, 5\}$ and (b) $L = 1$ with $N \in \{500, 1.5K, 2.5K\}$ as a function of the number of supervised training examples.

Number of Labels	N=500	N=500		L=1	
	L=1	L=3	L=5	N=1500	N=2500
500	57.23	<i>53.16</i>	51.10	58.09	57.37
1000	54.29	<i>50.42</i>	48.26	53.56	52.88
2000	52.34	<i>48.95</i>	46.57	49.64	47.66

CIFAR-10: We now perform similar experiments on the CIFAR-10 dataset. Prior to training, the images were preprocessed by first performing global contrast normalization, followed by a ZCA whitening transform. We again compare models with $N = 500$ and $L \in \{1, 3, 5\}$ to models with no-pooling and $N \in \{500, 1.5K, 2.5K\}$ and vary the number of training labels in $\{500, 1K, 2K\}$ ⁱ. The results are shown in Table 10.5. With $(N = 500, L = 5)$, we achieve 51.1% error, an 11% reduction in error compared to the 57.37% achieved with $(N = 2500, L = 1)$. As with MNIST, this relative boost in performance drops as we increase the number of training labels at our disposal: an 8.74% reduction in error with 1k labels, and 2.3% using 2k labels. Note that the goal of this experiment was to perform a comparative analysis. We did not employ convolutional architectures, depth nor large models of 10k units. This explains the gap with other published results (Yu and Zhang, 2010; Krizhevsky, 2010).

Figure 10.8(right) shows a random subset of filters, obtained with $L = 3$. We can clearly see a topological structure, which emerges from the pooling. Filters belonging to the same pool are similar but span rich subspaces through subtle shifts in phase, curvature and orientation of the Gabor-like filters. Interestingly, we also see filters with two (sometimes overlapping) edge detectors, which was not observed with $L = 1$ (not shown).

i. Using fewer labels yielded a significantly worse performance, regardless of model capacity and pooling size.

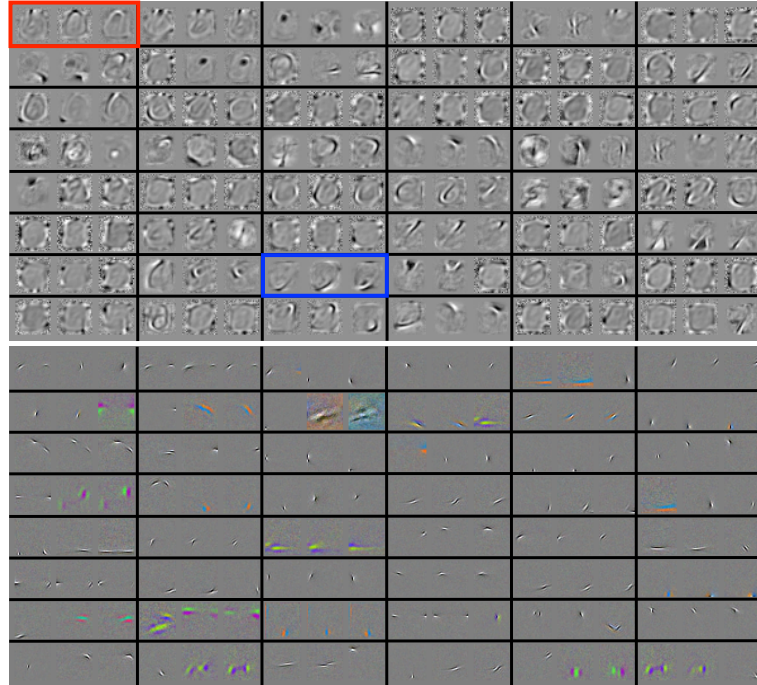


Figure 10.8: Filters learnt by a subspace ssRBM on MNIST and CIFAR-10. Random subset of filters drawn from competitive pooled (Left) binary-input ssRBM with $L = 3$, trained on MNIST, and (Right) ssRBM with $L = 3$ trained on global contrast-normalized and whitened CIFAR-10 images. Filters belonging to the same pool are arranged in contiguous blocks of L filters, with two examples shown with a blue and red outline.

10.10 Conclusion

The spike-and-slab RBM offers a powerful framework for modeling real-valued input data, in particular, we have explored its suitability for natural images. Unlike the Gaussian RBM which is limited to modeling diagonal conditional covariances, the slab variables affords the ssRBM rich modeling capacity. Contrary to the mcRBM and mPoT models however, this does not come at the expense of the simple RBM conditional dependency structure: the ssRBM allows for an efficient blocks Gibbs sampling algorithm, by conditioning on the slab variables when sampling from the visible units. One potential drawback of the ssRBM parametrization however, is that its energy function does not guarantee that the conditional covariance on units x be positive definite. This issue can be side-stepped however

by imposing additional constraints on Λ and Φ . On the competitive CIFAR-10 dataset, the ssRBM was shown to outperform many of its competing methods. Since it does not rely on HMC for generating visible samples, the ssRBM can also be naturally extended to modeling covariances in binary data and even sparse data.

In this paper we have also proposed two novel variants of the spike-and-slab latent variable framework that target (1) discrete (especially binary) data and (2) spike-and-slab data. We use the first of these to show how the spike-and-slab framework can yield superior performance as a feature extractor compared to the standard RBM for classification on the MNIST dataset. We then showed how we can use the model variant with spike-and-slab data to support the stacking of spike-and-slab RBMs in a spike-and-slab deep belief network (ssDBN). This model was shown to provide fairly competitive results on MNIST classification without the use of fine-tuning the model parameters for the discriminative task. We expect that fine-tuning the ssDBN would result in further improvements in performance. Finally, by extending slab variables to be vector-valued, one can learn features (spikes) which are invariant to a subspace spanned by the set of filters associated with each slab. We demonstrate the utility of this modeling framework on the MNIST and CIFAR-10 dataset in the low-data limit. We should note that in the low-labeled-training-data regime, that organizing the model capacity in the form of these pools leads to a significant boost in classification performance.

Acknowledgements

We acknowledge NSERC, FQRNT, CIFAR, RQCHP and Compute Canada for their financial and computational support; and Chris Williams for pointing out the connection to the PoPPCA model. We also thank the anonymous reviewers for their valuable suggestions for improving the paper.

11

Prologue to Fifth Article

11.1 Article Details

Disentangling Factors of Variation via Generative Entangling. Guillaume Desjardins, Aaron Courville and Yoshua Bengio. ArXiv 1210.5474v1.

Personal Contribution. This paper is again the result of a close and ongoing collaboration. Yoshua Bengio and Aaron Courville have a long standing interest in disentangling factors of variation, and the original idea to extend the ssRBM to the bilinear setting is Aaron Courville's. The particular form of the model was however developed jointly, after gaining new insights from the subspace ssRBM. All of the coding and experimentation was performed by myself. The motivation and model development sections of the paper were written by Aaron and Yoshua, while I wrote the experimental and discussion sections of the paper. I would also like to acknowledge and thank Ian Goodfellow, for developing an earlier version of the toy dataset of Section 12.5.1.

11.2 Context

Chapter 10 has focused on learning rich invariant representations in the ssRBM framework. Invariance has been a key driver in representation learning research. It is typically obtained via spatial pooling; bag of (visual) words or spatial pyramid approaches from computer vision; priors such as sparsity or slowness which have been found to induce invariance; or learning richer invariant subspaces directly, through the non-linear combination of small filter banks as seen in Chapter 10ⁱ. Invariance however is a lossy process: while translation invariance might be a

i. See Chapter 12 for references and a more thorough treatment of the material.

desirable property for object detection, it is completely at-odds with the related task of object localization. This has led to a renewed effort to learn representations which preserve as much information as possible about the input, while *disentangling the factors of variation* (Bengio et al., 2013).

This is the topic of Chapter 12. It follows in the footsteps of bilinear sparse coding (Tenenbaum and Freeman, 2000; Grimes and Rao, 2005; Olshausen et al., 2007), transforming auto-encoders (Hinton et al., 2011), implicit mixtures of RBMs (Nair and Hinton, 2009), general work on higher-order RBMs (Memisevic and Hinton, 2010) and previous strictly supervised approaches to learning disentangled representations (Osadchy et al., 2007).

11.3 Contributions

The main contribution of this paper is a framework for learning to disentangle factors of variation in a completely unsupervised manner: factors are discovered automatically via an unsupervised learning procedure which is agnostic to the underlying factors present in the input. This is contrast to the work of Tenenbaum and Freeman (2000); Grimes and Rao (2005) presented in Section 2.6.4: their work relied on a structured training procedure in which subsets of units are clamped while the input undergoes a known transformation. While transforming auto-encoders (Hinton et al., 2011) offer a powerful hierarchical framework for disentangling pose parameters, they similarly require advance knowledge of the transformations.

Our approach is also general in that the full rank nature of the weight tensor allows it to capture arbitrary transformations of the data. Computational efficiency is maintained via sparsity. In contrast, the approach of Olshausen et al. (2007) (also completely unsupervised) was specifically crafted to separate amplitude from phase.

Another significant contribution is the connection between disentangling and multi-way pooling. While invariance can be obtained by pooling over a subspace, disentangling can be achieved by pooling over multiple complimentary views of a given subspace.

Finally, the following paper represents to our knowledge the first successful attempt at training third-order BMs, where two of the factors are latent.

11.4 Recent Developments

It recently came to our attention that we failed to cite the work of [Culpepper et al. \(2011\)](#), which predates publication of our work. Of all papers published on the topic, it is the most similar to ours. Theirs is essentially a directed model equivalent of a bilatent, factored third-order BM (see Section 2.6.3). This is particularly interesting as our inability to train a factored bilatent ssRBM is what eventually led to us using a sparse weight tensor instead. Another core, and perhaps crucial difference between both methods, is in the use of different priors for each group of latent variable (Gaussian vs. sparsity inducing), which has the benefit of breaking the symmetry between latents g and h . Finally, the use of a second order optimization (L-BFGS) may explain their ability to train such a factored bi-latent model. This may warrant revisiting the factored bilatent ssRBM, but using the MFNG training algorithm of Chapter 8 instead of (variational) SML.

[Tang et al. \(2013\)](#) proposed disentangling factors of variation through a multi-linear extension of Factor Analysis (FA), dubbed Tensor Analyzers (TA). TAs model the input x as a multi-linear interaction between a K -th order factor loading tensor and K Gaussian latent variables z_1, \dots, z_k . The model's training procedure is similar to FA and relies on the EM algorithm (see Section 1.4.1): due to the multi-linear interactions, inference is no longer closed form however and must be approximated via Gibbs sampling. This is similar to our bilinear ssRBM, which relies on either Gibbs or a variational approximation to the posterior. TAs are shown to outperform FA in modeling complex distributions. On a facial recognition task, the ability to separate identity information from lighting conditions also provides the model with impressive one-shot learning performance on held-out data. This echoes our conclusions from Chapter 10, where learning the right invariance drastically improved performance in the low-labeled data regime.

The TA is closely related to our bilinear ssRBM formulation. At a high-level, it can be considered as the directed model equivalent of our bilinear model. A core difference is in the prior distribution: TAs use a Gaussian prior, while our bilinear model relies on a spike & slab prior to perform non-linear density estimation. The block-sparsity structure of our model also implements a mixture of bilinear ssRBMs, where each block can model local bilinear interactions, much like the proposed Mixture of TAs. In terms of drawbacks, inference in TAs appears rather costly as

a single evaluation of $p(z_1 | x, z_2)$ requires a matrix-inversion which is cubic in the dimensionality of z_1 . In contrast, sampling from the same conditional is linear in the bilinear ssRBM, but learning as a whole suffers from having to estimate the gradient through the partition function.

Rifai et al. (2012) proposed a novel solution for separating K additive factors. The Contractive Discriminative Analysis (CDA) loss linearly combines the loss functions of K independent CAEs (trained jointly to reconstruct a given input), with up to K logistic regressors. When labeled data is available for the k -th factor, a linear classifier is trained to predict the associated factor label from the latent representation of the k -th CAE. To promote learning complimentary views on the data, additional terms in the loss function penalize colinear directions of variation between the encoding function of the CAEs. At the time of publication, this led to state-of-the-art results on the Toronto Face Dataset (TFD) (Susskind et al., 2010). We hypothesize that the shallow encoder put forth in Rifai et al. (2012) would prevent the CDA from disentangling factors having multiplicative interactions. While the CDA framework does allow for a richer parametrization of encoding and decoding functions, its potential to disentangle general multiplicative factors remains to be seen.

Similarly, the point-wise mixture RBM (pmRBM) (Sohn et al., 2013) exploits a bilatent third-order BM to dynamically assign each input pixel to one of K groups. In this manner, the pmRBM can learn to distinguish foreground (class relevant) from background (class irrelevant) features. While the motivation is entirely different, this model can be thought as a special case of our bilinear model: the groupings can be implemented via a particular sparsity pattern of the weight tensor, while the latent variable g found in the bilinear model is modified to have a multinomial distribution.

Disentangling Factors of Variation via Generative Entangling

HERE we propose a novel model family with the objective of learning to disentangle the factors of variation in data. Our approach is based on the spike-and-slab restricted Boltzmann machine which we generalize to include higher-order interactions among multiple latent variables. Seen from a generative perspective, the multiplicative interactions emulates the entangling of factors of variation. Inference in the model can be seen as disentangling these generative factors. Unlike previous attempts at disentangling latent factors, the proposed model is trained using no supervised information regarding the latent factors. We apply our model to the task of facial expression classification.

12.1 Introduction

In many machine learning tasks, data originates from a generative process involving complex interaction of multiple factors. Alone each factor accounts for a source of variability in the data. Together their interaction gives rise to the rich structure characteristic of many of the most challenging domains of application. Consider, for example, the task of facial expression recognition. Two images of different individuals with the same facial expression may result in images that are well separated in pixel space. On the other hand, two images of the same individuals showing different expressions may well be positioned very close together in pixel space. In this simplified scenario, there are two factors at play: (1) the identity of the individual, and (2) the facial expression. One of these factors, the identity, is irrelevant to the task of facial expression recognition and yet of the two factors it could well dominate the representation of the image in pixel space. As a result, pixel space-based facial expression recognition systems seem likely to suffer poor performance due to the variation in appearance of individual faces.

Importantly, these interacting factors frequently do not combine as simple superpositions that can be easily separated by choosing an appropriate affine projection of the data. Rather, these factors often appear tightly *entangled* in the raw data. Our challenge is to construct representations of the data that cope with the reality of entangled factors of variation and provide features that may be appropriate to a wide variety of possible tasks. In the context of our face data example, a representation capable of disentangling identity and expression would be an effective representation for either the facial recognition or facial expression classification.

In an effort to cope with these factors of variation, there has been a broad-based movement in machine learning and in application domains such as computer vision toward hand-engineering feature sets that are *invariant* to common sources of variation in data. This is the motivation behind both the inclusion of feature pooling stages in the convolutional network architecture (LeCun et al., 1989) and the recent trend toward representations based on large scale pooling of low-level features (Wang et al., 2009; Coates et al., 2011). These approaches all stem from the powerful idea that invariant features of the data can be induced through the pooling together of a set of simple filter responses. Potentially even more powerful is the notion that one can actually *learn* which filters to be pooled together from purely unsupervised data, and thereby extract directions of variance over which the pooling features become invariant (Kohonen et al., 1979; Kohonen, 1996; Hyvärinen and Hoyer, 2000; Le et al., 2010; Kavukcuoglu et al., 2009; Ranzato and Hinton, 2010; Courville et al., 2011a). However, in situations where there are multiple relevant but entangled factors of variation that give rise to the data, we require a means of feature extraction that *disentangles* these factors in the data rather than simply learn to represent some of these factors at the expense of those that are lost in the filter pooling operation.

Here we propose a novel model family with the objective of *learning to disentangle* the factors of variation evident in the data. Our approach is based on the spike-and-slab restricted Boltzmann machine (ssRBM) (Courville et al., 2011b) which has recently been shown to be a promising model of natural image data. We generalize the ssRBM to include higher-order interactions among multiple binary latent variables. Seen from a generative perspective, the multiplicative interactions of the binary latent variables emulates the entangling of the factors that give rise

to the data. Conversely, inference in the model can be seen as an attempt to assign credit to the various interacting factors for their combined account of the data – in effect, to disentangle the generative factors. Our approach relies only on unsupervised approximate maximum likelihood learning of the model parameters, and as such we do not require the use of any label information in defining the factors to be disentangled. We believe this to be a research direction of critical importance, as it is almost never the case that label information exists for all factors responsible for variations in the data distribution.

12.2 Learning Invariant Features Versus Learning to Disentangle Features

The principle that invariant features can actually *emerge*, using only unsupervised learning, from the organization of features into subspaces was first established in the ASSOM model (Kohonen, 1996). Since then, the same basic strategy has reappeared in a number of different models and learning paradigms, including topological independent component analysis (Hyvärinen and Hoyer, 2000; Le et al., 2010), invariant predictive sparse decomposition (IPSD) (Kavukcuoglu et al., 2009), as well as in Boltzmann machine-based approaches (Ranzato and Hinton, 2010; Courville et al., 2011a). In each case, the basic strategy is to group filters together by, for example, using a variable (the pooling feature) that gates the activation for all elements of the group. This gated activation mechanism causes the filters within the group to share a common window on the dataset, which in turn leads to filter groups composed of mutually complementary filters. In the end, the span of the filter vectors defines a subspace which specifies the directions in which the pooling feature is invariant. Somewhat surprisingly, this basic strategy has repeatedly demonstrated that useful invariant features can be learned in a strictly unsupervised fashion, using only the statistical structure inherent in the data. While remarkable, one important problem with using this learning strategy is that the invariant representation formed by the pooling features offers a somewhat incomplete view on the data as the detailed representation of the lower-level features is abstracted away in the pooling procedure. While we would like higher

level features to be more abstract and exhibit greater invariance, we have little control over what information is lost through feature subspace pooling.

Invariant features, by definition, have reduced sensitivity in the direction of invariance. This is the goal of building invariant features and fully desirable if the directions of invariance all reflect sources of variance in the data that are uninformative to the task at hand. However, it is often the case that the goal of feature extraction is the *disentangling* or separation of many distinct but informative factors in the data. In this situation, the methods of generating invariant features – namely, the feature subspace method – may be inadequate. Returning to our facial expression classification example from the introduction, consider a pooling feature made invariant to the expression of a subject by forming a subspace of low-level filters that represent the subject with various facial expressions (forming a basis for the subspace). If this is the only pooling feature that is associated with the appearance of this subject, then the facial expression information is lost to the model representation formed by the set of pooling features. As illustrated in our hypothetical facial expression classification task, this loss of information becomes a problem when the information that is lost is necessary to successfully complete the task at hand.

Obviously, what we really would like is for a particular feature set to be invariant to the irrelevant features and disentangle the relevant features. Unfortunately, it is often difficult to determine *a priori* which set of features will ultimately be relevant to the task at hand. Further, as is often the case in the context of deep learning methods (Collobert and Weston, 2008), the feature set being trained may be destined to be used in multiple tasks that may have distinct subsets of relevant features. Considerations such as these lead us to the conclusion that the most robust approach to feature learning is to disentangle as many factors as possible, discarding as little information about the data as is practical. This is the motivation behind our proposed higher-order spike-and-slab Boltzmann machine.

$$\begin{aligned}
E(v, s, f, g, h) = & \frac{1}{2}v^T \Lambda v - \sum_k e_k f_k + \sum_{i,k} c_{ik} g_{ik} + \sum_{j,k} d_{jk} h_{jk} + \frac{1}{2} \sum_{i,j,k} \alpha_{ijk} s_{ijk}^2 \\
& + \sum_{i,j,k} \left(-v^T W_{\cdot,ijk} s_{ijk} - \alpha_{ijk} \mu_{ijk} s_{ijk} + \frac{1}{2} \alpha_{ijk} \mu_{ijk}^2 \right) g_{ik} h_{jk} f_k,
\end{aligned}$$

Figure 12.1: Higher-Order ssRBM energy function

Energy function of our higher-order spike & slab RBM (ssRBM), used to disentangle (multiplicative) factors of variation in the data. Two groups of latent spike variables, g and h , interact to explain the data v , through the weight tensor W . While the ssRBM instantiates a slab variable s_j for each hidden unit h_j , our higher-order model employs a slab s_{ij} for each pair of spike variables (g_i, h_j) . μ_{ij} and α_{ij} are respectively the mean and precision parameters of s_{ij} . An additional set of spike variables f are used to gate groups of latent variables h , g and serve to promote group sparsity. Most parameters are thus indexed by an extra subscript k . Finally, e , c and d are standard bias terms for variables f , g and h , while Λ is a diagonal precision matrix on the visible vector.

12.3 Higher-order Spike-and-Slab Boltzmann Machines

In this section, we introduce a model which makes some progress toward the ambitious goal of disentangling factors of variation. The model is based on the Boltzmann machine, an undirected graphical model. In particular we build on the spike-and-slab restricted Boltzmann Machine (ssRBM) (Courville et al., 2011a), a model family that has previously shown promise as a means of learning invariant features via subspace pooling. The original ssRBM model possessed a limited form of higher-order interaction of two latent random variables: the spike and the slab. Our extension adds higher-order interactions between four distinct latent random variables. These include one set of slab variables and three interacting binary spike variables. Unlike the ssRBM, the interactions between the latent variables violate the conditional independence constraint of the restricted Boltzmann machine and therefore does not belong to this class of models. As a consequence, exact inference in the model is not tractable and we resort to a mean-field approximation.

Our strategy in promoting this model is that we intend to disentangle factors of variation via inference (recovering the posterior distribution over our latent vari-

ables) in a generative model. In the context of generative models, inference can roughly be thought of as running the generative process in reverse. Thus if we wish our inference process to disentangle factors of variation, our generative process should describe a means of factor entangling. The generative model we propose here represents one possible means of factor entangling.

Let $v \in \mathbb{R}^D$ be the random visible vector that represents our observations with its mean zeroed. We build a latent representation of this data with binary latent variables $f \in \{0, 1\}^K$, $g \in \{0, 1\}^{M \times K}$ and $h \in \{0, 1\}^{N \times K}$. In the spike-and-slab context, we can think of f , g and h as a factored representation of the “spike” variables. We also include a set of real valued “slab” variables $s \in \mathbb{R}^{M \times N \times K}$, with element s_{ijk} associated with hidden units f_k , g_{ik} and h_{jk} . The interaction between these variables is defined through the energy function of Fig. 12.1.

The parameters are defined as follows. $W \in \mathbb{R}^{D \times M \times N \times K}$ is a weight 4-tensor connecting visible units to the interacting latent variables, these can be interpreted as forming a basis in image space; $\mu \in \mathbb{R}^{M \times N \times K}$ and $\alpha \in \mathbb{R}^{M \times N \times K}$ are tensors describing the mean and precision of each s_{ijk} ; $\Lambda \in \mathbb{R}^{D \times D}$ is a diagonal precision matrix on the visible vector; and finally $c \in \mathbb{R}^{M \times K}$, $d \in \mathbb{R}^{N \times K}$ and $e \in \mathbb{R}^K$ are biases on the matrices g , h and vector f respectively. The energy function fully specifies the joint probability distribution over the variables v , s , f , g and h : $p(v, s, f, g, h) = \frac{1}{Z} \exp \{-E(v, s, f, g, h)\}$ where Z is the partition function which ensures that the joint distribution is normalized.

As specified above, the energy function is similar to the ssRBM energy function (Courville et al., 2011a,b), but includes a factored representation of the standard ssRBM spike variable. Yet, clearly the properties of the model are highly dependent on the topology of the interactions between the real-valued slab variables s_{ijk} , and three binary spike variables f_k , g_{ik} and h_{jk} . We adopt a strategy that permits local interactions within small groups of f , g and h in a block-like organizational pattern as specified in Fig. 12.2. The local block structure allows the model to work incrementally towards disentangling the features by focusing on manageable subparts of the problem.

Similar to the standard spike-and-slab restricted Boltzmann machine (Courville et al., 2011a,b), the energy function in Eq. 12.1 gives rise to a Gaussian conditional

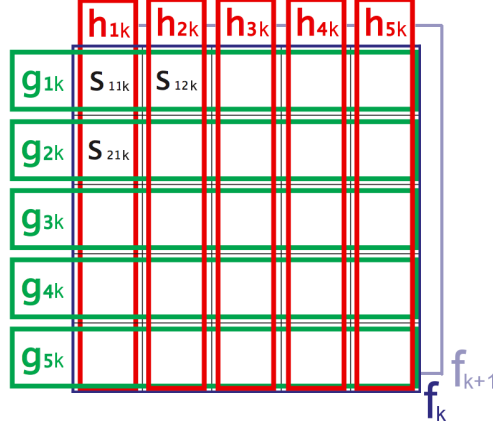


Figure 12.2: Block-sparse connectivity pattern with dense interactions between g and h within each block (only shown for k -th block). Each block is gated by a separate f_k variable.

over the visible variables:

$$p(v \mid s, f, g, h) = \mathcal{N} \left(\sum_{i,j,k} \Lambda^{-1} W_{ijk} s_{ijk} g_{ik} h_{jk} f_k, \Lambda^{-1} \right)$$

Here we have a four-way multiplicative interaction in the latent variables s , f , g and h . The real-valued slab variable s_{ijk} acts to scale the contribution of the weight vector W_{ijk} . As a consequence, after marginalizing out s , the factors f , g and h can also be seen as contributing both to the conditional mean and conditional variance of $p(v \mid f, g, h)$:

$$p(v \mid f, g, h) = \mathcal{N} \left(\sum_{i,j,k} \Lambda^{-1} W_{ijk} \mu_{ijk} g_{ik} h_{jk} f_k, C_{v|f,g,h} \right)$$

$$C_{v|f,g,h} = \left(\Lambda - \sum_{i,j,k} W_{ijk} W_{ijk}^T \alpha_{ijk}^{-1} g_{ik} h_{jk} f_k \right)^{-1}$$

This is an important property of the spike-and-slab framework that is also shared by other latent variable models of real-valued data such as the mean-covariance restricted Boltzmann machine (mCRBM) (Ranzato and Hinton, 2010) and the mean Product of T-distributions model (mPoT) (Ranzato et al., 2010).

From a *generative* perspective, the model can be thought of as consisting of a set of K factor blocks whose activity is gated by the f variables. Within each block, the variables $g_{\cdot k}$ and $h_{\cdot k}$ can be thought of as local latent factors whose interaction gives rise to the active block's contribution to the visible vector. Crucially, the multiplicative interaction between the $g_{\cdot k}$ and $h_{\cdot k}$ for a given block k is mediated by the weight tensor $W_{\cdot, \cdot, k}$ and the corresponding slab variables $s_{\cdot, \cdot, k}$. Contrary to more standard probabilistic factor models whose factors simply sum to give rise to the visible vector, the individual contributions of the elements of $g_{\cdot k}$ and $h_{\cdot k}$ are not easily isolated from one another. We can think of the generative process as *entangling* the local block factor activations.

From an *encoding* perspective, we are interested in using the posterior distribution over the latent variables as a representation or encoding of the data. Unlike in RBMs, in the case of the proposed model where we have higher-order interactions over the latent variables, the posterior over the latent variables does not factorize cleanly. By marginalizing over the slab variables s , we can recover a set of conditionals describing how the binary latent variables f , g and h interact. The conditional $P(f | v, g, h)$ is given below.

$$P(f_k = 1 | v, g, h) = \text{sigm} \left(c_{ik} + \sum_{i,j} v^T W_{\cdot ij k} \mu_{ijk} g_{ik} h_{jk} + \frac{1}{2} \sum_{i,j} \alpha_{ijk}^{-1} [v^T W_{\cdot ij k}]^2 g_{ik} h_{jk} \right)$$

It illustrates that with the factor configuration given in Fig. 12.2, the factors f_k are activated (assume value 1) through the sum-pooled response of all the weight vectors $W_{\cdot ij k}$ ($\forall 1 \leq i \leq M$ and $1 \leq j \leq N$) differentially gated by the values of g_{ik} and h_{jk} , whose conditionals are respectively given by:

$$P(g_{ik} = 1 | v, f, h) = \text{sigm} \left(c_{ik} + \sum_j v^T W_{\cdot ij k} \mu_{ijk} h_{jk} f_k + \frac{1}{2} \sum_j \alpha_{ijk}^{-1} [v^T W_{\cdot ij k}]^2 h_{jk} f_k \right)$$

$$P(h_{jk} = 1 | v, f, g) = \text{sigm} \left(d_{jk} + \sum_i v^T W_{\cdot ij k} \mu_{ijk} g_{ik} f_k + \frac{1}{2} \sum_i \alpha_{ijk}^{-1} [v^T W_{\cdot ij k}]^2 g_{ik} f_k \right)$$

For completeness, we also include the Gaussian conditional distribution over the slab variables s

$$p(s_{ijk} | v, f, g, h) = \mathcal{N} \left([\alpha_{ijk}^{-1} v^T W_{\cdot ij k} + \mu_{ijk}] f_k g_{ik} h_{jk}, \alpha_{ijk}^{-1} \right)$$

From an encoding perspective, the gating pattern on the g and h variables, evident from Fig. 12.2 and from the conditionals distributions, defines a form of local bilinear interaction (Tenenbaum and Freeman, 2000). We can interpret the values of g_{ik} and h_{jk} within block k acting as basis indicators, in dimensions i and j , for the linear subspace in the visible space defined by $W_{.ijk}s_{ijk}$.

From this perspective, we can think of $[g_{.k}, h_{.K}]$ as defining a block-local binary coordination encoding of the data. Consider the case illustrated by Fig. 12.2, where we have $M = 5$, $N = 5$ and the number of blocks (K) is 4. For each block, we have $M \times N = 25$ filters which we encode using $M + N = 10$ binary latent variables, where each g_{ik} (alternately h_{jk}) effectively pools over the subspace characterized by the variables h_{jk} , $1 \leq j \leq N$ (alternately g_{ik} , $1 \leq i \leq M$) through their relative interaction with $W_{.ijk}s_{ijk}$. As a concrete example, imagine that the structure of the weight tensor was such that, along the dimension indexed by i , the weight vectors $W_{.ijk}$ form oriented Gabor-like edge detectors of different orientations. Yet along the dimension indexed by j , the weight vectors $W_{.ijk}$ form oriented Gabor-like edge detectors of different colors. In this hypothetical example, g_{ik} encodes orientation information while being invariance to the color of the edge, while h_{jk} encodes color information while being invariant to orientation. Hence we could say that we have *disentangled* the latent factors.

12.3.1 Higher-order Interactions as a Multi-Way Pooling Strategy

As alluded to above, one interpretation of the role of g and h is as distinct and complementary sum-pooled feature sets. Returning to Fig. 12.2, we can see that, for each block, the g_{ik} pool across the columns of the k th block, along the i th row, while the $h_{.k}$ pool across rows, along the j th column. The f variables are also interpretable as pooling across all elements of the block. One way to interpret the complementary pooling structures of the g and h is as a multi-way pooling strategy.

This particular pooling structure was chosen to study the potential of learning the kind of bilinear interaction that exists between the $g_{.k}$ and $h_{.k}$ within a block. The f_k are present to promote block cohesion by gating the interaction of between $g_{.k}$ and $h_{.k}$ and the visible vector v .

This higher-order structure is of course just one choice of many possible higher-

order interaction architectures. One can easily imagine defining arbitrary overlapping pooling regions, with the number of overlapping pooling regions specifying the order of the latent variable interaction. We believe that explorations of overlapping pooling regions of this type is a promising direction of future inquiry. One potentially interesting direction is to consider overlapping blocks (such as our f blocks). The overlap will define a topology over the features as they will share lower-level features (i.e. the slab variables). A topology thus defined could potentially be exploited to build higher-level data representations that possess local receptive fields. These kind of local receptive fields have been shown to be useful in building large and deep models that perform well in object classification tasks in natural images (Coates et al., 2011).

12.3.2 Variational inference and unsupervised learning

Due to the multiplicative interaction between the latent variables f , g and h , computation of $P(f | v)$, $P(g | v)$ and $P(h | v)$ is intractable. While the slab variables also interact multiplicatively, we are able to analytically marginalize over them. Consequently we resort to a variational approximation of the joint conditional $P(f, g, h | v)$ with the standard mean-field structure. i.e. we choose $Q_v(f, g, h) = Q_v(f)Q_v(g)Q_v(h)$ such that the KL divergence $\text{KL}(Q_v(f, g, h) || P(f, g, h | v))$ is minimized, or equivalently, that the variational lower bound $\mathcal{L}(Q_v)$ on the log likelihood of the data is maximized:

$$\max_{Q_v} \mathcal{L}(Q_v) = \max_{Q_v} \sum_{f, g, h} Q_v(f)Q_v(g)Q_v(h) \log \left(\frac{p(f, g, h | v)}{Q_v(f)Q_v(g)Q_v(h)} \right),$$

where the sums are taken over all values of the elements of f , g and h respectively. Maximizing this lower bound with respect to the variational parameters $\hat{f}_k \equiv Q_v(f_k = 1)$, $\hat{g}_{ik} \equiv Q_v(g_{ik} = 1)$ and $\hat{h}_{jk} \equiv Q_v(h_{jk} = 1)$, results in the set of

approximating factored distributions:

$$\begin{aligned}\hat{f}_k &= \text{sigm} \left(c_{ik} + \sum_{i,j} v^T W_{.ijk} \mu_{ijk} \hat{g}_{ik} \hat{h}_{jk} + \frac{1}{2} \sum_{i,j} \alpha_{ijk}^{-1} [v^T W_{.ijk}]^2 \hat{g}_{ik} \hat{h}_{jk} \right), \\ \hat{g}_{ik} &= \text{sigm} \left(c_{ik} + \sum_j v^T W_{.ijk} \mu_{ijk} \hat{h}_{jk} \hat{f}_k + \frac{1}{2} \sum_j \alpha_{ijk}^{-1} [v^T W_{.ijk}]^2 \hat{h}_{jk} \hat{f}_k \right), \\ \hat{h}_{jk} &= \text{sigm} \left(d_{jk} + \sum_i v^T W_{.ijk} \mu_{ijk} \hat{g}_{ik} \hat{f}_k + \frac{1}{2} \sum_i \alpha_{ijk}^{-1} [v^T W_{.ijk}]^2 \hat{g}_{ik} \hat{f}_k \right).\end{aligned}$$

The above equations form a set of fixed point equations which we iterate until the values of all $Q_v(f_k)$, $Q_v(g_{ik})$ and $Q_v(h_{jk})$ converge. Since the expression for \hat{f}_k does not depend on $\hat{f}_{k'}$, $\forall k'$, \hat{g}_{ik} does not depend on $\hat{g}_{i'k'}$, $\forall i', k'$, and \hat{h}_{jk} does not depend on $\hat{h}_{j'k'}$, $\forall i', k'$, we can define a three stage update strategy where we update the values of all K values of \hat{f} in parallel, then update all $K \times M$ values of \hat{g} in parallel and finally update all $K \times N$ values of \hat{h} in parallel.

Following the variational EM training approach (Saul et al., 1996), we alternately maximize the lower bound $\mathcal{L}(Q_v)$ with respect to the variational parameters \hat{f} , \hat{g} and \hat{h} (E-step) and maximizing $\mathcal{L}(Q_v)$ with respect to the model parameters (M-step). The gradient of $\mathcal{L}(Q_v)$ with respect to the model parameters $\theta = \{W, \mu, \alpha, \Lambda, b, c, d, e\}$ is given by:

$$\frac{\partial \mathcal{L}(Q_v)}{\partial \theta} = \left\{ \sum_{f,g,h} Q_v(f) Q_v(g) Q_v(h) \mathbb{E}_{p(s|v,f,g,h)} \left[-\frac{\partial E}{\partial \theta} \right] \right\} + \mathbb{E}_{p(v,s,g,h)} \left[\frac{\partial E}{\partial \theta} \right] \quad (12.1)$$

where E is the energy function given in Eq. 12.1. As is evident from Eq. 12.1, the gradient of $\mathcal{L}(Q_v)$ with respect to the model parameters contains two terms: a positive phase that depends on the data v and a negative phase, derived from the partition function of the joint $p(v, s, f, g, h)$ that does not. We adopt a training strategy similar to that of Salakhutdinov and Hinton (2009a), in that we combine a variational approximation of the positive phase of the gradient with a block Gibbs sampling-based stochastic approximation of the negative phase. Our Gibbs sampler alternately samples, in parallel, each set of random variables, sampling from $p(f | v, g, h)$, $p(g | v, f, h)$, $p(h | v, f, g)$, $p(s | v, f, g, h)$, and finally sampling from $p(v | f, g, h, s)$.

12.3.3 The Challenge of Unsupervised Learning to Disentangle

Above we have briefly outlined our procedure for training the unsupervised learning. The web of interactions between the latent random variables, particularly those between g and h , makes the unsupervised learning of the model parameters a particularly challenging learning problem. It is the difficulty of learning that motivates our block-wise organization of the interactions between the g and h variables. The block structure allows the interactions between g and h to remain local, with each g interacting with relatively few h and each h interacting with relatively few g . This local neighborhood structure allows the inference and learning procedures to better manage the complexities of teasing apart the latent variable interactions and adapting the model parameters to (approximately) maximize likelihood.

By using many of these blocks of local interactions we can leverage the known tractable learning properties of models such as the RBM. Specifically, if we consider each block as a kind of super hidden unit gated by f , then with no interactions across blocks (apart from those mediated by the mutual connections to the visible units) the model assumes the form of an RBM.

While our chosen interaction structure allows our higher-order model to be able to learn, one consequence is that the model is only capable of disentangling relatively local factors that appear within a single block. We suggest that one promising avenue to accomplish more extensive disentangling is to consider stacking multiple version of the proposed model and consider layer-by-layer disentangling of the factors of variation present in the data. The idea is to start with local disentangling and move gradually toward disentangling non local and more abstract factors.

12.4 Related Work

The model proposed here was strongly influenced by previous attempts to disentangle factors of variation in data using latent variable models. One of the earlier efforts in this direction also used higher-order interactions of latent variables, specifically bilinear (Tenenbaum and Freeman, 2000; Grimes and Rao, 2005) and

multilinear (Vasilescu and Terzopoulos, 2005) models. One critical difference between these previous attempts to disentangle factors of variation and our method is that unlike these previous methods, we are attempting to learn to disentangle from entirely unsupervised information. In this way, one can interpret our approach as an attempt to extend the subspace feature pooling approach to the problem of disentangling factors of variation.

Bilinear models are essentially linear models where the higher-level state is factored into the product of two variables. Formally, the elements of observation x are given by $x_k = \sum_i \sum_j W_{ijk} y_i z_j$, $\forall k$, where y_i and z_j are elements of the two factors (y and z) representing the observation and W_{ijk} is an element of the tensor of model parameters (Tenenbaum and Freeman, 2000). The tensor W can be thought of as a generalization of the typical weight matrix found in most unsupervised models we have considered above. Tenenbaum and Freeman (2000) developed an EM-based algorithm to learn the model parameters and demonstrated, using images of letters from a set of distinct fonts, that the model could disentangle the style (font characteristics) from content (letter identity). Grimes and Rao (2005) later developed a bilinear sparse coding model of a similar form as described above but included additional terms to the objective function to render the elements of both y and z sparse. They also require observation of the factors in order to train the model, and used the model to develop transformation invariant features of natural images. Multilinear models are simply a generalization of the bilinear model where the number of factors that can be composed together is 2 or more. Vasilescu and Terzopoulos (2005) develop a multilinear ICA model, which they use to model images of faces, to disentangle factors of variation such as illumination, views (orientation of the image plane relative to the face) and identities of the people.

Hinton et al. (2011) also propose to disentangle factors of variation by learning to extract features associated with pose parameters, where the changes in pose parameters (but not the feature values) are known at training time. The proposed model is also closely related to recent work (Memisevic and Hinton, 2010), where higher-order Boltzmann Machines are used as models of spatial transformations in images. While there are a number of differences between this model and ours, the most significant difference is our use of multiplicative interactions between *latent* variables. While they included higher-order interactions within the Boltzmann en-

ergy function, they were used exclusively between observed variables, dramatically simplifying the inference and learning procedures. Another major point of departure is that instead of relying on low-rank approximations to the weight tensor, our approach employs highly structured and sparse connections between latent variables (e.g. g_{ik} is not interact with $h_{jk'}$ for $k' \neq k$), reminiscent of recent work on structured sparse coding (Gregor et al., 2011) and structured l_1 -norms (Bach et al., 2011). As discussed above, our use of a sparse connection structure allows us to isolate groups of interacting latent variables. Keeping the interactions local in this way, is a key component of our ability to successfully learn using only unsupervised data.

12.5 Experiments

12.5.1 Toy Experiment

We showcase the ability of our model to disentangle factors of variation, by training it on a synthetic dataset, a subset of which is shown in Fig. 12.3 (top). Each color image, of size 3×20 is composed of one basic object of varying color, which can appear at five different positions. The constraint is that all objects in a given image must be of the same color. Additive Gaussian noise is superimposed on the resulting images to facilitate mixing of the RBM negative phase. A bilinear ssRBM with $M = 3$ and $N = 5$ should in theory have the capacity to disentangle the two factors of variation present in the data, as there are 2^3 possible colors and 2^5 configurations of object placement. The resulting filters are shown in Fig. 12.3 (bottom): the model has successfully learnt a binary encoding of color along g -units (rows) and positions along h (columns). Note that this would have been extremely difficult to perform without multiplicative interactions of latent variables: an RBM with 15 hidden units technically has the capacity to learn similar filters, however it would be incapable of enforcing mutual exclusivity between hidden units of different color. The bilinear ssRBM model on the other hand generates near-perfect samples (not shown), while factoring the representation for use in deeper layers.

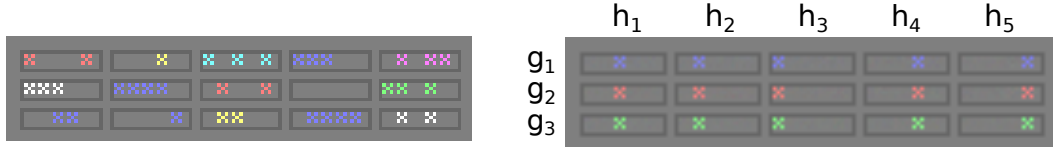


Figure 12.3: (top) Samples from our synthetic dataset (before noise). In each image, a figure “X” can appear at five different positions, in one of eight basic colors. Objects in a given image must all be of the same color. (bottom) Filters learnt by a bilinear ssRBM with $M = 3$, $N = 5$, which successfully show disentangling of color information (rows) from position (columns).

12.5.2 Toronto Face Dataset

We evaluate our model on the recently introduced Toronto Face Dataset (TFD) (Susskind et al., 2010), which contains a large number of black & white 48×48 preprocessed facial images. These span a wide range of identities and emotions and as such, the dataset is well suited to study the problem of disentangling: models which can successfully separate identity from emotion should perform well at the supervised learning task, which involves classifying images into one of seven categories: {anger, disgust, fear, happy, sad, surprise, neutral}. The dataset is divided into two parts: a large unlabeled set (meant for unsupervised feature learning) and a smaller labeled set. Note that emotions appear much more prominently in the latter, since these are acted out and thus prone to exaggeration. In contrast, most of the unlabeled set contains natural expressions over a wider range of individuals.

In the course of this work, we have made several key refinements to the original spike-and-slab formulation. Notably, since the slab variables $\{s_{ijk}; \forall j\}$ can be interpreted as coordinates in the subspace of the spike variable g_{ik} (which spans the set of filters $\{W_{.ijk}, \forall j\}$), it is natural for these filters to be unit-norm. Each maximum likelihood gradient update is thus followed by a projection of the filters onto the unit-norm ball. Similarly, there exists an over-parametrization in the direction of $W_{.ijk}$ and the sign of μ_{ijk} , the parameter controlling the mean of s_{ijk} . We thus constrain μ_{ijk} to be positive, in our case greater than 1. Similar constraints are applied on B and α to ensure that the variances on the visible and slab variables remain bounded. While previous work (Courville et al., 2011b) used the expected value of the spike variables as the input to classifiers, or higher-layers in deep networks, we found that the above re-parametrization consistently lead to better results when using the product of expectations of h and s . For pooled models, we simply take

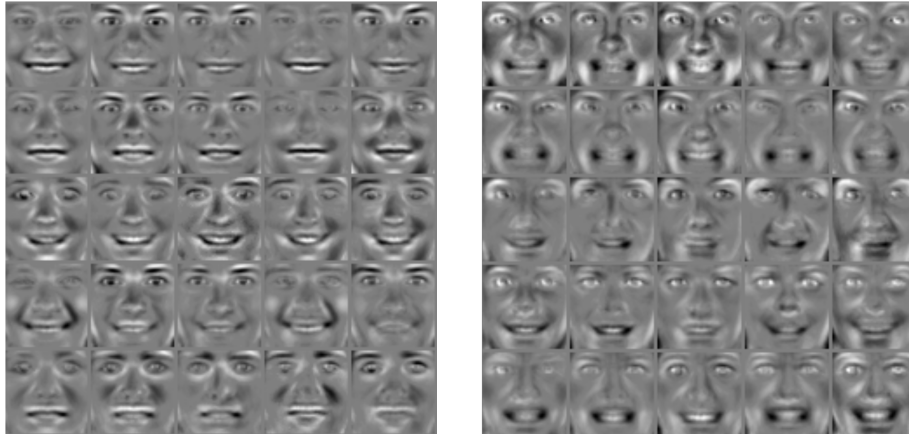


Figure 12.4: Example blocks obtained with $K = 100$, $M = N = 5$. The filters (inner-most dimension of tensor W) in each block exhibit global cohesion, specializing themselves to a subset of identities and emotions: {happiness, fear, neutral} in (left) and {happiness, anger} in (right). In both cases, g -units (which pool over columns) encode emotions, while h -units (which pool over rows) are more closely tied to identity.

the product of each binary spike, with the norm of its associated slab vector.

Disentangling Emotion from Identity. We begin with a qualitative evaluation of our model, by visualizing the learned filters (inner-most dimension of the matrix W) and pooling structures. We trained a model with $K = 100$ and $M = N = 5$ (that is to say 100 blocks of 5×5 interacting g and h units) on a weighted combination of the labeled and unlabeled training sets. Doing so (as opposed to training on the unlabeled set only) allows for greater interpretability of the results, as emotion is a more prominent factor of variation in the labeled set). The results, shown in Figure 12.4, clearly show global cohesion within blocks pooled by f_k , with row and column structure correlating with variances in appearance/identity and emotions.

Disentangling via Unsupervised Feature Learning. We now evaluate the representation learnt by our disentangling RBM, by measuring its usefulness for the task of emotion recognition. Our main objective here is to evaluate the usefulness of disentangling, over traditional approaches of pooling, as well as the use of larger, unpooled models. We thus consider ssRBMs with 3000 and 5000 features, with either (i) no pooling (i.e. $K = 5000$ spikes with $N = 1$ slabs per spike), (ii) pooling

Model	K	M	N	Factored		Unfactored	
				valid	test	valid	test
ssRBM	3000		1	n/a	n/a	76.0%	75.7%
ssRBM	999		3	72.9%	74.4%	74.9%	73.5%
hossRBM	330	3	3	76.0%	75.7%	75.3%	75.2%
hossRBM	120	5	5	71.4%	70.7%	74.5%	74.2%
ss-RBM	5000		1	n/a	n/a	76.7%	76.3%
ss-RBM	1000		5	74.2%	74.0%	75.9%	74.6%
hossRBM	555	3	3	77.6%	77.4%	76.2%	75.9%
hossRBM	200	5	5	73.3%	73.3%	75.6%	75.3%

Table 12.1: Classification accuracy for Toronto Face Dataset. We compare our higher-order ssRBM for various block sizes K and pooling regions $M \times N$. The comparison is against first-order ssRBMs, which thus pool in a single dimension of size N . First four models contain approximately 3,000 filters, while bottom four contain 5,000. In both cases, we compare the effect of using the factored representation, to the unfactored representation.

along a single dimension (i.e. $K = 1000$ spike variables, pooling $N = 5$ slabs) or (iii) disentangled through our higher-order ssRBM (i.e. $K = 200$, with g and h units arranged in a $M \times N$ grid, with $M = N = 5$).

We followed the standard TFD training protocol of performing unsupervised training on the unlabeled set, and then using the learnt representation as input to a linear SVM, trained and cross-validated on the labeled set. Table 12.1 shows the test accuracy obtained by various spike-and-slab models, averaged over the 5-folds.

We report two sets of numbers for models with pooling or disentangling: one where we use the “factored representation”, which is the element-wise product of spike variables with the norm of their associated slab vector, and the “unfactored representation”: the higher-dimensional representation formed by considering all slab variables, each multiplied by their associated spikes.

We can see that the higher-order ssRBM achieves the best result: 77.4%, using the factored representation. The fact that that our model outperforms the “unfactored” one, confirms our disentangling hypothesis: our model has successfully learnt a lower-dimensional (factored) representation of the data, useful for classification. For reference, a linear SVM classifier on the pixels achieves 71.5% (Susskind et al., 2010), an MLP trained with supervised backprop 72.72%ⁱ, while a deep mPoT

i. Salah Rifai, personal communication.

model (Ranzato et al., 2011), which exploits local receptive fields achieves 82.4%.

12.6 Conclusion

We have presented a higher-order extension of the spike-and-slab restricted Boltzmann machine that factors the standard binary spike variable into three interacting factors. From a generative perspective, these interactions act to entangle the factors represented by the latent binary variables. Inference is interpreted as a process of disentangling the factors of variation in the data. As previously mentioned, we believe an important direction of future research to be the exploration of methods to gradually disentangle the factors of variation by stacking multiple instantiations of proposed model into a deep architecture.



Variational Inference

A.0.1 Variational Lower Bound

The log-likelihood of $p(v)$ can be decomposed into the variational lower-bound $\mathcal{L}(q)$ and a KL divergence term as shown below. A similar derivation is obtained for discrete random variables by replacing integrals with summation. To unclutter notation, we omit dh to indicate that the integrals are computed over the latent variables.

$$\begin{aligned}\log p(v) &= \log p(v) + \int q(h | v) \log \left(\frac{p(v, h)}{q(h | v)} \right) - \int q(h | v) \log \left(\frac{p(v, h)}{q(h | v)} \right) \\ &= \int q(h | v) \log p(v) dh + \int q(h | v) \log \left(\frac{p(v, h)}{q(h | v)} \right) - \int q(h | v) \log \left(\frac{p(v, h)}{q(h | v)} \right) \\ &= \int_h q(h | v) \log \left(\frac{p(v, h)}{q(h | v)} \right) - \int_h q(h | v) \left[\log \left(\frac{p(v, h)}{q(h | v)} \right) - \log p(v) \right] \\ &= \int q(h | v) \log \left(\frac{p(v, h)}{q(h | v)} \right) - \int q(h | v) \left[\log \left(\frac{p(v, h)}{p(v) q(h | v)} \right) \right] \\ &= \int q(h | v) \log \left(\frac{p(v, h)}{q(h | v)} \right) - \int q(h | v) \left[\log \left(\frac{p(h | v)}{q(h | v)} \right) \right] \\ &= \mathcal{L}(q) + \text{KL}(q||p)\end{aligned}$$

with $\mathcal{L}(q)$ and $\text{KL}(q||p)$ defined as:

$$\begin{aligned}\mathcal{L}(q, \theta) &= \int q(h) \log \left\{ \frac{p(x, h; \theta)}{q(h)} \right\} \\ \text{KL}(q||p) &= - \int q(h) \log \left\{ \frac{p(h | x; \theta)}{q(h)} \right\}\end{aligned}$$

A.0.2 Euler-Lagrange Equations

Assuming a factorial form for the variational distribution such that $q(Z) = \prod_{i=1}^M q_i(Z_i)$, the mean-field update equations are given by the Euler-Lagrange equations shown below. Note that these are simply an expanded version of the derivation found in Bishop (2006).

$$\begin{aligned}
\mathcal{L}(q) &= \int q(Z) \ln \frac{p(X, Z)}{q(Z)} dZ \\
&= \int \prod_i q_i \left\{ \ln p(X, Z) - \sum_k \ln q_k \right\} dZ \\
&= \int q_j \left\{ \int \prod_{i \neq j} q_i \left\{ \ln p(X, Z) - \sum_k \ln q_k \right\} dZ_i \right\} dZ_j \\
&= \int q_j \left\{ \int \prod_{i \neq j} q_i \ln p(X, Z) dZ_i - \int \prod_{i \neq j} q_i \left[\sum_k \ln q_k \right] dZ_i \right\} dZ_j \\
&= \int q_j \left\{ \int \ln p(X, Z) \prod_{i \neq j} q_i dZ_i - \int \prod_{i \neq j} q_i \left[\sum_k \ln q_k \right] dZ_i \right\} dZ_j \\
&= \int q_j \left\{ \int \ln p(X, Z) \prod_{i \neq j} q_i dZ_i - \int \left[\prod_{i \neq j} q_i \ln q_1 + \cdots + \prod_{i \neq j} q_i \ln q_j + \cdots + \prod_{i \neq j} q_i \ln q_M \right] \prod_{i \neq j} dZ_i \right\} dZ_j \\
&= \int q_j \left\{ \int \ln p(X, Z) \prod_{i \neq j} q_i dZ_i - \left[\int \prod_{i \neq j} q_i (\ln q_1) \prod_{i \neq j} dZ_i + \cdots \right. \right. \\
&\quad \left. \left. \int \prod_{i \neq j} q_i (\ln q_j) \prod_{i \neq j} dZ_i + \cdots + \int \prod_{i \neq j} q_i (\ln q_M) \prod_{i \neq j} dZ_i \right] \right\} dZ_j \\
&= \int q_j \left\{ \int \ln p(X, Z) \prod_{i \neq j} q_i dZ_i - \left[\int q_1 \ln q_1 dZ_1 + \int q_2 \ln q_2 dZ_2 + \cdots + \int q_j \ln q_j + \cdots + \int q_M \ln q_M dZ_M \right] \right\} dZ_j \\
&= \int q_j \left\{ \int \ln p(X, Z) \prod_{i \neq j} q_i dZ_i - \ln q_j \right\} dZ_j - \sum_{i \neq j} \int q_i \ln q_i dZ_i \\
&= \int q_j \left\{ \mathbb{E}_{\prod_{i \neq j} q_i} [\ln p(X, Z)] \right\} dZ_j - \int q_j \ln(q_j) dZ_j + cte
\end{aligned}$$

Here, the entropy terms on q_i with $i \neq j$ are considered constants, because we increase the lower bound wrt. one q_j , iterating over $q_j \in [1, M]$. Lets define $\ln \tilde{p}(X, Z_j) = \mathbb{E}_{i \neq j} [\ln p(X, Z)]$. The above is actually a Kullback-Divergence as

can be shown below:

$$\begin{aligned}
\mathcal{L}(q) &= \int q_j \left\{ \mathbb{E}_{\prod_{i \neq j} q_i} [\ln p(X, Z)] \right\} dZ_j - \int q_j \ln(q_j) dZ_j + cte \\
&= \int q_j \{ \ln \tilde{p}(X, Z_j) \} dZ_j - \int q_j \ln(q_j) dZ_j + cte \\
&= \int q_j \ln \left[\frac{\tilde{p}(X, Z_j)}{q_j} \right] dZ_j + cte \\
&= -KL(q_j \| \tilde{p}(X, Z_j))
\end{aligned}$$

Maximizing $\mathcal{L}(q)$ with respect to q_j is thus equivalent to minimizing the KL divergence term, which is solved trivially by:

$$\begin{aligned}
q_j^*(Z_j) &= \tilde{p}(X, Z_j) \ln q_j^* \\
&= \mathbb{E}_{i \neq j} [\ln p(X, Z)]
\end{aligned}$$

B

On Tracking The Partition Function

The supplementary material provides a more comprehensive view of our log partition function tracking algorithm. The notation is mostly identical to that of the paper and is summarized below for convenience. One minor difference however, is that we make widespread use of the notations $A_{:,l}$ and $A_{m,:}$ to indicate the vectors formed by the l -th column and m -th row of matrix A (respectively).

$q_{i,t}$	RBM at inverse temp. β_i at time-step t . $q_{i,t}(x) = \tilde{q}_{i,t}(x)/Z_{i,t}$, with $i \in [1, M]$.
θ	set of model parameters.
$F_{i,t}(x)$	free-energy assigned by $q_{i,t}$ to input configuration x .
$\zeta_{i,t}$	log partition function of model $q_{i,t}$.
$\mathcal{X}_{i,t}$	mini-batch of samples $\{x_{i,t}^{(n)} \sim q_{i,t}(x); n \in [1, N]\}$.
\mathcal{Y}_t	additional mini-batch of samples $\{x_{1,t}^{(n)} \sim q_{1,t}(x); n \in [1, N_Y]\}$, $N_Y \geq N$.
\mathcal{D}	set of training examples.
$\mu_{t,t}, P_{t,t}$	estimated mean and covariance of posterior distribution $p(\zeta_t O_{t:0}^{(\Delta t)}, O_{t:0}^{(\Delta \beta)})$.
Σ_ζ	fixed covariance matrix of $p(\zeta_t \zeta_{t-1})$.
$\epsilon_{init}, \epsilon_t$	initial learning rate and rate at time t .
$s_{i,t}$	coarse estimate of $Z_{i+1,t}/Z_{i,t}$ used to generate bridging distribution q^* .
$q_{i,t}^*$	bridging distribution with $q_{i,t}^*(x) = \frac{\tilde{q}_{i,t}(x)\tilde{q}_{i+1,t}(x)}{s_{i,t}\tilde{q}_{i,t}(x) + \tilde{q}_{i+1,t}(x)}$.
$r(q_1, q_2, x_1, x_2)$	swap probability used by PT, with $r_{i,t} = \min\left(1, \frac{\tilde{q}_1(\mathbf{x}_2)\tilde{q}_2(\mathbf{x}_1)}{\tilde{q}_1(\mathbf{x}_1)\tilde{q}_2(\mathbf{x}_2)}\right)$.

B.1 Parallel Tempering Algorithm

We divide our algorithm into three parts. Algorithm 9 presents the pseudo-code for the Parallel Tempering sampling algorithm. For details, we refer the reader to Desjardins et al. (2010b).

Algorithm 9 `sample_PT` ($q_{:,t}, \mathcal{X}_{:,t-1}, k$)

Initialize $\mathcal{X}_{i,t}$ as empty sets, $\forall i \in [1, M]$.

for $n \in [1, N]$ **do**

for $i \in [1, M]$ **do**

 Initialize Markov chain associated with model $q_{i,t}$ with state $x_{i,t-1}^{(n)}$.

 Perform k steps of Gibbs sampling, yielding $x_{i,t}^{(n)}$.

end for

 Swap $x_{i,t}^{(n)} \leftrightarrow x_{i+1,t}^{(n)}$ with prob. $r(q_{i,t}, q_{i+1,t}, x_{i,t}^{(n)}, x_{i+1,t}^{(n)})$, \forall even i .

 Swap $x_{i,t}^{(n)} \leftrightarrow x_{i+1,t}^{(n)}$ with prob. $r(q_{i,t}, q_{i+1,t}, x_{i,t}^{(n)}, x_{i+1,t}^{(n)})$, \forall odd i .

$\mathcal{X}_{i,t} \leftarrow \mathcal{X}_{i,t} \cup \{x_{i,t}^{(n)}\}$, $\forall i$.

end for

return $\mathcal{X}_{:,t}$.

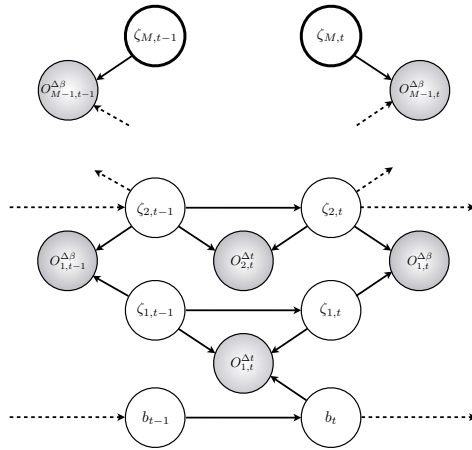
B.2 Kalman Filter

Algorithm 10 presents the tracking algorithm. The statistical estimate $O_{i,t}^{(\Delta t)}$ is computed as an average of importance weights, measured between adjacent models $q_{i,t}$ and $q_{i,t-1}$. The estimate $O_{i,t}^{(\Delta \beta)}$ is computed through bridge sampling, applied to models $q_{i+1,t}$ and $q_{i,t}$. These observations are combined through a Kalman filter, which also exploits a smoothness prior on the evolution of ζ_t .

We include the graphical model, system equations and inference equations in Figures B.1 & B.2 for completeness. We however refer the reader to the accompanying paper for a more thorough description of these figures.

B.3 Simultaneous Tracking and Learning

Finally, Algorithm 11 ties everything together, performing joint training and estimation of the log partition function. Note that using two sets of samples from the target distribution ($\mathcal{X}_{1,t}$ and \mathcal{Y}_t) is not required. Their use is inspired from Salakhutdinov (2010a) and allows us to separately tune N , the number of “tempered” mini-batches and N_Y , the size of the mini-batch used to estimate the gradient.



System Equations:

$$p(\zeta_0) = \mathcal{N}(\mu_0, \Sigma_0)$$

$$p(\zeta_t | \zeta_{t-1}) = \mathcal{N}(\zeta_{t-1}, \Sigma_\zeta)$$

$$p(O_t^{(\Delta t)} | \zeta_t, \zeta_{t-1}) = \mathcal{N}(C[\zeta_t, \zeta_{t-1}]^T, \Sigma_{\Delta t})$$

$$p(O_t^{(\Delta\beta)} | \zeta_t) = \mathcal{N}(H\zeta_t, \Sigma_{\Delta\beta})$$

$$C = \begin{bmatrix} 1 & 0 \\ I_M & -I_M \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} -1 & +1 & 0 & 0 & 0 \\ 0 & -1 & +1 & 0 & \vdots & 0 \\ & & \dots & & & 0 \\ 0 & 0 & 0 & -1 & +1 & 0 \end{bmatrix}$$

Figure B.1: Graphical model capturing the evolution of the log-partition function

A directed graphical model for log partition function tracking. The shaded nodes represent observed variables, and the double-walled nodes represent the tractable $\zeta_{M,\cdot}$ with $\beta_M = 0$. For clarity of presentation, we show the bias term as distinct from the other $\zeta_{i,t}$ (recall $b_t = \zeta_{M+1,t}$.)

Inference Equations:

- (i) $p\left(\zeta_{t-1}, \zeta_t \mid O_{t-1:0}^{(\Delta t)}, O_{t-1:0}^{(\Delta \beta)}\right) = \mathcal{N}(\eta_{t-1,t-1}, V_{t-1,t-1})$
 with $\eta_{t-1,t-1} = \begin{bmatrix} \mu_{t-1,t-1} \\ \mu_{t-1,t-1} \end{bmatrix}$ and $V_{t-1,t-1} = \begin{bmatrix} P_{t-1,t-1} & P_{t-1,t-1} \\ P_{t-1,t-1} & \Sigma_\zeta + P_{t-1,t-1} \end{bmatrix}$
- (ii) $p(\zeta_{t-1}, \zeta_t \mid O_{t:0}^{(\Delta t)}, O_{t-1:0}^{(\Delta \beta)}) = \mathcal{N}(\eta_{t,t-1}, V_{t,t-1})$
 with $V_{t,t-1} = (V_{t-1,t-1}^{-1} + C^T \Sigma_{\Delta t}^{-1} C)^{-1}$ and $\eta_{t,t-1} = V_{t,t-1} (C^T \Sigma_{\Delta t} O_t^{(\Delta t)} + V_{t-1,t-1}^{-1} \eta_{t-1,t-1})$
- (iii) $p\left(\zeta_t \mid O_{t:0}^{(\Delta t)}, O_{t-1:0}^{(\Delta \beta)}\right) = \mathcal{N}(\mu_{t,t-1}, P_{t,t-1})$ with $\mu_{t,t-1} = [\eta_{t,t-1}]_2$ and $P_{t,t-1} = [V_{t,t-1}]_{2,2}$
- (iv) $p(\zeta_t \mid O_{t:0}^{(\Delta t)}, O_{t:0}^{(\Delta \beta)}) = \mathcal{N}(\mu_{t,t}, P_{t,t})$
 with $P_{t,t} = (P_{t,t-1}^{-1} + H^T \Sigma_{\Delta \beta}^{-1} H)^{-1}$ and $\mu_{t,t} = P_{t,t} (H^T \Sigma_{\Delta \beta} O_t^{(\Delta \beta)} + P_{t,t-1}^{-1} \mu_{t,t-1})$

Figure B.2: Inference equations for tracking the partition function

Inference equations for our log partition tracking algorithm, a variant on the Kalman filter. For any vector v and matrix V , we use the notation $[v]_2$ to denote the vector obtained by preserving the bottom half elements of v and $[V]_{2,2}$ to indicate the lower right-hand quadrant of V .

Algorithm 10 `kalman_filter` ($q_{:,t-1}, q_{:,t}, \mu_{t-1,t-1}, P_{t-1,t-1}, s_{i,t}, \Sigma_\zeta$)

Using $\mu_{t-1,t-1}$, $P_{t-1,t-1}$ and Σ_ζ , compute $\eta_{t-1,t-1}$ and $V_{t-1,t-1}$ through equation (i).

for $i \in [1, M]$ **do**

$$w_{i,t}^{(n)} \leftarrow \frac{\tilde{q}_{i,t}(x_{i,t-1}^{(n)})}{\tilde{q}_{i,t-1}(x_{i,t-1}^{(n)})}; \quad O_{i,t}^{(\Delta t)} \leftarrow \log \left[\frac{1}{N} \sum_{n=1}^N w_{i,t}^{(n)} \right]; \quad \Sigma_{\Delta t} \leftarrow \text{Diag} \left[\frac{\text{Var}[w_{i,t}]}{(\sum_n w_{i,t}^{(n)})^2} \right]$$

end for

Using $O_{i,t}^{(\Delta t)}$ and $\Sigma_{\Delta t}$, compute $\eta_{t,t-1}$ and $V_{t,t-1}$ through equation (ii).

Compute $\mu_{t,t-1}$ and $P_{t,t-1}$ using equation (iii).

for $i \in [1, M-1]$ **do**

$$u_{i,t}^{(n)} \leftarrow \frac{q_{i,t}^*(x_{i,t}^{(n)})}{\tilde{q}_{i,t}(x_{i,t}^{(n)})}; \quad v_{i,t}^{(n)} \leftarrow \frac{q_{i,t}^*(x_{i+1,t}^{(n)})}{\tilde{q}_{i+1,t}(x_{i+1,t}^{(n)})}$$

$$O_{i,t}^{(\Delta \beta)} \leftarrow \log \frac{1}{N} \sum_{n=1}^N u_{i,t}^{(n)} - \log \frac{1}{N} \sum_{n=1}^N v_{i,t}^{(n)}$$

$$\Sigma_{\Delta \beta} \leftarrow \text{Diag} \left[\frac{\text{Var}[u_{i,t}]}{(\sum_n u_{i,t}^{(n)})^2} + \frac{\text{Var}[v_{i,t}]}{(\sum_n v_{i,t}^{(n)})^2} \right]$$

end for

Using $O_{i,t}^{(\Delta \beta)}$ and $\Sigma_{\Delta \beta}$, compute $\eta_{t,t}$ and $V_{t,t}$ through equation (iv).

Return $(\eta_{t,t}, V_{t,t})$.

Algorithm 11 main

Initialize θ_1 and compute exact log partition functions $\zeta_{:,1}$.
Initialize $\mu_{1,1}$ with $\zeta_{:,1}$, $P_{1,1}[1 : M, 1 : M] = 0$ and $P_{1,1}[M + 1, M + 1] = \epsilon_{init} \cdot \sigma_b^2$.
Initialize samples $\mathcal{X}_{:,1}$ and $\mathcal{Y}_{:,1}$ according to the RBM visible biases.
Initialize $s_{i,1}$ to $\exp(\zeta_{i+1,1} - \zeta_{i,1})$, $\forall i \in [1, M - 1]$.

for $t \in [2, T]$ **do**

Obtain training examples $\mathcal{X}_t^+ = \{x^{(n)} \in \mathcal{D}; n \in [1, N]\}$

$\theta_t \leftarrow \theta_{t-1} - \epsilon_t \left(\frac{1}{N} \sum_{x \in \mathcal{X}_t^+} \left[\frac{\partial F(x; \theta_t)}{\partial \theta} \right] - \frac{1}{N} \sum_{y \in \mathcal{Y}_t} \left[\frac{\partial F(y; \theta_{F,t})}{\partial \theta} \right] \right)$.

Choose N samples from \mathcal{Y}_t to swap with $\mathcal{X}_{1,t}$.

$\mathcal{X}_{:,t} \leftarrow \text{sample_PT}(q_{:,t}, \mathcal{X}_{:,t-1}, k)$.

$\mathcal{Y}_t \leftarrow \text{sample_Gibbs}(q_{1,t}, \mathcal{Y}_{t-1}, k)$.

$(\mu_{t,t}, P_{t,t}) \leftarrow \text{kalman_filter}(q_{:,t-1}, q_{:,t}, \mu_{t-1,t-1}, P_{t-1,t-1}, s_{i,t}, \Sigma_\zeta)$

$\hat{\zeta}_{:,t} \leftarrow \mu_{t,t}; \quad s_{i,t+1} \leftarrow \exp(\hat{\zeta}_{i+1,t} - \hat{\zeta}_{i,t})$, $\forall i \in [1, M - 1]$.

end for



Metric-Free Natural Gradient

We include the following derivations for completeness.

C.0.1 Expected Hessian of $\log Z$ and Fisher Information.

$$\begin{aligned}\mathbb{E}_{p_\theta} \left[-\frac{\partial^2 \log p(x)}{\partial \theta_i \partial \theta_j} \right] &= \mathbb{E}_{p_\theta} \left[\frac{1}{p(x)^2} \frac{\partial p(x)}{\partial \theta_j} \frac{\partial p(x)}{\partial \theta_i} - \frac{1}{p(x)} \frac{\partial^2 p(x)}{\partial \theta_i \partial \theta_j} \right] \\ &= \mathbb{E}_{p_\theta} \left[\left(\frac{1}{p(x)} \frac{\partial p(x)}{\partial \theta_i} \right) \left(\frac{1}{p(x)} \frac{\partial p(x)}{\partial \theta_j} \right) - \frac{1}{p(x)} \frac{\partial^2 p(x)}{\partial \theta_i \partial \theta_j} \right] \\ &= \mathbb{E}_{p_\theta} \left[\frac{\partial \log p(x)}{\partial \theta_i} \frac{\partial \log p(x)}{\partial \theta_j} \right] - \sum_x p(x) \frac{1}{p(x)} \frac{\partial^2 p(x)}{\partial \theta_i \partial \theta_j} \\ &= \mathbb{E}_{p_\theta} \left[\frac{\partial \log p(x)}{\partial \theta_i} \frac{\partial \log p(x)}{\partial \theta_j} \right] - \sum_x \frac{\partial^2 p(x)}{\partial \theta_i \partial \theta_j} \\ &= \mathbb{E}_{p_\theta} \left[\frac{\partial \log p(x)}{\partial \theta_i} \frac{\partial \log p(x)}{\partial \theta_j} \right] - \frac{\partial^2 \sum_x p(x)}{\partial \theta_i \partial \theta_j} \\ &= \mathbb{E}_{p_\theta} \left[\frac{\partial \log p(x)}{\partial \theta_i} \frac{\partial \log p(x)}{\partial \theta_j} \right]\end{aligned}$$

C.0.2 Derivation of Equation 8.5

$$\begin{aligned}
 \log p(x) &= -E(x) - \log Z \\
 \frac{\partial \log p(x)}{\partial \theta_i} &= -\frac{\partial E(x)}{\partial \theta_i} - \frac{1}{Z} \sum_x \frac{\partial}{\partial \theta_i} [\exp(-E(x))] \\
 &= -\frac{\partial E(x)}{\partial \theta_i} + \mathbb{E}_{p_\theta} \left[\frac{\partial E(x)}{\partial \theta_i} \right] \\
 \mathbb{E}_{p_\theta} \left[-\frac{\partial^2 \log p(x)}{\partial \theta_i \partial \theta_j} \right] &= \mathbb{E}_{p_\theta} \left[\left(\frac{\partial E(x)}{\partial \theta_i} - \mathbb{E}_{p_\theta} \left[\frac{\partial E(x)}{\partial \theta_i} \right] \right) \left(\frac{\partial E(x)}{\partial \theta_j} - \mathbb{E}_{p_\theta} \left[\frac{\partial E(x)}{\partial \theta_j} \right] \right) \right] \\
 &= \mathbb{E}_{p_\theta} \left[\frac{\partial E(x)}{\partial \theta_i} \frac{\partial E(x)}{\partial \theta_j} \right] - \mathbb{E}_{p_\theta} \left[\frac{\partial E(x)}{\partial \theta_i} \right] \mathbb{E}_{p_\theta} \left[\frac{\partial E(x)}{\partial \theta_j} \right]
 \end{aligned}$$



Bibliography

- Alain, G. and Y. Bengio (2013). What regularized auto-encoders learn from the data generating distribution. In *International Conference on Learning Representations (ICLR'2013)*. (Cited on pages [56](#) and [57](#).)
- Amari, S. (1985). Differential geometrical methods in statistics. *Lecture notes in statistics 28*. (Cited on page [98](#).)
- Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation 10*(2), 251–276. (Cited on page [97](#).)
- Amari, S., K. Kurata, and H. Nagaoka (1992). Information geometry of Boltzmann machines. *IEEE Trans. on Neural Networks 3*, 260–271. (Cited on pages [97](#) and [100](#).)
- Arnold, L., A. Auger, N. Hansen, and Y. Ollivier (2011). Information-geometric optimization algorithms: A unifying picture via invariance principles. Technical report, arXiv:1106.3708. (Cited on page [95](#).)
- Bach, F., R. Jenatton, J. Mairal, and G. Obozinski (2011). Structured sparsity through convex optimization. *CoRR abs/1109.2397*. (Cited on page [158](#).)
- Basilevsky, A. (1994). *Statistical Factor Analysis and Related Methods: Theory and Applications*. Wiley. (Cited on page [20](#).)
- Behrens, G., N. Friel, and M. Hurn (2010, October). Tuning Tempered Transitions. *ArXiv e-prints*. (Cited on page [66](#).)
- Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning 2*(1), 1–127. Also published as a book. Now Publishers, 2009. (Cited on pages [22](#), [23](#), [40](#), [62](#), [78](#) and [81](#).)

- Bengio, Y. and A. Courville (2013). Deep learning of representations. In *Handbook on Neural Information Processing*, Volume 49. Springer: Berlin Heidelberg. (Cited on page 22.)
- Bengio, Y., A. Courville, and P. Vincent (2013, August). Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35(8), 1798–1828. (Cited on page 142.)
- Bengio, Y., P. Lamblin, D. Popovici, and H. Larochelle (2007). Greedy layer-wise training of deep networks. In *NIPS'2006*. (Cited on pages 24, 41, 55 and 93.)
- Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5(2), 157–166. Special Issue on Recurrent Neural Networks, March 94. (Cited on pages 13 and 93.)
- Bengio, Y., E. Thibodeau-Laufer, and J. Yosinski (2013). Deep generative stochastic networks trainable by backprop. Technical Report arXiv:1306.1091, Université de Montreal. (Cited on page 57.)
- Bengio, Y., L. Yao, G. Alain, and P. Vincent (2013). Generalized denoising auto-encoders as generative models. In *NIPS'2013*. (Cited on page 57.)
- Bennett, C. (1976, October). Efficient estimation of free energy differences from Monte Carlo data. *Journal of Computational Physics* 22(2), 245–268. (Cited on pages 52, 79 and 80.)
- Bergstra, J., R. Bardenet, Y. Bengio, and B. Kégl (2011). Algorithms for hyperparameter optimization. In *NIPS'2011*. (Cited on page 5.)
- Bergstra, J. and Y. Bengio (2012). Random search for hyper-parameter optimization. *J. Machine Learning Res.* 13, 281–305. (Cited on pages 5 and 131.)
- Bergstra, J., Y. Bengio, and J. Louradour (2011, March). Suitability of V1 energy models for object classification. *Neural Computation* 23(3), 774–790. (Cited on page 118.)
- Bergstra, J., O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio (2010, June). Theano: a CPU and

- GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation. (Cited on page [xv](#).)
- Bergstra, J., G. Desjardins, P. Lamblin, and Y. Bengio (2009). Quadratic polynomials learn better image features. Technical Report 1337, DIRO, Université de Montréal. (Cited on pages [14](#) and [106](#).)
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. (Cited on pages [6](#), [17](#), [18](#) and [164](#).)
- Byrd, R. H., G. M. Chin, W. Neveitt, and J. Nocedal (2011). On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization* *21*(3), 977–995. (Cited on page [95](#).)
- Byrd, R. H., P. Lu, J. Nocedal, and C. Zhu (1995, September). A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.* *16*(5), 1190–1208. (Cited on page [9](#).)
- Carreira-Perpiñan, M. A. and G. E. Hinton (2005). On contrastive divergence learning. In *AISTATS'2005*, pp. 33–40. (Cited on page [96](#).)
- Chapelle, O. and D. Erhan (2011). Improved preconditioner for hessian free optimization. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. (Cited on page [104](#).)
- Cho, K. (2013). Boltzmann machines and denoising autoencoders in image denoising. Technical report, arXiv:1301.3468. (Cited on page [55](#).)
- Cho, K., T. Raiko, and A. Ilin (2010, July). Parallel tempering is efficient for learning restricted Boltzmann machines. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2010)*, Barcelona, Spain. (Cited on pages [35](#), [60](#), [64](#) and [128](#).)
- Coates, A., H. Lee, and A. Y. Ng (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*. (Cited on pages [109](#), [125](#), [126](#), [137](#), [146](#) and [154](#).)

- Coates, A. and A. Y. Ng (2011a). The importance of encoding versus training with sparse coding and vector quantization. In *ICML'2011*. (Cited on pages 109 and 128.)
- Coates, A. and A. Y. Ng (2011b). Selecting receptive fields in deep networks. In *NIPS'2011*. (Cited on pages 109 and 128.)
- Collobert, R. and J. Weston (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pp. 160–167. ACM. (Cited on page 148.)
- Comon, P. (1994). Independent component analysis - a new concept? *Signal Processing* 36, 287–314. (Cited on page 16.)
- Courville, A., J. Bergstra, and Y. Bengio (2011a). A Spike and Slab Restricted Boltzmann Machine. In *Proceedings of The Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'11)*. (Cited on pages 3, 33, 44, 107, 146, 147, 149 and 150.)
- Courville, A., J. Bergstra, and Y. Bengio (2011b, June). Unsupervised models of images by spike-and-slab RBMs. In *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML'11)*. (Cited on pages 22, 44, 107, 119, 146, 150 and 159.)
- Culpepper, B. J., J. Sohl-Dickstein, and B. A. Olshausen (2011). Building a better probabilistic model of images by factorization. In D. N. Metaxas, L. Quan, A. Sanfeliu, and L. J. V. Gool (Eds.), *ICCV*, pp. 2011–2017. IEEE. (Cited on page 143.)
- Dahl, G. E., M. Ranzato, A. Mohamed, and G. E. Hinton (2010). Phone recognition with the mean-covariance restricted Boltzmann machine. In *Advances in Neural Information Processing Systems (NIPS)*. (Cited on page 1.)
- Dauphin, Y. and Y. Bengio (2013). Stochastic ratio matching of rbms for sparse high-dimensional inputs. In *Advances in Neural Information Processing Systems 26 (NIPS'13)*. Nips Foundation (<http://books.nips.cc>). (Cited on page 54.)

- Desjardins, G., A. Courville, and Y. Bengio (2010a). Adaptive parallel tempering for stochastic maximum likelihood learning of rbms. Deep Learning and Unsupervised Feature Learning Workshop, NIPS. (Not cited.)
- Desjardins, G., A. Courville, and Y. Bengio (2010b, May). Tempered Markov chain Monte Carlo for training of restricted Boltzmann machine. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, Volume 9, pp. 145–152. (Cited on pages 35, 36, 37, 38, 59, 64, 68, 79, 82, 89, 128 and 166.)
- Desjardins, G., A. Courville, and Y. Bengio (2011). On tracking the partition function. In *NIPS'2011*. (Not cited.)
- Desjardins, G., A. Courville, and Y. Bengio (2012). Disentangling factors of variation via generative entangling. (Not cited.)
- Desjardins, G., R. Pascanu, A. Courville, and Y. Bengio (2013). Metric-free natural gradient for joint-training of Boltzmann machines. In *International Conference on Learning Representations (ICLR'2013)*. (Not cited.)
- Erhan, D., A. Courville, Y. Bengio, and P. Vincent (2010). Why does unsupervised pre-training help deep learning? In *JMLR W&CP: Proc. AISTATS'2010*, Volume 9, pp. 201–208. (Cited on pages 21 and 41.)
- Freund, Y. and D. Haussler (1992). A fast and exact learning rule for a restricted class of Boltzmann machines. In J. M. S. Hanson and R. Lippmann (Eds.), *Advances in Neural Information Processing Systems 4 (NIPS'91)*, Denver, CO, pp. 912–919. Morgan Kaufmann, San Mateo. (Cited on page 96.)
- Freund, Y. and D. Haussler (1994). Unsupervised learning of distributions on binary vectors using two layer networks. Technical Report UCSC-CRL-94-25, University of California, Santa Cruz. (Cited on page 62.)
- Garrigues, P. and B. Olshausen (2008). Learning horizontal connections in a sparse coding model of natural images. In *NIPS'20*. (Cited on page 110.)
- Giles, C. L. and T. Maxwell (1987). Learning, invariance, and generalization in high-order neural networks. *Applied Optics* 26(23), 4972. (Cited on page 106.)

- Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS'2010*. (Cited on page 13.)
- Glorot, X., A. Bordes, and Y. Bengio (2011). Deep sparse rectifier neural networks. In *AISTATS*. (Cited on page 14.)
- Goodfellow, I., A. Courville, and Y. Bengio (2012). Large-scale feature learning with spike-and-slab sparse coding. In *ICML'2012*. (Cited on page 110.)
- Goodfellow, I., Q. Le, A. Saxe, and A. Ng (2009). Measuring invariances in deep networks. In Y. Bengio, D. Schuurmans, C. Williams, J. Lafferty, and A. Culotta (Eds.), *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pp. 646–654. (Cited on page 22.)
- Goodfellow, I. J., D. Erhan, P.-L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shave-Taylor, M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Romaszko, B. Xu, Z. Chuang, and Y. Bengio (2013). Challenges in representation learning: A report on three machine learning contests. In *International Conference On Neural Information Processing*. (Cited on page 2.)
- Goodfellow, I. J., D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio (2013). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*. (Cited on page xv.)
- Goodfellow, I. J., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio (2013). Maxout networks. In S. Dasgupta and D. McAllester (Eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pp. 1319–1327. ACM. (Cited on pages 14 and 108.)
- Gregor, K. and Y. LeCun (2010). Learning fast approximations of sparse coding. In *ICML'2010*. (Cited on page 58.)
- Gregor, K., A. Szlam, and Y. LeCun (2011). Structured sparse coding via lateral inhibition. In *Advances in Neural Information Processing Systems (NIPS 2011)*, Volume 24. (Cited on page 158.)

- Grimes, D. B. and R. P. Rao (2005, January). Bilinear sparse coding for invariant vision. *Neural computation* 17(1), 47–73. (Cited on pages 47, 48, 142, 156 and 157.)
- Gutmann, M. and J. Hirayama (2011). Bregman divergence as general framework to estimate unnormalized statistical models. In *Proc. Conf. on Uncertainty in Artificial Intelligence (UAI)*. (Cited on page 54.)
- Gutmann, M. and A. Hyvarinen (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10)*. (Cited on page 54.)
- Hinton, G. (2010). A practical guide to training restricted Boltzmann machines. Technical Report 2010-003, University of Toronto. version 1. (Cited on pages 32, 43 and 131.)
- Hinton, G., L. Deng, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury (2012, Nov.). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine* 29(6), 82–97. (Cited on page 1.)
- Hinton, G., A. Krizhevsky, and S. Wang (2011). Transforming auto-encoders. In *ICANN'2011: International Conference on Artificial Neural Networks*. (Cited on pages 142 and 157.)
- Hinton, G. E. (1999). Products of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN)*, Volume 1, Edinburgh, Scotland, pp. 1–6. IEE. (Cited on page 63.)
- Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation* 14, 1771–1800. (Cited on pages 37, 63 and 110.)
- Hinton, G. E., S. Osindero, and Y. Teh (2006). A fast learning algorithm for deep belief nets. *Neural Computation* 18, 1527–1554. (Cited on pages 1, 21, 24, 40, 62, 63, 78, 81, 93, 107 and 134.)

- Hinton, G. E. and R. Salakhutdinov (2006, July). Reducing the dimensionality of data with neural networks. *Science* 313(5786), 504–507. (Cited on pages 55 and 93.)
- Hinton, G. E., B. Sallans, and Z. Ghahramani (1998). A hierarchical community of experts. In *Proceedings of the NATO Advanced Study Institute on Learning in graphical models*, Norwell, MA, USA, pp. 479–494. Kluwer Academic Publishers. (Cited on page 111.)
- Hinton, G. E., T. J. Sejnowski, and D. H. Ackley (1984). Boltzmann machines: Constraint satisfaction networks that learn. Technical Report TR-CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science. (Cited on page 24.)
- Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2012). Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580. (Cited on page 14.)
- Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München. (Cited on page 13.)
- Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 359–366. (Cited on page 12.)
- Hyvärinen, A. (2005). Estimation of non-normalized statistical models using score matching. *Journal of Machine Learning Research* 6, 695–709. (Cited on pages 53, 54 and 63.)
- Hyvärinen, A. (2007). Some extensions of score matching. *Computational Statistics and Data Analysis* 51, 2499–2512. (Cited on pages 54 and 63.)
- Hyvärinen, A. and P. Hoyer (2000). Emergence of phase and shift invariant features by decomposition of natural images into independent feature subspaces. *Neural Computation* 12(7), 1705–1720. (Cited on pages 135, 146 and 147.)
- Hyvärinen, A., J. Karhunen, and E. Oja (2001). *Independent Component Analysis*. Wiley-Interscience. (Cited on page 125.)

- Jarrett, K., K. Kavukcuoglu, M. Ranzato, and Y. LeCun (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pp. 2146–2153. IEEE. (Cited on page 15.)
- Jianzhu Ma, Jian Peng, S. W. and J. Xu (2013). Estimating the partition function of graphical models using langevin importance sampling. In *AISTATS'2013*. (Cited on page 76.)
- Katzgraber, H. G., S. Trebst, D. A. Huse, and M. Troyer (2006). Feedback-optimized parallel tempering monte carlo. *Journal of Statistical Mechanics: Theory and Experiment 2006*(03), P03018. (Cited on pages 60, 62, 64, 66, 67, 68 and 70.)
- Kavukcuoglu, K., M. Ranzato, and Y. LeCun (2008). Fast inference in sparse coding algorithms with applications to object recognition. Technical report, Computational and Biological Learning Lab, Courant Institute, NYU. Tech Report CBL-TR-2008-12-01. (Cited on page 58.)
- Kavukcuoglu, K., M.-A. Ranzato, R. Fergus, and Y. LeCun (2009). Learning invariant features through topographic filter maps. In *CVPR'2009*. (Cited on pages 135, 146 and 147.)
- Kivinen, J. J. and C. K. I. Williams (2012). Multiple texture Boltzmann machines. In *AISTATS'2012*. (Cited on page 110.)
- Kohonen, T. (1996). Emergence of invariant-feature detectors in the adaptive-subspace self-organizing map. *Biological Cybernetics 75*, 281–291. (Cited on pages 135, 146 and 147.)
- Kohonen, T., G. Nemeth, K.-J. Bry, M. Jalanko, and H. Riittinen (1979). Spectral classification of phonemes by learning subspaces. In *ICASSP '79.*, Volume 4, pp. 97 – 100. (Cited on page 146.)
- Koller, D. and N. Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press. (Cited on pages 18, 19 and 21.)
- Krizhevsky, A. (2010). Convolutional deep belief networks on CIFAR-10. Technical report, U. Toronto. (Cited on pages 15, 127, 128 and 138.)

- Krizhevsky, A. and G. Hinton (2009). Learning multiple layers of features from tiny images. Technical report, U. Toronto. (Cited on pages 44 and 124.)
- Krizhevsky, A., I. Sutskever, and G. Hinton (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*. (Cited on pages 1 and 108.)
- Lafferty, J., A. McCallum, and F. C. N. Pereira (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In C. E. Brodley and A. P. Danyluk (Eds.), *Proceedings of the Eighteenth International Conference on Machine Learning (ICML'01)*. Morgan Kaufmann. (Cited on page 21.)
- Larochelle, H., Y. Bengio, and J. Turian (2010, September). Tractable multivariate binary density estimation and the restricted Boltzmann forest. *Neural Computation* 22(9), 2285–2307. (Cited on pages 62 and 81.)
- Larochelle, H. and I. Murray (2011). The Neural Autoregressive Distribution Estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'2011)*, Volume 15 of JMLR: W&CP. (Cited on pages 90 and 91.)
- Le, Q., J. Ngiam, Z. Chen, D. J. hao Chia, P. W. Koh, and A. Ng (2010). Tiled convolutional neural networks. In *NIPS'2010*. (Cited on pages 15, 135, 146 and 147.)
- Le, Q. V., W. Y. Zou, S. Y. Yeung, and A. Y. Ng (2011). Learning hierarchical spatio-temporal features for action recognition with independent subspace analysis. In *CVPR'2011*. (Cited on page 109.)
- LeCun, Y. and Y. Bengio (1994). Word-level training of a handwritten word recognizer based on convolutional neural networks. In IEEE (Ed.), *International Conference on Pattern Recognition (ICPR'94)*, Jerusalem 1994. (Cited on page 14.)
- LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient based learning applied to document recognition. *Proc. IEEE*. (Cited on pages 8, 42, 88, 102 and 131.)
- LeCun, Y., S. Chopra, R. Hadsell, M.-A. Ranzato, and F.-J. Huang (2006). A tutorial on energy-based learning. In G. Bakir, T. Hofman, B. Scholkopf, A. Smola,

- and B. Taskar (Eds.), *Predicting Structured Data*, pp. 191–246. MIT Press. (Cited on page 26.)
- LeCun, Y. and F. Huang (2005). Loss functions for discriminative training of energy-based models. In R. G. Cowell and Z. Ghahramani (Eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS'05)*. (Cited on page 26.)
- LeCun, Y., F.-J. Huang, and L. Bottou (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'04)*, Volume 2, Los Alamitos, CA, USA, pp. 97–104. IEEE Computer Society. (Cited on page 42.)
- LeCun, Y., L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard (1989, November). Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine* 27(11), 41–46. (Cited on page 146.)
- Lee, H., R. Grosse, R. Ranganath, and A. Y. Ng (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. Montreal (Qc), Canada: ACM. (Cited on pages 41 and 93.)
- Lingenheil, M., R. Denschlag, G. Mathias, and P. Tavan (2009). Efficiency of exchange schemes in replica exchange. *Chemical Physics Letters* 478(1-3), 80 – 84. (Cited on pages 37, 66, 68 and 82.)
- Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28(2), 129–137. (Cited on page 16.)
- Lowe, D. (1999). Object recognition from local scale invariant features. In *ICCV'99*. (Cited on page 109.)
- Lücke, J. and A.-S. Sheikh (2011). A closed-form EM algorithm for sparse coding. arXiv:1105.2493. (Cited on page 110.)

- Luo, H., P.-L. Carrier, A. Courville, and Y. Bengio (2013). Texture modeling with convolutional spike-and-slab RBMs and deep extensions. In *AISTATS'2013*. (Cited on page 113.)
- Marinari, E. and G. Parisi (1992). Simulated tempering: A new monte carlo scheme. *EPL (Europhysics Letters)* 19(6), 451. (Cited on page 61.)
- Marlin, B., K. Swersky, B. Chen, and N. de Freitas (2010). Inductive principles for restricted Boltzmann machine learning. In *Proceedings of The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10)*, Volume 9, pp. 509–516. (Cited on pages 38, 53, 64, 65 and 89.)
- Martens, J. (2010, June). Deep learning via Hessian-free optimization. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pp. 735–742. ACM. (Cited on pages 14, 93, 94, 96, 97 and 101.)
- Martens, J. and I. Sutskever (2010). Parallelizable sampling of Markov random fields. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010), JMLR W&CP*, Volume 9. (Cited on page 111.)
- Martens, J. and I. Sutskever (2011). Learning recurrent neural networks with Hessian-free optimization. In *Proc. ICML'2011*. ACM. (Cited on page 93.)
- Memisevic, R. and G. E. Hinton (2007). Unsupervised learning of image transformations. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'07)*. (Cited on page 45.)
- Memisevic, R. and G. E. Hinton (2010, June). Learning to represent spatial transformations with factored higher-order Boltzmann machines. *Neural Computation* 22(6), 1473–1492. (Cited on pages 45, 46, 47, 142 and 157.)
- Mesnil, G., Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, P. Vincent, A. Courville, and J. Bergstra (2011). Unsupervised and transfer learning challenge: a deep learning approach. In *JMLR W&CP: Proc. Unsupervised and Transfer Learning*, Volume 7. (Cited on pages 2 and 22.)

- Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller (1953). Equation of state calculations for fast computing machines. *Journal of Chemical Physics* 21, 1087–1092. (Cited on page 36.)
- Minka, T. (2005). Divergence measures and message passing. *Microsoft Research Cambridge UK Tech Rep MSRTR2005173 72*(TR-2005-173). (Cited on page 50.)
- Minsky, M. L. and S. A. Papert (1969). *Perceptrons*. Cambridge: MIT Press. (Cited on page 106.)
- Mitchell, T. J. and J. J. Beauchamp (1988). Bayesian variable selection in linear regression. *J. Amer. Statistical Assoc.* 83(404), 1023–1032. (Cited on page 110.)
- Mohamed, S., K. Heller, and Z. Ghahramani (2011). Bayesian and l1 approaches to sparse unsupervised learning. arXiv:1106.1157. (Cited on page 110.)
- Montavon, G. and K.-R. Müller (2012). Deep Boltzmann machines and the centering trick. In G. Montavon, G. Orr, and K.-R. Müller (Eds.), *Neural Networks: Tricks of the Trade*, Volume 7700 of *Lecture Notes in Computer Science*, pp. 621–637. (Cited on pages 43, 94, 97, 102 and 104.)
- Montavon, G. and K.-R. Müller (2012). Learning feature hierarchies with centered deep Boltzmann machines. *CoRR abs/1203.4416*. (Cited on pages 102 and 103.)
- Murray, I. and Z. Ghahramani (2004). Bayesian learning in undirected graphical models: Approximate mcmc algorithms. (Cited on page 78.)
- Nadler, W. and U. H. E. Hansmann (2007, Feb). Generalized ensemble and tempering simulations: A unified view. *Phys. Rev. E* 75(2), 026109. (Not cited.)
- Nair, V. and G. E. Hinton (2009). Implicit mixtures of restricted Boltzmann machines. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), *Advances in Neural Information Processing Systems 21 (NIPS'08)*, pp. 1145–1152. (Cited on page 142.)
- Nair, V. and G. E. Hinton (2010). Rectified linear units improve restricted Boltzmann machines. In *Proc. 27th International Conference on Machine Learning*. (Cited on pages 14, 117 and 121.)

- Neal, R. M. (1993). Probabilistic inference using Markov chain Monte-Carlo methods. Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto. (Cited on pages 115 and 123.)
- Neal, R. M. (1994). Sampling from multimodal distributions using tempered transitions. Technical Report 9421, Dept. of Statistics, University of Toronto. (Cited on pages 59 and 66.)
- Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics. Springer. (Cited on page 33.)
- Neal, R. M. (2001, April). Annealed importance sampling. *Statistics and Computing* 11(2), 125–139. (Cited on pages 50, 51, 78, 79, 80, 85 and 103.)
- Neal, R. M. (2005). Estimating ratios of normalizing constants using linked importance sampling. (Cited on pages 53 and 81.)
- Nocedal, J. and S. Wright (2006). *Numerical Optimization*. Springer. (Cited on pages 8 and 9.)
- Olshausen, B., C. Cadieu, J. Culpepper, and D. K. Warland (2007). Bilinear models of natural images. *SPIE Proceedings vol. 6492: Human Vision Electronic Imaging XII, (B.E. Rogowitz, T.N. Pappas, S.J. Daly, Eds.) 6492*. (Cited on pages 47 and 142.)
- Olshausen, B. A. and D. J. Field (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature* 381, 607–609. (Cited on pages 20, 41 and 109.)
- Osadchy, M., Y. LeCun, and M. Miller (2007, May). Synergistic face detection and pose estimation with energy-based models. *Journal of Machine Learning Research* 8, 1197–1215. (Cited on page 142.)
- Paige, C. C. and M. A. Saunders (1975). Solution of Sparse Indefinite Systems of Linear Equations. *SIAM Journal on Numerical Analysis* 12(4), 617–629. (Cited on page 101.)
- Parisi, G. (1988). *Statistical Field Theory*. Redwood City, CA: Addison-Wesley. (Cited on page 18.)

- Pascanu, R. and Y. Bengio (2013). Revisiting natural gradient for deep networks. Technical report, arXiv:1301.3584. (Cited on pages 94 and 95.)
- Pearlmutter, B. (1994). Fast exact multiplication by the Hessian. *Neural Computation* 6(1), 147–160. (Cited on page 101.)
- Pinto, N., D. D. Cox, and J. J. DiCarlo (2008, 01). Why is real-world visual object recognition hard? *PLoS Comput Biol* 4. (Cited on page 22.)
- Raiko, T., H. Valpola, and Y. LeCun (2012). Deep learning made easier by linear transformations in perceptrons. In *AISTATS'2012*. (Cited on pages 44 and 97.)
- Raina, R., A. Battle, H. Lee, B. Packer, and A. Y. Ng (2007). Self-taught learning: transfer learning from unlabeled data. In *ICML'2007*. (Cited on page 22.)
- Ranzato, M. and G. H. Hinton (2010). Modeling pixel means and covariances using factorized third-order Boltzmann machines. In *CVPR'2010*, pp. 2551–2558. (Cited on pages 33, 44, 107, 110, 112, 113, 121, 123, 127, 135, 146, 147 and 151.)
- Ranzato, M., A. Krizhevsky, and G. E. Hinton (2010). Factored 3-way restricted Boltzmann machines for modeling natural images. In *Proceedings of AISTATS 2010*. (Cited on pages 44, 110, 113 and 122.)
- Ranzato, M., V. Mnih, and G. Hinton (2010). Generating more realistic images using gated MRF's. In *NIPS'2010*. (Cited on pages 107, 110, 121, 123, 128 and 151.)
- Ranzato, M., C. Poultney, S. Chopra, and Y. LeCun (2007). Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, and T. Hoffman (Eds.), *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pp. 1137–1144. MIT Press. (Cited on page 24.)
- Ranzato, M., J. Susskind, V. Mnih, and G. Hinton (2011). On deep generative models with applications to recognition. In *CVPR'2011*. (Cited on pages 110 and 162.)
- Richard Socher, Milind Ganjoo, C. D. M. and A. Y. Ng (2013). Zero-shot learning through cross-modal transfer. In *27th Annual Conference on Neural Information Processing Systems (NIPS 2013)*. (Cited on page 2.)

- Rifai, S., Y. Bengio, A. Courville, P. Vincent, and M. Mirza (2012). Disentangling factors of variation for facial expression recognition. In *European Conference on Computer Vision*. (Cited on page 144.)
- Rifai, S., P. Vincent, X. Muller, X. Glorot, and Y. Bengio (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML'2011*. (Cited on pages 56 and 109.)
- Robert, C. P. and G. Casella (1999). *Monte Carlo Statistical Methods*. Springer. (Cited on page 30.)
- Roger Grosse, C. M. and R. Salakhutdinov (2013). Annealing between distributions by averaging moments. In *ICML'2013*. (Cited on page 76.)
- Roweis, S. and L. K. Saul (2000, December). Nonlinear dimensionality reduction by locally linear embedding. *Science* 290(5500), 2323–2326. (Cited on page 16.)
- Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors. *Nature* 323, 533–536. (Cited on page 13.)
- Salakhutdinov, R. (2010a). Learning deep Boltzmann machines using adaptive MCMC. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, Volume 1, pp. 943–950. ACM. (Cited on pages 35, 42, 61, 64, 75, 82 and 167.)
- Salakhutdinov, R. (2010b). Learning in Markov random fields using tempered transitions. In *NIPS'2010*. (Cited on pages 35, 59, 64, 82 and 128.)
- Salakhutdinov, R. and G. Hinton (2009a). Deep Boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, Volume 8. (Cited on pages 41, 42, 43, 94, 96, 97, 100, 102, 103 and 155.)
- Salakhutdinov, R. and G. E. Hinton (2009b). Deep Boltzmann machines. In *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS'09)*, Volume 5, pp. 448–455. (Cited on page 78.)
- Salakhutdinov, R. and I. Murray (2008). On the quantitative analysis of deep belief networks. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings*

- of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, Volume 25, pp. 872–879. ACM. (Cited on pages [51](#), [74](#), [78](#), [89](#) and [103](#).)
- Salakhutdinov, R., J. Tenenbaum, and A. Torralba (2010). One-shot learning with a hierarchical nonparametric bayesian model. Technical report, MIT. MIT Technical Report MIT-CSAIL-TR-2010-052. (Cited on page [2](#).)
- Saul, L. K., T. Jaakkola, and M. I. Jordan (1996). Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research* 4, 61–76. (Cited on page [155](#).)
- Schaul, T., S. Zhang, and Y. LeCun (2012, June). No More Pesky Learning Rates. Technical report, New York University, arxiv 1206.1106. (Cited on page [14](#).)
- Schraudolph, N. N. (1998). Centering neural network gradient factors. In G. B. Orr and K.-R. Muller (Eds.), *Neural Networks: Tricks of the Trade*, pp. 548–548. Springer. (Cited on pages [44](#) and [97](#).)
- Simoncelli, E. P. and B. A. Olshausen (2001). Natural image statistics and neural representation. *Annual Review of Neuroscience* 24, 1193–1216. (Cited on pages [32](#) and [106](#).)
- Snoek, J., H. Larochelle, and R. P. Adams (2012a). Practical bayesian optimization of machine learning algorithms. In *NIPS'2012*. (Cited on page [5](#).)
- Snoek, J., H. Larochelle, and R. P. Adams (2012b). Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*. (Cited on page [127](#).)
- Socher, R., C. Manning, and A. Y. Ng (2011). Parsing natural scenes and natural language with recursive neural networks. In *ICML'2011*. (Cited on page [109](#).)
- Sohn, K., G. Zhou, and H. Lee (2013). Learning and selecting features jointly with point-wise gated Boltzmann machines. In *ICML'2013*. (Cited on page [144](#).)
- Stefano Ermon, Carla Gomes, A. S. and B. Selman (2011). Accelerated adaptive markov chain for partition function computation. In *NIPS'2011*. (Cited on page [75](#).)

- Stefano Ermon, Carla Gomes, A. S. and B. Selman (2012). Density propagation and improved bounds on the partition function. In *NIPS'2012*. (Cited on page 76.)
- Susskind, J., A. Anderson, and G. E. Hinton (2010). The Toronto face dataset. Technical Report UTML TR 2010-001, U. Toronto. (Cited on pages 144, 159 and 161.)
- Sutskever, I., J. Martens, G. Dahl, and G. Hinton (2013). On the importance of initialization and momentum in deep learning. In *ICML*. (Cited on pages 13 and 14.)
- Swersky, K., M. Ranzato, D. Buchman, B. Marlin, and N. de Freitas (2011). On autoencoders and score matching for energy based models. In *ICML'2011*. ACM. (Cited on page 56.)
- Tang, Y., R. Salakhutdinov, and G. Hinton (2013). Tensor analyzers. In *Proceedings of the 30th International Conference on Machine Learning, 2013, Atlanta, USA*. (Cited on page 143.)
- Tang, Y. and I. Sutskever (2011). Data normalization in the learning of restricted boltzmann machines. Technical Report UTML-TR-11-2, Department of Computer Science, University of Toronto. (Cited on page 44.)
- Taylor, G. and G. Hinton (2009). Factored conditional restricted Boltzmann machines for modeling motion style. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*, Montreal, Quebec, Canada, pp. 1025–1032. ACM. (Cited on pages 45, 62 and 81.)
- Tenenbaum, J. B. and W. T. Freeman (2000). Separating style and content with bilinear models. *Neural Computation* 12(6), 1247–1283. (Cited on pages 47, 142, 153, 156 and 157.)
- Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In *ICML'2008*, pp. 1064–1071. (Cited on pages 33, 34, 63, 81, 96, 97, 110, 121 and 126.)
- Tieleman, T. and G. Hinton (2009). Using fast weights to improve persistent contrastive divergence. In L. Bottou and M. Littman (Eds.), *Proceedings of*

- the Twenty-sixth International Conference on Machine Learning (ICML'09)*, pp. 1033–1040. ACM. (Cited on pages [34](#), [35](#), [63](#), [65](#) and [82](#).)
- Tipping, M. E. and C. M. Bishop (1999). Probabilistic principal components analysis. *Journal of the Royal Statistical Society B* 61(3), 611–622. (Cited on page [20](#).)
- Titsias, M. K. and M. Lázaro-Gredilla (2011). Spike and slab variational inference for multi-task and multiple kernel learning. In *NIPS'2011*. (Cited on page [110](#).)
- van der Maaten, L. and G. E. Hinton (2008, November). Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 2579–2605. (Cited on page [16](#).)
- Vasilescu, M. A. O. and D. Terzopoulos (2005). Multilinear independent components analysis. In *CVPR'2005*, Volume 1, pp. 547–553. (Cited on page [157](#).)
- Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Computation* 23(7). (Cited on page [56](#).)
- Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008). Extracting and composing robust features with denoising autoencoders. In *ICML 2008*. (Cited on pages [55](#) and [109](#).)
- Wainwright, M. J. and M. I. Jordan (2008, January). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning* 1(1-2), 1–305. (Cited on page [21](#).)
- Wang, F. and D. P. Landau (2001). Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical Review Letters* 86(10), 2050–2053. (Cited on page [61](#).)
- Wang, H., M. M. Ullah, A. Kläser, I. Laptev, and C. Schmid (2009, September). Evaluation of local spatio-temporal features for action recognition. In *British Machine Vision Conference (BMVC)*, London, UK, pp. 127–127. (Cited on page [146](#).)
- Welling, M., G. E. Hinton, and S. Osindero (2003). Learning sparse topographic representations with products of Student-t distributions. In *NIPS'2002*. (Cited on page [123](#).)

- Welling, M., M. Rosen-Zvi, and G. E. Hinton (2005). Exponential family harmoniums with an application to information retrieval. In L. Saul, Y. Weiss, and L. Bottou (Eds.), *Advances in Neural Information Processing Systems 17 (NIPS'04)*, Volume 17, Cambridge, MA. MIT Press. (Cited on pages 62 and 81.)
- Weston, J., F. Ratle, and R. Collobert (2008). Deep learning via semi-supervised embedding. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, New York, NY, USA, pp. 1168–1175. ACM. (Cited on page 54.)
- Williams, C. K. I. and F. V. Agakov (2002). Products of Gaussians and Probabilistic Minor Component Analysis. *Neural Computation* 14(5), 1169–1182. (Cited on page 124.)
- Wiskott, L. and T. J. Sejnowski (2002). Slow feature analysis: Unsupervised learning of invariances. *Neural Computation* 14(4), 715–770. (Cited on page 22.)
- Younes, L. (1998). On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates. In *Stochastics and Stochastics Models*, pp. 177–228. (Cited on pages 33, 34, 63, 64, 96 and 126.)
- Yu, K. and T. Zhang (2010, June). Improved local coordinate coding using local tangents. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*. ACM. (Cited on pages 131 and 138.)
- Zeiler, M. D. and R. Fergus (2013). Stochastic pooling for regularization of deep convolutional neural networks. In *International Conference on Learning Representations*. (Cited on page 15.)
- Zhou, M., H. Chen, J. W. Paisley, L. Ren, G. Sapiro, and L. Carin (2009). Non-parametric Bayesian dictionary learning for sparse image representations. In *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pp. 2295–2303. (Cited on page 110.)