Het schatten en toepassen van niet-genormaliseerde statistische modellen

The Estimation and Application of Unnormalized Statistical Models

Philémon Brakel

Promotor: prof. dr. ir. B. Schrauwen
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. J. Van Campenhout
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2013 - 2014

UNIVERSITEIT
GENT

ISBN 978-90-8578-716-7
NUR 984
Wettelijk depot: D/2014/10.500/62

Dankwoord

Het schrijven van dit doctoraatsproefschrift is zonder twijfel één van de grootste uitdagingen uit mijn leven geweest. Vermoedelijk was dit dan ook niet gelukt zonder de aanwezigheid en steun van een grote groep mensen waarvan ik een groot deel pas tijdens mijn verblijf in Gent heb leren kennen. Hierbij wil alvast iedereen van harte bedanken en een aantal mensen in het bijzonder.

In de eerste plaats gaat mijn dank natuurlijk uit naar mijn promotor en begeleider Benjamin Schrauwen. Toen ik nog maar pas in het lab aanwezig was nam hij mij gelijk mee naar een belangrijke conferentie om inspiratie op te doen. Ook had hij tal van onderzoeksideeën en heeft hij gezorgd voor een stimulerende werkomgeving. Natuurlijk mag ik zijn bijdragen aan de kwaliteit van mijn onderzoek en dit proefschrift ook niet vergeten. Verder wil ik David en Joni bedanken voor hun begeleiding en advies. Jean-Pierre wil ik bedanken voor het delen van zijn kennis over spraakherkenning. Ook wil ik Stefan Frank (begeleider van mijn masterscriptie) nog bedanken voor de momenten van samenwerking gedurende het begin van mijn doctoraat.

De kwaliteit van een werkomgeving wordt vooral bepaald door je collega's en ik kan enkel zeggen dat ik wat dat betreft zeer veel geluk heb gehad. Sommigen van hen beschouw ik inmiddels als goede vrienden. Zodoende wil ik natuurlijk mijn kantoorgenoot Aäron bedanken voor zijn motiverende enthousiasme, talrijke interessante discussies en het organiseren van de lan-parties. Verder gaat mijn dank met name uit naar Michiel Hermans met wie ik veel heb gediscussieerd, gesport en het glas heb geheven. Ook wil ik Tim bedanken voor zijn bereidheid over anderen hun onderzoek mee te denken en mij elke dinsdag naar het voetbalveld te brengen. Verder gaat mijn dank uit naar Sander voor zijn behulpzaamheid en interessante discussies over deep learning en tal van andere zaken. Francis wil ik bedanken voor zijn advies en kennis over de precieze werking van het universiteitsapparaat.

Pieter ben ik natuurlijk dank verschuldigd omdat hij in mijn jury zit maar ook voor zijn advies en hulpvaardigheid in het algemeen. Verder heb ik ook leuke tijden meegemaakt met mijn collega's Ken, Pieter-Jan, Jonas en Elias. Brahim wil ik bedanken voor leuke gesprekken en goede filmtips. Tot slot wil ik Marnix en Muriel nog bedanken voor hun hulpvaardigheid.

Buiten het werk heb ik veel gehad aan een paar geweldige mensen die ik in Gent heb leren kennen en aan de leuke voetbalsessies op dinsdagavond. Menig weekend heb ik doorgebracht met mijn goede vriend Mouctar. Vaak om samen in stilte te werken maar ook om lange discussies te voeren en de stad te verkennen. Raf wil ik bedanken voor zijn paraatheid om een avondje weg te gaan, aanstekelijke gevoel voor humor en vriendschap in het algemeen. Verder heb ik ook leuke tijden beleefd in Gent met Arda en veel over voetbal geleerd van Matias en Michiel Sanne. De groep mensen die ik van voetballen ken is te groot om iedereen te noemen maar ik wil in ieder geval nog vooral Tom, Paul, Jens, Dieter, Maarten, Seppe, Emiel en Niels bedanken voor alle sportieve maar vooral ook leuke avonden.

Vanuit Nederland ontving ik zeer veel steun van mijn ouders Monique en Richard, die ik wil bedanken voor hun interesse, advies en onvoorwaardelijke steun. Verder ontving ik ook steun van mijn grootouders Fred, Margot en Han. Mijn dank gaat ook uit naar ooms en tantes Fred en Gerlinde, Wouter en Saskia, Roland en mijn oudoom Kees.

During my research I also met various people from a wide variaty of nationalities I need to thank. I want to thank my former office mate Fionntàn for good times, Alexia for some good laughs, Amit for nice discussions and Fabian and Azarakhsh for teaching me about speech recognition. I would also like to thank Antonio for showing me Brussels and his friendship in general, Louis-Charles for great times in both Ghent and Amsterdam and Eric for his advice and starting the football group. I would also like to thank Yoshua Bengio for not only being in my jury but also inviting me to visit the LISA lab in Montreal. I want to thank the members of the LISA lab for making my first visit there both inspiring and enjoyable. Finally, I had some great times with Juan-Pablo thanks to his enthusiasm and sense of humor.

Tot slot wil ik alle leden van mijn examencommissie, die ik nog niet heb genoemd, bedanken voor hun commentaar en raadgevingen die mij instaat hebben gesteld de kwaliteit van mijn proefschrift aanzienlijk te verbeteren.

Philémon Brakel
Gent, 1 augustus 2014

Examencommissie

Prof.	Patrick De Baets, voorzitter Academisch secretaris, Faculteit Ingenieurswetenschappen en Architectuur Universiteit Gent
Prof.	Jean-Pierre Martens, secretaris Vakgroep ELIS, Faculteit Ingenieurswetenschappen en Architectuur Universiteit Gent
Prof.	Benjamin Schrauwen, promotor Vakgroep ELIS, Faculteit Ingenieurswetenschappen en Architectuur Universiteit Gent
Prof.	Willem Waegeman Vakgroep Wiskundige modellering, Statistiek en Bio-informatica, Faculteit Bio-Ingenieurswetenschappen Universiteit Gent
Prof.	Gert De Cooman Vakgroep EESA, Faculteit Ingenieurswetenschappen en Architectuur Universiteit Gent
Prof.	Laurens van der Maaten Vakgroep Intelligente Systemen, Faculteit EWI Technische Universiteit Delft

Prof. Yoshua Bengio
Department of Computer Science and Operations Research
Université de Montréal

Dr. Pieter Buteneers
Vakgroep ELIS, Faculteit Ingenieurswetenschappen en Architectuur
Universiteit Gent

Samenvatting

Lerende systemen

Het herkennen van spraak, afbeeldingen en handgeschreven tekst zijn typische voorbeelden van toepassingen waarin het niet meer mogelijk is om een competitief systeem te ontwerpen dat geen gebruik maakt van data sets met geluidsopnames, digitale fotos of ingescande bezorgadressen. Dit zijn vaak domeinen waarin wij zelf goed presteren zonder in detail weten hoe we dat doen. Deze details zijn echter vaak wel noodzakelijk om een goed computersysteem te ontwerpen dat onze eigen prestatie evenaart of overtreft. *Machinaal Leren* gaat over het ontwikkelen en onderzoeken van systemen die leren om problemen op te lossen door het observeren van data. Dit gebeurd doorgaans met behulp van methoden uit de statistiek. Een belangrijke eigenschap van dergelijke systemen is dat zij de data niet simpelweg opslaan maar belangrijke regelmatigheden en structuur extraheren die het mogelijk maakt om voorspellingen te toen over nieuwe data waar het systeem nog geen toegang toe heeft gehad.

Naarmate de rekenkracht van computers en computernetwerken toeneemt, wordt het ook steeds interessanter lerende systemen te creëeren op basis van geavanceerdere en grotere statistische modellen. Vooral omdat ook de hoeveelheden beschikbare data om modellen op te trainen ook zeer snel toeneemt. Helaas zijn veel soorten statische modellen al vrij snel te complex om nog op wiskundige wijze de exacte waarden van de parameters van deze modellen te kunnen bepalen. Ook het beantwoorden van gangbare vragen over de variabelen die deze modellen beschrijven is vaak niet meer mogelijk. Mijn onderzoek gaat over het schatten en ontwerpen van modellen die te complex zijn om nog gebruik te kunnen maken van exacte methoden. Ik richt mij met name op het verbeteren van benaderingsmethoden en het zoeken naar alternatieven voor standaard methoden die de juiste balans hebben van praktische toepasbaarheid en theoretische correctheid. De focus van de praktischere modellen die ik heb ontwikkeld is het verwerken van tijdsreeksen zoals de opnames van motion capture sensoren en spraak. De specifieke toepassingen van deze modellen zijn het invullen van ontbrekende waarden en het verwijderen van ruis uit spraak opnames teneinde de prestatie van spraakherkenningssystemen te verbeteren.

Efficiëntere MCMC voor RBMs

Veel optimalisatie methoden voor onberekenbare modellen maken gebruik van stochastische benaderingen. Restricted Boltzmann Machines (RBMs) zijn populaire onberekenbare modellen en worden vaak getraind met behulp van een steekproefmethode om de gradiÃ«nten te benaderen die nodig zijn voor optimalisatie van hun parameters. De drijvende veer achter de benadering is een Markov Chain Monte-Carlo (MCMC) methode. MCMC-methoden hebben echter het nadeel dat ze er vaak te lang over doen om een betrouwbare schatting te genereren. Een verbeterde versie van deze methode simuleert meerdere Markovketens in parallel onder verschillende temperaturen teneinde grotere sprongen door de steekproefruimte te kunnen nemen. Voor mijn onderzoek heb ik twee uitbreidingen voor deze verbeterde methode onderzocht die afkomstig zijn uit de physica literatuur. De eerste uitbreiding laat meer onderlinge communicatie tussen de Markovketens toe. De tweede uitbreiding gebruikt de informatie van meerdere ketens bij het schatten van de gradient informatie voor het trainen van de RBMs. De eerste uitbreiding blijkt het toe te laten om een groter aantal ketens te gebruiken. Wanneer beide uitbreidingen worden toegepast blijkt dat dit ook leidt tot betere prestaties voor het trainen van RBMs op een data set van handgeschreven cijfers. Wel zijn deze methoden computationeel gezien nogal intensief en het hangt waarschijnlijk van de specifieke toepassing af hoe waardevol ze zijn in de praktijk.

Alternatieve schatters

De *meest aannemelijke schatter* of *maximum likelihood* estimator (MLE), is een zeer populaire methode voor het schatten van statistische modellen. Theoretisch is het de zuivere schatter met de laagst mogelijke variantie. Helaas is het echter vaak niet mogelijk om de MLE exact toe te passen voor complexere modellen. In deze thesis kijk ik naar alternatieve schatters uit de literatuur die doorgaans theoretisch minder sterke garanties bieden maar vaak practisch beter toepasbaar zijn dan de MLE. Uiteindelijk combineer ik enkele ideeen van bestaande schatters om via het raamwerk van de Bregman divergenties een schatter af te leiden die computationeel efficiënt is en relatief simpel om te implementeren. Ik laat empirisch zien dat de schatter consistent is. Dit betekent onder meer dat de schatter naar de correcte parameter waarden convergeert naarmate de hoeveelheid beschikbare train data groeit. Ook laat ik zien dat de schatter instaat is om kenmerken van afbeeldingen te vinden die lijken op de resultaten van bestaande schatters.

Modellen voor het invullen van ontbrekende waarden

Vaak is het niet mogelijk om complexe modellen te schatten die dienst kunnen doen als generatieve verdeling voor de data. Eerder onderzoek heeft echter laten zien dat een generatief model voor het oplossen van veel problemen niet nodig is en er directere manieren zijn om complexe modellen te trainen die vaak wel haalbaar zijn. Ik breid deze ideeën uit naar andere typen modellen en laat zien dat die gebruikt kunnen worden om ontbrekende waarden in tijdsreeksen in te vullen. Om dit te tonen, beschrijf ik drie typen neurale netwerk modellen die geschikt zijn voor het verwerken van tijdsreeksen en toon hoe deze direct voor het invullen van ontbreken getraind kunnen worden. De modellen halen betere prestaties dan wanneer ze geschat worden met gangbare benaderingsmethoden. Ik heb de modellen toegepast op handgeschreven karakters, motion capture data en sensor data van een robot. Ik laat ook zien dat de modellen nog altijd generatief gebruikt kunnen worden om steekproeven te genereren, wanneer ze om worden gezet in zogenaamde Generatieve Stochastische Netwerken.

Toepassing in spraakverwerking

Om te onderzoeken of de structuur van het meest praktische model om ontbrekende waarden in te vullen ook bruikbaar was voor andere problemen, heb ik experimenten gedaan waarbij de taak was om ruis uit geluidsopnames te verwijderen voor spraakherkenningstoepassingen. Na wat kleine aanpassingen had dit type model de vorm van een Recurrent Neuraal Netwerk dat informatie in twee richtingen tegelijk verwerkt. Ik heb ook een versimpelde versie van dit model onderzocht die computationeel gezien efficiÃ«nter is. De precieze taak was om additieve achtergrond ruis, afkomstig van bijvoorbeeld een metro station of een druk café, te verwijderen uit opnames van gesproken zinnen. Vervolgens werden de opnames aangeboden aan een standaard spraakherkenningsysteem en werd de prestatie gemeten als het aantal fouten dat het systeem maakte op woordniveau. De twee varianten van de model structuur haalden een betere prestatie dan vergelijkbare neurale netwerk modellen uit de literatuur. De modellen werkten echter niet goed wanneer de ruissoort bij het testen erg verschilde van de ruissoorten die gebruikt werden tijdens het trainen.

Summary

Learning systems

The recognition of speech, images, and handwritten text are typical examples of applications for which it is not possible anymore to design competitive systems that don't make use of data sets with audio recordings, digital pictures or scans of delivery addresses. These are often domains in which we perform very well without knowing the full details about the ways in which we do so. However, these details are often necessary to design a good computer system that rivals or surpasses our own performance. Machine Learning deals with the development and investigation of systems that learn to solve problems by observing data. This often involves methods with origins in statistics. An important property of such systems is that they don't simply store the data, but extract important regularities and structural properties that make it possible to do predictions about new data to which the systems never had access before.

As the computational power of computers and computer networks increases, it also becomes more interesting to create learning systems that incorporate larger and more advanced statistical models. Especially because the amounts of available data to train statistical models increases at a high rate as well. Unfortunately, many types of statistical models quickly become too complex to determine the values of their parameters with exact mathematical methods. Answering common questions a statistician might ask about the variables described by these models is often not possible either. My research is about the estimation and design of models that are too complex to allow for the application of analytical methods. I focus especially on the improvement of approximate methods and the search for alternatives to standard methods that provide a good trade-off between practical usefulness and theoretical correctness. The aim of the more practical models I devel-

oped is to process time-series like the recordings of motion capture sensors and speech. The specific applications of these models are the imputation of missing values and the removal of noise from speech recordings to improve the performance of speech recognition systems.

More efficient MCMC for RBMs

Many optimization methods for intractable models make use of stochastic approximations. Restricted Boltzmann Machines (RBMs) are popular intractable models and their training often involves a sampling method to approximate the gradients that are required to optimize their parameters. The backbone of the approximation is a Markov Chain Monte-Carlo (MCMC) method. However, MCMC-methods can often take very long to provide reliable estimates. An improved version of this method simulates multiple Markov chains in parallel that operate under different temperatures to make it possible to take bigger steps through the sample space. For my research, I investigated two extensions of this improved method that were proposed in the physics literature. The first extension allows for more communication among the Markov chains. The second extension uses information about multiple chains to estimate the gradient information for training the RBMs. It turns out that the first extension makes it possible to use a larger number of chains. When both of the extensions are applied, this also leads to better performance for training RBMs on a data set of handwritten digits. However, these methods also increase the computational costs of the learning algorithm and their practical value probably depends on the specific application.

Alternative estimators

The Maximum Likelihood Estimator (MLE), is a very popular method for the estimation of statistical models. Theoretically, it is the unbiased estimator with the lowest possible variance. Unfortunately, it is often not possible to apply the MLE exactly for models that are more complex. In this dissertation, I look at alternative estimators from the literature that typically provide weaker theoretical guarantees but who's application is often more feasible in practice than the MLE. Finally, I combine some ideas derived from existing estimators to use the framework of Bregman divergences to derive an estimator that is computationally efficient and relatively simple to implement. I show empirically that the estimator is consistent. This means, among other things, that the estimator converges to the correct parameter values as the amount of available train data grows. Subsequently, I show that the estimator is able to learn features of images that look similar to those found by more established estimators.

Models for missing-value imputation

Often, it is not possible to estimate complex models to serve as generative distributions of the data. However, earlier work has shown that for many problems a generative model is not required and that there are more direct ways to train complex models that are often more feasible. I extend these ideas to other types of models and show that they can be used to impute missing values in time-series data. To demonstrate this, I describe three types of neural network models that are suitable for processing time-series and show how they can be trained directly for imputing missing values. The models perform better than when they are estimated with common approximate methods. I applied the models to handwritten digits, motion capture data, and sensor recordings of a robot. Finally, I also show that the models can still be used to generate samples when they are used as so-called Generative Stochastic Networks.

Applications in speech processing

To investigate whether the structure of the model that performed best at missing value imputation could also be applied to other types of problems, I did experiments in which the task was to remove noise from audio recordings for speech recognition applications. After some small adjustments, this model took the form of a Recurrent Neural Network that processes information in two directions at the same time. I also investigated a simplified version of this model that is more computationally efficient. The precise task was to remove additive background noise, for example caused by a metro station or a busy pub, from recordings of spoken sentences. After that, the recordings were provided to a standard speech recognition system and the performance was measured as the number of mistakes made by this system on the word-level. The two variants of the model structure performed better than similar neural network models from the literature. However, the models did not perform well when the noise type during test time was very different from the ones that were used during training.

List of Abbreviations

AFE	Advanced Front-End
AIS	Annealed Importance Sampling
ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
BTRNN	Bi-directional Truncated Recurrent Neural Network
CD	Contrastive Divergence
CEBM	Convolutional Energy-Based Model
CRBM	Conditional Restricted Boltzmann Machine
DBN	Deep Belief Network
DDCE	Data Dependent Contrastive Estimation
DFG	Dynamical Factor Graph
DRDAE	Deep Recurrent Denoising Autoencoder
DTBM	Discriminative Temporal Boltzmann Machine
EBM	Energy-Based Model
GPU	Graphical Processing Unit
GSN	Generative Stochastic Network
HMC	Hamiltonian Monte-Carlo
HMM	Hidden Markov Model
LL	Log-Likelihood
MCMC	Markov Chain Monte-Carlo

MFCC	Mel-Frequency Cepstral Coefficients
MLE	Maximum Likelihood Estimation/Estimator
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
MT	Multi-Tempering
MTw	Multi-Tempering with weighted averaging
NADE	Neural Autoregressive Distribution Estimator
NCE	Noise Contrastive Estimation
PBTRNN	Parallel Bi-directional Truncated Recurrent Neural Network
PCA	Principal Component Analysis
PCD	Persistent Contrastive Divergence
PDE	Parzen Density Estimator
PoT	Product of Student-T distribution
PT	Parallel Tempering
RBM	Restricted Boltzmann Machine
REBM	Recurrent Energy-Based Model
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SPLICE	Stereo-based Piecewise Linear Compensation for Environments
WER	Word Error Rate

List of Symbols and Notations

$p(\cdot)$	Probability density function
$P(\cdot)$	Probability mass function
$p_{\text{data}}(\cdot)$	True data distribution
$p_{\text{model}}(\cdot)$	Distribution defined by model
$E(\cdot)$	Energy function
$\mathbb{E}_p[\cdot]$	Expectation over distribution p
$\text{sigm}(\cdot)$	Logistic sigmoid function
$D_{\text{KL}}(q\|\|p)$	Kullback-Leibler divergence between distributions p and q
θ	Set of trainable model parameters
β_i	Inverse temperature of sampling chain i
$\mathbf{H}$	Set of hidden (latent) variables or units
σ^2	Variance
$\mathbf{I}$	Identity matrix
L	Loss function/functional
$\mathbf{x}$	Vector of variables
$\mathbf{x}_n$	Vector of values
$\mathcal{S}$	Set or sequence of data samples
$\mathcal{N}(\cdot\|\mu, \mathbf{\Sigma})$	Normal distribution with mean μ and covariance $\mathbf{\Sigma}$
$\mathcal{H}$	Hamiltonian
T	Total number of time-steps
N	Total number of data points
N_h	Number of hidden units
N_x	Number of visible units
D	Data dimensionality
Z	Partition function

Contents

1 Introduction

The research in this thesis deals with subjects that can all be considered part of the research fields statistics and Machine Learning. More precisely, the research focusses on methods for statistical models that are too complicated to be trained with standard methods. The term 'statistical model' is very general and most of the work is about undirected graphical models, artificial neural networks or combinations of the two. The first half of the thesis is mainly concerned with the estimation/training of generative models for data sets when normalization of these models is not computationally feasible. The second half is about methods that circumvent generative training by optimizing models directly for the specific task they would be used for, merely using a generative model with the same structure as a source of inspiration.

This introduction will briefly introduce the field of Machine Learning and some of the most relevant basics of statistics and probabilities to quickly delve more deeply in the required background knowledge about energy models and statistical estimation. This part of the text also aims to provide enough general information about my work to form a more intuitive understanding of it for those who are not familiar with the field and perhaps less interested in the mathematical details.

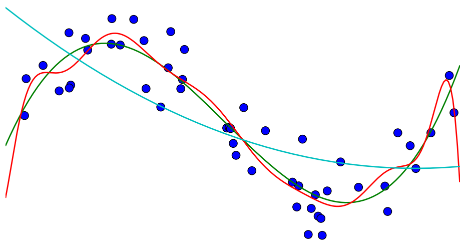
1.1 Machine Learning

Machine Learning is a scientific field that is concerned with computer systems that are able to learn from information. How to design such systems and how to train them to solve certain problems, are major areas of research in this field. The idea behind Machine Learning is to program a computer to learn by itself what would be too complicated to program it to do with explicitly written instructions. Examples of such skills would be language comprehension, face recognition, and motor control. Skills that we humans are such experts in that it may be easy to forget how complicated they actually are. Nowadays, the presence of machine learning algorithms is ubiquitous. Most people own cell phones with applications for speech recognition, visit web pages that employ face recognition, and receive mail packages that have been sorted by machines that are able to read handwritten address information. This presence of systems that learned to do their jobs from data can only be expected to increase in the near future.

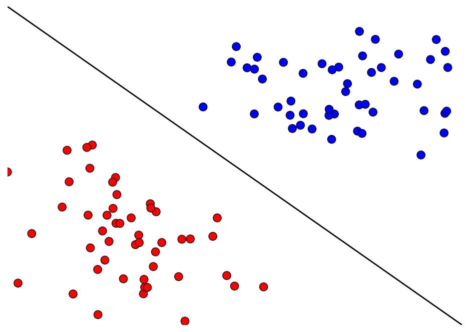
Many of the ideas and algorithms in Machine Learning have their roots in statistics and information theory. An important reason for this is that many types of learning can be seen as a search for regularities in information that is either incomplete or of questionable quality. In other words, one is trying to find relations and structures while dealing with uncertainty. A problem domain that statistics and information theory deal with as well. Noticeable differences between Machine Learning and statistics seem to be the amount of data and the complexity of the models that are used. Research in statistics tends to deal with smaller data sets and simpler models to provide evidence for claims about the underlying processes that generated the data. Machine Learning typically deals with larger data sets and more complex models; interpretation of the modelled underlying processes is often less important. Nonetheless, the research areas of statistics and machine learning display significant overlap and perhaps the most profound difference is in the connotations people have with the two terms.

Many machine learning tasks can be interpreted as the search for functions. Figure 1.1 shows some common problems in statistics and machine learning for which the goal is to find some function that can be used to make predictions on the basis of a limited number of observed points. The examples in Figure 1.1 are regression, classification and density estimation. These tasks will be explained in more detail in Section 1.1.1. In all of the examples, many different functions could describe the available data and the most challenging task is to select that function that is expected to be most useful for describing data points that have not yet been observed. Of course these problems become far more interesting when we start talking about the prediction of stock prices, the classification of pictures of animals and density models that provide a ranking of the most plausible translations of this sentence into some other language.

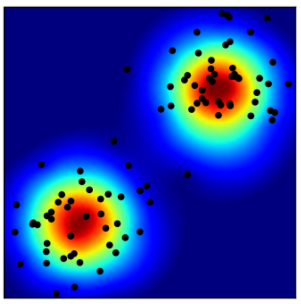
Many models in machine learning and statistics provide families of functions from which a specific function can be selected by tuning certain *parameters*. In such models, optimization methods are used to find those values for the parameters that give the desired function according to some objective of loss functional. For *parametric models*, the number of parameters is fixed



(a) Regression



(b) Classification



(c) Density estimation

Figure 1.1: Many machine learning problems can be interpreted as the search for some function. In Fig. 1.1a the goal is to find function that predicts the hight of the points given some value of the x-axis. In Figure 1.1b the task is to find the line that separates the blue dots from the red dots. In Figure 1.1c the goal is to predict the likelihood of that points will be found in certain regions of the 2 dimensional xy plane, where red indicates a high probability density and blue a low probability density.

regardless of the amount of data that is used to estimate them. For *non-parametric models*, the number of parameters (despite the name they still have them) grows as more data becomes available. The terms ‘variable’ and ‘parameter’ can sometimes lead to some confusion. Unless explicitly stated otherwise, ‘variables’ are typically assumed to be *stochastic* and ‘parameters’ are values that are typically found using optimization to determine the predictions a model makes about these variables.

I will now explain a couple of aspects of machine learning research in detail. First I will talk about the most common types of machine learning problems and elaborate on the types of problems that appear in this thesis. Subsequently, I will talk about common remedies for the overfitting problem that can plague machine learning experiments. Finally I will talk about how the actual learning takes place, or more precisely: how the parameters of many machine learning models are optimized.

1.1.1 Types of Learning Problems

Machine learning is concerned with several specific types of learning problems. The type of learning problem is determined by a couple of factors like the amount of information that is available and the type of information. The most relevant types of learning for this dissertation are *supervised* and *unsupervised* learning.

The most common types of supervised learning tasks are regression and classification. Examples of regression tasks are the prediction of house prices based on their sizes and the prediction of someone’s weight based on variables like age and daily calorie intake. As Figure 1.1a shows, the aim is to find a function that captures a certain relation between two or more variables. In classification problems, the goal is to assign the correct class label to a certain data point. As Figure 1.1b shows, the goal is now to find a line or plane that separates points from different categories (in this case red dots and blue dots). Practical examples of classification are the identification of people based on fingerprints and the categorization of text documents under categories that represent their content (e.g., politics, sports, et cetera). More formally, given some vector of input values $\mathbf{x}$, a supervised system is also presented with some desired target output value t. In the case of *classification*, this value t is the desired class label. For a *regression* problem, t takes a continuous value. The goal for the system is now to learn to predict the value t as well as possible given some input pattern $\mathbf{x}$ that has possibly not been observed before.

Most of the research in this dissertation is about unsupervised learning. Since the annotation of data sets with labels is a task that often has to be performed by humans (and this tends to be a tedious endeavour), the amount of available labelled data is many orders of magnitudes smaller than the amount of unlabelled data. Unsupervised learning is involved with the search for useful structure in data for which no or insufficiently many labels are available. A good example of unsupervised learning is *clustering*, the task of grouping data points together based on some kind of similarity. An unsupervised algorithm may still recognize that news articles about politics have more with each other in common than they have with news articles about sports, without ever being told that these classes exist. Other common examples of unsupervised learning are data compression and dimensionality reduction. From a statistical point of view it can also be interesting to train *generative* models of a data set. These are models that learn to assign a higher probability mass (or density in the case of continuously valued data) to data that is similar to data points they have seen before while assigning a low probability to observations that are very different. Language models, for instance, are trained to assign higher probabilities to sentences that are likely to occur in that language than to sentences that are grammatically incorrect, sound very strange, or are from a different language. Language models are used to select the most plausible sentences from sets of candidates that have been suggested by speech recognition and automatic translation systems.

Several other types of learning problems like *semi-supervised* learning and *reinforcement* learning exist but are not the focus of this work. Since new Machine Learning applications are still being developed, the number of different types of learning problems is also still growing.

1.1.2 Preventing Overfitting

An important goal of most Machine Learning algorithms is to learn properties of data that can be *generalized* to data that has not yet been observed. This necessity to generalize is what separates *learning* from simple *storage*. If the goal was just to remember information, one would simply store it in a database. A system that learns too many specific properties of a data sample without learning much information that is generalizable to unseen data is said to be *overfitting*. This principle is similar to how the memorization of the answers of a math exam doesn't teach one much about the math itself.

If the amount of training data is limited, the complexity of a model can be controlled using various *regularization* methods. Typically, these methods control the complexity of the model by, for example, constraining the magnitude certain parameters can take or the thoroughness of the optimization procedures used to train the model. If the regularization settings are chosen well, a model may perform a bit worse on the train data, but it will make better predictions about new data. Of course it is important to know how to choose the settings for these regularization methods as well as possible based on the data that is available. The most common trick is to pretend that some of the data is new while determining the regularization settings.

A first step, is to divide a dataset into a *train* and *test* set. The model of interest should only be trained on the train set, while its performance should be evaluated on the test set. Simply storing the train data will not suffice to perform well on the test data so the model is now forced to learn more about the structure in the data for solving the task. Unfortunately, this approach does not eliminate the overfitting problem completely. If a researcher keeps on looking for settings of a model that work well on the test data, he or she may still find settings that capture particular characteristics of this sample of test data that don't generalize to new data. A possible solution is to split the data into three parts: a train set on which the model is trained, a *validation* set on which the performance as a function of different settings for the model and its training procedure is evaluated, and a test set on which the final performance is evaluated after it has been decided that the settings will not be altered any further.

When there is not much data available, splitting the data into separate train, validation and test sets may be too wasteful. A popular solution to this problem is *cross-validation*. To perform cross-validation, the train data is split into a number of so called *folds*. The model is now trained as many times as there are folds. For every fold, a model is trained on all the data except for the data in that particular fold. Subsequently, the model's performance is evaluated on that fold and in the end the performance scores of the different models are averaged. To provide an even more reliable result, a separate test set is still used to evaluate the performance of the model after good settings have been identified using cross-validation.

Finally, several alternatives to cross-validation exist that may have certain relative disadvantages and advantages given the amount of available data. Examples of these methods are bootstrapping and Bayesian approaches to model selection.

1.1.3 Optimization

Given a certain model and a task we want the model to learn, finding good parameters for the model is an *optimization* problem. An optimization problem is defined by an objective function (or functional) of both the model parameters and the data, for which we want to find an extreme value (i.e., a maximum or a minimum). For a probabilistic model this may be a value for which the parameters describe the distribution of the data as well as possible. Optimization methods can be classified into those that are *global* and those that are *local*. These terms refer to the fact that an objective can have many optima that can be either local or global. As the name implies, local optima are the best solutions in a local region of the search space. The global optima (or optimum if there is only one) are the subset of local optima that are also the optima of the entire objective function. In other words, they are the *best* local optima. An intuitive analogy for an optimization problem with many optima is the search for the lowest point in a desert landscape with many dunes and valleys. When we are at the bottom of a valley that is surrounded by dunes, we know that we are in a local optimum because we only see higher elevated soil around us, but there is no way of knowing whether this valley is also the deepest one in the whole desert.

That said, the divide between global and local methods is sometimes vague because some of the methods that are commonly considered to be global may still use local information and are not guaranteed to find a global optimum either. I will only discuss optimization methods that don't assume the objective function to be convex.

Global optimization methods often assume that we only have access to the value of the objective function itself and nothing else. The simplest global optimization method is a *brute-force search*. One simply tries as many different values for the parameters as possible and uses those that lead to the best value of the objective function. Assuming that good values of the parameter are located near each other, one can choose a certain location in the parameters space, search in its neighbourhood for better values, and keep searching in directions that look promising. Approaches that are based on this last principle are simulated annealing (Kirkpatrick et al., 1983) and genetic algorithms (Goldberg, 1989). Other examples of global optimization methods are particle swarm optimization (Kennedy and Eberhart, 1995) and Bayesian optimization (Pelikan et al., 1999).

Local optimization methods start at a certain location in the parameter space and try to find the steps that are required to find an optimum as quickly as possible. These methods generally use more knowledge about the objective function, like first and second order derivatives. A widely used local optimization method is gradient descent. This algorithm finds a minimum of a function by evaluation its gradient at a certain location and taking a small step in the negative direction of this gradient. This process is repeated until either the objective is not decreasing anymore or a pre-specified number of steps has been carried out. More advanced methods for local optimization also use second order information. Most of these are based on Newton's method in which a quadratic approximation of the objective function is constructed at a certain location and a step into the direction of the minimum of this quadratic function is taken. Examples of second order optimization methods are quasi-Newton methods like BFGS and truncated Newton or Hessian-free methods (Nocedal and Wright, 2000). A downside of local optimization methods is that if the objective function contains multiple optima, the actual optimum that will be found depends heavily on the location at which the optimization algorithm is initialized.

One of the most widely used optimization methods that falls somewhat in between global and local optimization methods is Stochastic Gradient Descent (SGD). This method is similar to the standard gradient descent algorithm, but instead of computing an exact gradient for a data set, only a small part of the dataset is used at each step. This causes the gradient estimates to be quite noisy but allows for many more updates of the parameters each time the data set is processed. A nice feature of SGD is that the noisy updates may allow the optimization algorithm to jump out of bad local optima to (hopefully) find better optima instead.

1.2 Statistical Models and Energy Functions

After stating that many machine learning and statistical estimation problems can be seen as the searches for functions, it is important to clarify what kind of functions one should think of. In principle *any* function can be used as a statistical model but the more popular families of functions tend to have some properties in common. I will first take a function oriented look at statistical models in general. After that, I will get into the estimation of models using loss functions and in particular the commonly used method of maximum likelihood estimation. The intention is not to provide a thorough introduction to statistics but to review the most relevant theory and terminology. Finally, I will describe some classes of parameterized functions like neural networks and undirected graphical models in more detail.

1.2.1 Properties of Probability Distributions

In the same way that a mathematical model describes the relations between a set of variables, a statistical model describes the relations between stochastic variables. The most commonly used statistical models describe distributions of discrete and continuous variables.[1] Discrete variables can be described by *probability mass functions* that assign a certain probability mass to all possible values of the variables. Continuous variables are described by *probability density functions* that allow the probability masses of certain regions of values to be computed by integrating the density functions over these regions. The integral of a probability density function is called the cumulative distribution function. The set of all possible values a set of random variables can take, is the *sample space.* The term *event* is often used to refer to any possible subset of the sample space. In case the sample space is uncountably infinite, events are typically restricted to be subsets of the sample space that are *Lebesgue measurable* (i.e., subsets for which measures comparable to a volume or length can be computed).

The sum of the masses corresponding to all possible outcomes should be equal to one for a valid probability mass function. All individual mass values should also be positive. Similarly, probability density functions should integrate to one with respect to the sample space.

Given two variables x and y, $p(x|y)$ is the *conditional* distribution of x given y, $p(x, y)$ is the *joint* distribution of x and y. We say a variable has been *observed* when there is no more uncertainty about its value. The distribution $p(x)$ is sometimes referred to as the *marginal* distribution of x to emphasize that it doesn't depend on any other variables. The process of integrating or summing over all values of certain variables to obtain the marginal distribution for one of them is called *marginalization.* Marginalizing y out of the conditional or joint distribution refers to the application of the identity $p(x) = \sum_y p(x, y) = \sum_y p(x|y)p(y)$.

The term *inference* is very general and refers to the answering of questions about the model and the variables it describes. Examples of inference are the determination of the likelihood of a specific state of all the variables or the determination of the variable state for which the probability assigned by the model has the highest value. When inference in a model is said to be *intractable*, this generally means that it is not computationally feasible to compute either a normalized density or to compute the most likely state of the variables in a model.

A term that will be used very often is the *log-likelihood* of a data set. This is the logarithm of the probability mass or density that a model assigns to the available data. For a data set $\mathcal{S} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, the log-likelihood is given by the joint probability of all its N data points $p(\mathbf{x}_1, \dots, \mathbf{x}_N)$. When it is assumed that the points in a data set are independent and sampled from the same distribution (often abbreviated as iid), the log-likelihood of a data

[1]More complex distributions may assign probabilities to functions or other distributions for example.

set is simply the sum of the log likelihoods of the individual data points: $\sum_i \ln p(\mathbf{x}_i)$.

Restricting the discussion to probability *density* functions for a moment, many arbitrary non-negative functions can be forced to integrate to 1 by rescaling the functions by dividing them by their integrals. This obviously requires that the integral over the sample space of such a function converges. A popular way to map an arbitrary function $f(.)$ to a density function is to make it positive by taking the exponential of it and to normalize it by dividing the result by the integral of the exponential of this function:

$$p(x) = \frac{e^{f(x)}}{\int e^{f(x^*)} \, \mathrm{d}x^*}. \tag{1.1}$$

As a matter of fact, many common density functions can be written this way. A good example is the normal distribution given by

$$\mathcal{N}(x, \mu, \sigma) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}, \tag{1.2}$$

where σ is the standard deviation and μ the mean parameter. Equation 1.2 can be written as Equation 1.1 by noting that $f(x) = -\frac{1}{2}\sigma^{-2}(x-\mu)^2$ and $\int \exp(f(x)) \, \mathrm{d}x = (2\pi\sigma^2)^{-\frac{1}{2}}$. Functions that can be transformed to probability density functions in this way are often called *energy functions* because of their origin in statistical physics. In that case, high energy represents low probability so the exponential of the negative energy is used. The next section discusses energy functions in more detail.

1.2.2 Energy Functions and Undirected Graphs

An Energy-Based Model (EBM; LeCun et al., 2006) defines a function $E(\cdot)$ that maps an observation vector $\mathbf{x}_n$ or matrix $\mathbf{X}_n$ to an energy value. This energy value represents a measure of the 'goodness' of a configuration of the input variables. Most models for regression and classification can be seen as energy models. For a classifier, for example, the variables might be the pixels of an image and a binary class label. A well-trained model should assign a lower energy to the image when it is combined with the correct class label than when it is combined with the incorrect one. Inference in energy-based models is done by minimizing the energy function and predictions are defined as

$$\hat{\mathbf{X}}_{(i,j)\in\Omega} \leftarrow \operatorname*{arg\,min}_{\mathbf{X}_{(i,j)\in\Omega}} E(\mathbf{X}; \theta), \tag{1.3}$$

where θ is the set of model parameters. The set Ω contains the indices of the values in $\mathbf{X}$ to perform the minimization over.

Undirected graphical models, often called Markov Random Fields, provide a convenient way for visually depicting and mathematically representing a large class of energy models. Undirected graphical models are defined as graphs in which edges represent symmetric dependencies between variables. An example of an undirected graphical model is given in Figure 1.2. Using the graphical representation of a Markov Random Field it is easy to extract various independence properties of the model. Two vertices are independent if there exists no connection between them or no path of other vertices that correspond to unobserved variables. Figure 1.3 shows an undirected graphical model that is popular in image processing where the edges represent the prior intuition that nearby pixels in an image are strongly correlated with each other.

The joint distribution of a vector of multiple variables $\mathbf{x}$ defined by an undirected graphical model is written as a product of so-called potential functions $\psi_C(\cdot)$ given by:

$$p(\mathbf{x}) = \frac{1}{Z}\prod_C \psi_C(x_C). \tag{1.4}$$

The sets of variables defined by C are so-called *maximal cliques* which are defined as groups with a maximum number of vertices under the constraint that all vertices are connected to each other. The normalization constant Z is called the partition function and defined as

$$Z = \sum_{\mathbf{x}}\prod_C \psi_C(x_C). \tag{1.5}$$

In the case that $\mathbf{x}$ is continuous, the summation in Equation 1.5 is replaced by an integral. To ensure that all the potential functions are positive, they are often defined as the exponentials of energy functions $E(\cdot)$ in the form

$$\psi_C(\mathbf{x}_C) = e^{-E(\mathbf{x}_C)}. \tag{1.6}$$

The fact that the individual potential functions of an undirected graph don't need to be normalized makes them somewhat more flexible than the functions in a directed graphical model which are constrained to be normalized conditional distributions. Theoretically, any energy function can be represented by an undirected graphical model because one can assume that none of the variables are independent.

A particularly well-known type of undirected graphical model is the Bolzmann Machine (Hinton and Sejnowski, 1986). In the original definition of

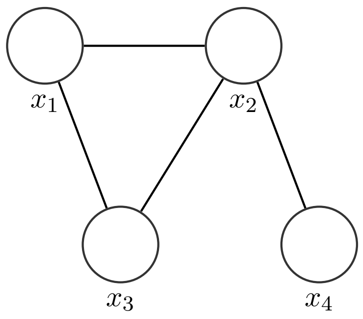


Figure 1.2: Example of a simple undirected graphical model with four variables. The maximal cliques of this graph are $\{x_1, x_2, x_3\}$ and $\{x_2, x_4\}$. The variables x_1 and x_4 are independent when x_2 is observed.

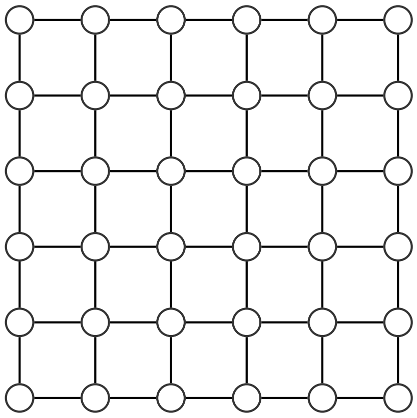


Figure 1.3: A typical grid structured Markov random field suitable for capturing local pixel correlations in images.

the Boltzmann Machine, all the variables in the model are binary and from $\{0, 1\}$. All the potential functions involve either one or two variables (i.e., the maximal cliques are never greater than two). These potential functions are defined as:

$$\psi_i(x_i) = \exp(b_i x_i), \tag{1.7}$$

$$\psi_{ij}(x_i, x_j) = \exp(w_{ij} x_i x_j), \tag{1.8}$$

where b_i is the *bias* parameter of the variable with index i and w_{ij} is a connection weight between the variables with indices i and j. Equivalently, a Boltzmann Machine has the following energy function:

$$E(\mathbf{x}) = -\sum_{i<j} w_{ij} x_i x_j - \sum_i b_i x_i. \tag{1.9}$$

Nowadays, the terms Boltzmann Machine and Markov Random Field display considerable overlap because many models that are called 'Boltzmann Machines' don't model all possible second order interactions, include variables that are not binary or do include higher order interactions between the variables.

To model complex processes, it is common practice to introduce latent or 'hidden' variables that represent interactions between the observed variables. This leads to an energy function of the form $E(\mathbf{X}, \mathbf{H})$, where $\mathbf{H}$ is the set of hidden variables, which need to be marginalized out to obtain the energy value for $\mathbf{X}$. To discriminate between the value $E(\mathbf{X}, \mathbf{H})$ and the sometimes intractable value $E(\mathbf{X})$, I will often refer to the former as the energy and the latter as the *free* energy. This summation (or integration) over hidden variables can be greatly simplified by designing models for which the hidden variables are conditionally independent given an observation. A model that satisfies this independence condition is the Restricted Boltzmann Machine (RBM) (Freund and Haussler, 1994; Hinton, 2002), which is often used to build deep belief networks (Hinton et al., 2006). The energy function of an RBM is given by:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_{i=1}^{N_h} \sum_{j=1}^{N_x} W_{ij} h_i x_j - \sum_{i=1}^{N_h} h_i a_i - \sum_{j=1}^{N_x} x_j b_j, \tag{1.10}$$

where $\mathbf{a}$ is a vector with bias parameters for the visible units.

1.2.3 Energy-Based Learning

Energy-based learning (LeCun et al., 2006) is a rather general framework for describing a broad class of learning algorithms. It can be interesting to represent existing methods in terms of the energy-based learning framework to improve them or to come up with new approaches. A downside of the general nature of this framework is that it can lead to a loss of interpretability. The energy-based learning framework is more general than that of probabilistic graphical models in the sense that a probabilistic interpretation is not necessary; other objectives than the likelihood can be used as well. In practice, one is rarely interested in the likelihood a model assigns to data but just in solving some task like regression or classification.

What defines the energy-based learning framework, is a separation between the *energy* function that defines the 'goodness' of a set of variable values, and the *objective* function that is used to optimize the parameters of the energy function. Any model that defines a proper energy function with a converging integral can be trained as a probabilistic model by setting the objective to be the log likelihood.

1.2.4 Loss Functionals

The goal of a loss functional (a functional is a 'function of a function') is to reward energy functions that assign low energy values at desirable variable values and ideally also punish low energy values at undesirable values. Common objectives for energy-based learning other than the log likelihood are the *generalized perceptron* rule and functions that maximize a margin between the energy values of the desired and undesired configurations of variables (LeCun and Huang, 2005). In general, given some dataset $\mathcal{S}$ of N sample vectors, a loss functional is defined as

$$\mathcal{L}(E, \mathcal{S}) = \frac{1}{N} \sum_{n=1}^{N} L(E(\mathbf{x}_n; \theta)) + R(\theta)), \tag{1.11}$$

where $E(\cdot)$ is an energy function, θ the vector with parameters that determines the shape of that function and $R(\cdot)$ a function of the parameter vector that acts as a regularizer to promote energy functions that are compatible with prior intuitions we may have. The functional $L(\cdot)$ is defined for individual data points and $\mathcal{L}(\cdot)$ can be seen as an estimation of its expectation under the data distribution. In most cases, these functionals can be written as normal functions by making them a function of the parameter vector θ instead of a functional of E. In the remaining discussion I will refer to loss *functionals* that are defined per data point $\mathbf{x}_n$ to make the distinction with energy *functions* more explicit.

Not all combinations of energy functions and loss functionals work well in practice. While energy-based models do not necessarily need to be normalized, a so called *contrastive* term in the loss functional is still needed to prevent the model from learning trivial solutions. To illustrate this problem, I ran a couple of small experiments very similar to those in LeCun et al. (2006).

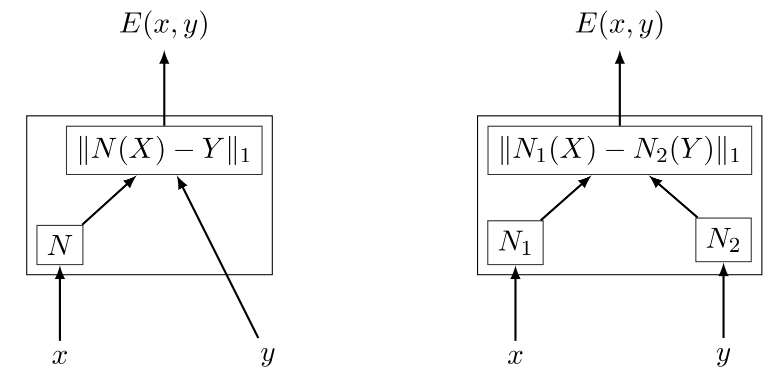
I trained two types of energy models using two different loss functionals: the energy loss and the negative log likelihood loss. The first of these two loss functionals only pulls the energy down for the correct examples, while the second of these also tries to push the energy up for bad answers. The point was to show that certain combinations of loss functionals and models don't work at all.

Let x be from $[-2, 2]$ and the mapping to learn be $t = x^2$. The goal is to train an EBM to learn this mapping. Figure 1.4a shows an EBM for this task in which x is processed by a neural network (see Section 1.3) and the energy is defined as the L^1 norm of the difference between the output of the network and some target value y. I will refer to this model as EBM1. Figure 1.4b shows an EBM that consists of two neural networks; each network processes one of the two variables x or y. The energy for this second EBM is defined as the L^1 norm between the outputs of the two neural networks. Let's refer to this model as EBM2.

The training was done with gradient descent. The EBM1 model had 30 hidden units and the EBM2 two times 6 hidden units.[2] The log likelihood and its gradient were approximated with importance sampling (a method that works reasonably well for problems with such a low dimensionality).

While the two energy models in Figure 1.4 look similar, they behave quite differently depending on the loss functional that is used for training them. Figure 1.5 shows plots of the energy landscapes for different combinations of energy models and loss functionals. As Figure 1.5a shows, the parabola $y = x^2$ is clearly visible as a region of low energy after training EBM1 with the energy loss. The EBM2 model however, does not learn an energy landscape that makes much sense at all, when the energy loss is used to train it (see Figure 1.5b). Finally, the EBM2 does learn a suitable landscape when the negative log likelihood loss is used (see Figure 1.5d). When the EBM1 is trained using the negative log likelihood loss, it learns an energy surface that is very similar to when it was trained using the energy loss. Note that in Figure 1.5d the height of the energy is also strongly related to

[2]The EBM1 could also get good results with far fewer hidden units but this made the optimization procedure more time consuming. Since this is not a performance comparison between models but between loss functionals, the difference between the numbers of parameters between the models is not relevant.



(a) Single network EBM (EBM1).

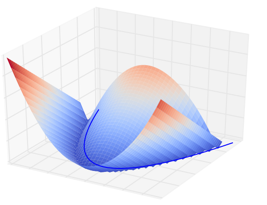
(b) Double network EBM (EBM2).

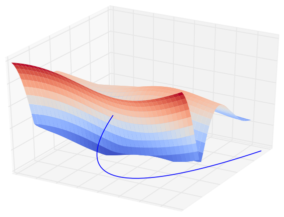
Figure 1.4: Two examples of EBMs for matching a pair of variables.

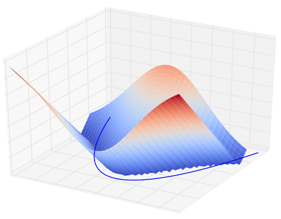
the relative density of the data points (i.e., there are more y values near the origin because x was sampled uniformly).

The EBM1 could be used with both the energy loss and the negative log likelihood loss. However, the EBM2 fails to learn the task when the energy loss is used. This so called *collapse* of the energy function is caused when there is no constraint preventing the energy function to be low *everywhere*. The EBM2 can easily learn to ignore its input such that both of the neural networks always produce the same constant as their output. The contrastive term in the negative log likelihood term prevents this from happening because the model is forced to increase the energy for all locations of the input space that are different from the data. The EBM1 did not suffer from this problem because the subset of input values that can be assigned a low energy is limited in size. As pointed out in LeCun et al. (2006), the energy as a function of y given a certain value of x is constrained to be a V-shaped function with fixed slopes and single minimum. Placing the apex of the V shape on a certain value of y, minimizes the energy there and automatically causes all other possible values of y to be larger.

Another way in which the optimization of a loss functional may fail, is if the margin between correct and incorrect answers becomes zero (LeCun et al., 2006). The generalized perceptron loss pushes the energy down at the correct answer and pushes it up at the answer the model itself would have given. For a binary classification, the gradient of this loss would be zero if the energy scores of both of the possible answers are always identical. While


(a) EBM1 with energy loss.


(b) EBM2 with energy loss.


(c) EBM1 with negative log likelihood.

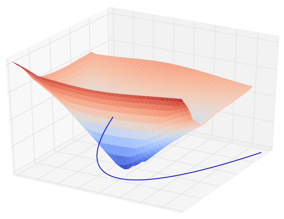

(d) EBM2 with negative log likelihood.

Figure 1.5: Energy landscapes obtained after two types of loss functionals and two architectures. The scale of the z-axis differs wildly between the plots because the negative log likelihood tends to drive the energy to infinity for values that are not in the data.

Table 1.1: Examples of loss functionals for energy-based learning. The symbol $\hat{\mathbf{x}}$ is in this case the so-called *least offending answer*: the value of $\mathbf{x}$ that leads to the lowest energy value but is not identical to $\mathbf{x}_n$. All loss functionals are defined for as single data point.

Loss functional	Equation	Margin
energy	$E(\mathbf{x}_n)$	none
neg. LL	$E(\mathbf{x}_n) + \ln\left(\int e^{-E(\mathbf{x})}\,\mathrm{d}\mathbf{x}\right)$	> 0
perceptron	$E(\mathbf{x}_n) - \min_{\mathbf{x}}(E(\mathbf{x}))$	0
hinge	$\max\left(0, m + E(\mathbf{x}_n) - E(\hat{\mathbf{x}})\right)$	m

this situation is not very likely to happen, it can be completely prevented from happening by ensuring that there is always a finite margin between the energy values of the correct answer and the answer the model is inclined to give. In their tutorial paper, LeCun et al. (2006) give some sufficient conditions for good loss functionals of which a detailed description is beyond the scope of this dissertation. Table 1.1 shows some common loss functionals and the margin they create between desired and undesired answers.

In summary, energy-based learning involves two optimization problems: minimization of the energy with respect to certain variables for inference and minimization of the loss with respect to the parameters. Good loss functionals limit the volume of the variable space that can be assigned very low energy values and care should be taken when loss functionals are used that can result in a collapse of the energy function.

I think that the most important lesson to learn from energy-based learning is that suitable objectives can be constructed using a good loss functional in combination with a suitable family of energy functions. While the framework is so general that there is still plenty of room for constructing bad models, it allows one to combine existing models and optimization methods in new ways.

1.2.5 Statistical Estimators

An estimator is a method or rule for determining a certain quantity on the basis of available data. In practice this involves the prediction of the parameters of a statistical model as a function of the samples in a data set. Important properties of a statistical estimator are its bias, variance and mean squared error. All these properties are defined as expectations of the different estimates with respect to the data distribution. The bias of an estimator is the difference between its expected value and the true parameters, which can be written as $\mathbb{E}_{p(\mathcal{S})}[\hat{\theta}(\mathcal{S})] - \theta$. Similarly, its variance is the expected squared difference from the mean of the estimates and represents how wildly the estimator changes as different data samples are provided. The MSE of an estimator is given by $\mathbb{E}_{p(\mathcal{S})}[(\theta - \hat{\theta}(\mathcal{S}))^2]$, where $\hat{\theta}$ is the estimate as a function of the data sample in its argument. The MSE is related to the bias and the variance in that it is the sum of the square of the bias, the variance, and a third term that represents the noise variance of the data itself. A reduction of either the bias or the variance should lead to a lower MSE but in practice there is often a trade-off such that a lower bias leads to a higher variance and vice versa.

There are a couple of other properties of statistical estimators that are relevant with respect to the work described in this dissertation. These properties are an estimator its *consistency*, whether or not it is *efficient*, and whether it is *asymptotically normal*.

An estimator is consistent if it converges in probability as a function of the sample size. Being consistent also implies that the estimator is asymptotically unbiased. In other words, as the number of data points grows, the estimator becomes more accurate. An estimator is efficient if it is, among the unbiased estimators of a model, the estimator with the lowest variance. Finally, an estimator is asymptotically normal if it is consistent and the estimates (as functions of samples from the data distribution) approach a normal distribution around the correct value of the parameters.

Maximum Likelihood estimation is probably the most popular method for estimating the parameters of statistical models. The Maximum Likelihood Estimator (MLE) is both consistent, efficient and asymptotically normal. All in all, these are good theoretical reasons for using maximum likelihood estimation when this is possible.

Let $p_{\text{model}}(\cdot)$ be some trainable density function that we want to make as similar as possible to some distribution $p_{\text{data}}(\cdot)$ we don't know but can obtain samples from. Formally, the maximum likelihood estimator (MLE) for a certain set of N independent identically distributed data points $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ sampled from $p_{\text{data}}(\cdot)$, is defined as

$$\theta_{\text{MLE}} = \arg\max_{\theta} \sum_{n=1}^{N} \ln p_{\text{model}}(\mathbf{x}_n; \theta), \tag{1.12}$$

where θ is the set of trainable model parameters. When the number of data points goes to infinity, the sum in Equation 1.12 becomes an integral over the sample space weighted by the distribution $p_{\text{data}}(\mathbf{x})$ we try to approximate.

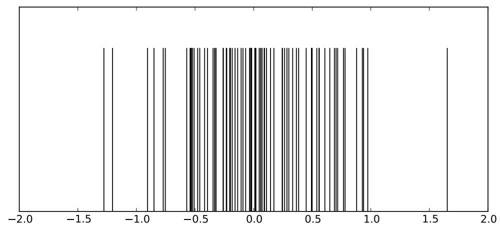


Figure 1.6: Density plot of the empirical data distribution of a collection of points sampled from a Gaussian.

The estimator is now defined as

$$\theta_{\mathrm{MLE}} = \arg\max_{\theta} \int p_{\mathrm{data}}(\mathbf{x}) \ln p_{\mathrm{model}}(\mathbf{x};\theta)\ \mathrm{d}\mathbf{x}. \tag{1.13}$$

The objective which is maximized, is identical to the negative of the so-called *cross entropy* between the two distributions. Minimization of cross entropy is also equivalent to a minimization of the Kullback-Leibler divergence between the distributions.

A weakness of the maximum likelihood estimator is that it is very vulnerable to the over-fitting problem when the number of data points is finite. This is easy to understand considering that the *empirical data distribution*, is actually a set of point masses (see Figure 1.6). Unless the flexibility of the approximating model is constrained in a proper way, MLE will aim to make the density zero nearly everywhere instead of interpolating between the data points.

1.3 Artificial Neural Networks

The term *artificial neural network* (ANN) refers to models in fields as diverse as statistics, cognitive science and computational neuroscience. In machine learning, ANNs are first and foremost nonlinear functions who's shape is determined by a set of trainable parameters. Most of the models I worked with can be seen as neural networks or hybrids of neural architectures and undirected graphical models.

What most ANNs have in common, is that computations are carried out in a largely parallel fashion using computational units called *neurons*. This naming is obviously inspired by observations about the structure of the human brain. In general, learning in an ANN takes place by adapting the values of the *connections* between those neurons in order to learn some

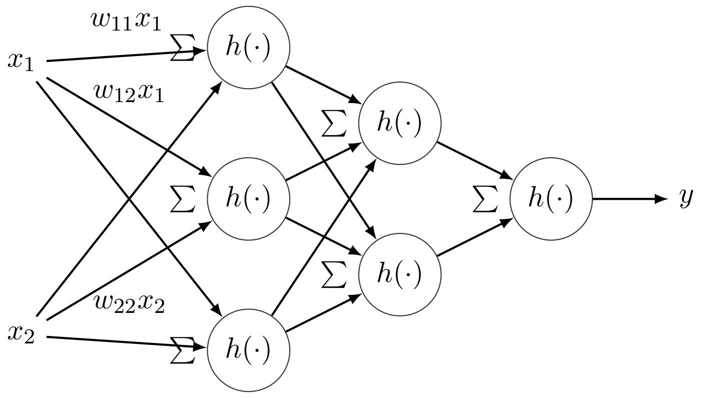


Figure 1.7: Schematic depiction of the computational graph of a feed-forward neural network with two hidden layers and a single output unit.

desired mapping from the input to the output units.

The most common type of ANN is the feed-forward neural network (see Figure 1.7). Feed-forward neural networks implement a certain mapping $f : \mathbf{x} \mapsto y$, where $\mathbf{x}$ is a vector of features and y is some (possibly multivariate) desired output value. The mapping $f(\mathbf{x})$ itself is a weighted sum of functions $\phi_i\mathbf{x}$ given by

$$f(\mathbf{x}) = \sum_i w_i\phi_i(\mathbf{x}) + b, \tag{1.14}$$

where each function $\phi_i(\mathbf{x})$ itself can also be composed of linear combinations of other functions of $\mathbf{x}$ and w_i represents a vector (or matrix) of trainable connection weights. Typically, each linear combination of functions or variable values, is followed by a non-linear transformation $h(\cdot)$. The parameter b represents a bias parameter. In practice, the bias parameter is often included in the weights by appending a 1 to the vector of feature functions. The depth of this recursion determines the number of *layers* of the network. Feed-forward neural networks can model arbitrarily complex functions, given that the number of hidden units is large enough. The algebraical definition of this network would be

$$f(\mathbf{x}) = h\left(\mathbf{u}^{\mathrm{T}} h\left(\mathbf{V} h\left(\mathbf{W}\mathbf{x} + \mathbf{c}\right) + \mathbf{b}\right) + a\right), \tag{1.15}$$

were $\mathbf{u}$, $\mathbf{V}$ and $\mathbf{W}$ are a vector and two matrices that contain the connection weights and h is applied in an element wise fashion. The vectors $\mathbf{b}$ and $\mathbf{c}$ and the scalar a are bias parameters. In a feed-forward neural network there are no connections among units that are in the same layer.

The nonlinear functions $h(\cdot)$ are often chosen to have a sigmoid shape. Common examples of sigmoid *activation functions* are the hyperbolic tangent and the logistic sigmoid given by $\mathrm{sigm}(x) = (1+\exp(-x))^{-1}$. Functions of this type are nearly linear for input values that are close to 0 while very large and very small values are squashed within a certain range.

ANNs are typically trained with gradient based optimization methods. The reason for this is that due to their structure, the gradient of some loss defined as a function of the outputs of an ANN with respect to the parameters can be easily computed using the chain rule. The application of the chain rules to compute the loss gradient for a neural network is called *backpropagation.* This name refers to the way error values that are computed at the output units of the network are sent back through the weights to determine the gradients of the previous layers. This computational process is to a large extent the reverse of the process that was used to convert the input of a network to its output prediction. Let's say that the loss function of interest is the binary cross entropy:

$$L(t, \mathbf{x}) = -t \ln f(\mathbf{x}) - (1-t)\ln(1-f(\mathbf{x})), \tag{1.16}$$

where $f(\mathbf{x})$ is the neural network defined in Equation 1.15 and t is from $\{0, 1\}$. The gradient of the loss with respect to the output before the sigmoid function is applied is now given by

$$\frac{\partial L(t, \mathbf{x})}{\partial o} = f(\mathbf{x}) - t, \tag{1.17}$$

where $o = \mathbf{u}^{\mathrm{T}}\mathrm{sigm}\left(\mathbf{V}\mathrm{sigm}\left(\mathbf{W}\mathbf{x}+\mathbf{c}\right)+\mathbf{b}\right)+a$. This is enough information to compute the gradient of the loss with respect to the weight vector $\mathbf{u}$ as it is given by

$$\nabla_{\mathbf{u}} L(t, \mathbf{x}) = \frac{\partial L(t, \mathbf{x})}{\partial o}\mathbf{h}, \tag{1.18}$$

where $\mathbf{h} = \mathrm{sigm}\left(\mathbf{V}\mathrm{sigm}\left(\mathbf{W}\mathbf{x}+\mathbf{c}\right)+\mathbf{b}\right)$. Now the gradient with respect to hidden unit h_i is given by

A nice property of ANNs is that while many methods use certain hand engineered feature functions to represent the data, ANNs are able to learn those feature functions themselves. Another nice property of ANNs is that it is relatively efficient to apply them to the task of interest after they have been trained. Furthermore, only the values of the weight parameters need to be stored after training. This is in contrast to methods that require the availability of the whole training set during testing and makes ANNs more suitable for processing very large data sets. This follows from the fact that ANNs are *parametric* models that don't grow in size as they are trained on larger data sets. Finally, the neural network framework makes it easy to design large models with many layers and connection structures that capture prior intuitions about the data.

1.3.1 Neural Networks for Time-Series

Modelling time-series can be a challenging task, especially if those time-series are high dimensional and contain nonlinear dependencies. Since two chapters contain neural architectures that are applied to time-series problems I will describe two very common ways of adapting neural networks to time-series data: convolutions and recurrent connections.

Convolutional Networks

A simple way to allow a neural network to process time-series data is by concatenating neighbouring data points in a *sliding window* fashion. For a time-series $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ and a window size of 3, this means that the neural network will be presented with a transformed set of data vectors $\{(\mathbf{x}_1^\mathsf{T}, \mathbf{x}_2^\mathsf{T}, \mathbf{x}_3^\mathsf{T})^\mathsf{T}, (\mathbf{x}_2^\mathsf{T}, \mathbf{x}_3^\mathsf{T}, \mathbf{x}_4)^\mathsf{T}, \ldots, (\mathbf{x}_{T-2}^\mathsf{T}, \mathbf{x}_{T-1}^\mathsf{T}, \mathbf{x}_T^\mathsf{T})^\mathsf{T}\}$, where $(\cdot, \cdot, \cdot)$ signifies the concatenation of three vectors. In other words, the network will still assume that the data points it is processing are independent, but the data has been transformed to take the time related dependencies into account to some extent. This way of presenting the data to a neural network can also be seen as a 1-dimensional *convolution.* Networks of this type are often referred to as *convolutional networks* or *time-delay networks* (Waibel et al., 1989).

This approach is easy to implement and easy to parallelize, but it has a couple of drawbacks. First of all, the size of the context that is taken into account by the network is directly limited by the number of time frames in each window. If there is an important dependency between two frames that are four time steps apart, this relation cannot be captured by a model that uses windows of three frames. Another downside is that the order of the concatenated frames is irrelevant. When a network receives the concatenated vectors from the first three time series as input, it will not assume that $\mathbf{x}_1$ and $\mathbf{x}_2$ are likely to be more related than $\mathbf{x}_1$ and $\mathbf{x}_3$. Last but not least, the number of parameters that are required to process longer dependencies grows linearly with the number of concatenated input frames. This is especially problematic for high dimensional input data like, for example, video recordings.

Recurrent Neural Networks

Unlike standard feed-forward neural networks, recurrent neural networks (RNNs) also contain feedback connections that use activation values from

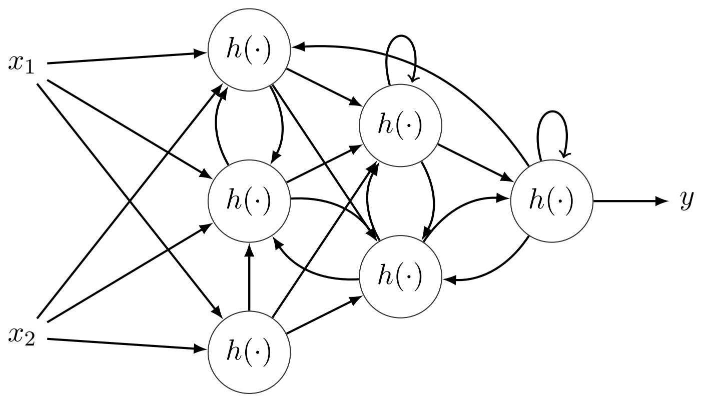


Figure 1.8: RNN

the previous time-step to determine the current values of the units they connect to. This also means that units can be connected to themselves (see Figure 1.8). These feedback connections add a time dimensions to the simulation of the network because the values of the neurons now depend on the values of the neurons at the previous time step.

I will now give a more formal example of a recurrent neural network. Let $\mathbf{X}_n$ be a sequence of d dimensional column vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ that represent features of a d dimensional time-series. A typical recurrent neural network is now defined as

$$\mathbf{y}_t = \mathbf{W}_{\text{out}}\mathbf{h}_t + \mathbf{W}_{\text{dir}}\mathbf{x}_t + \mathbf{b}_{\text{out}} \tag{1.19}$$

$$\mathbf{h}_t = \tanh(\mathbf{W}_{\text{rec}}\mathbf{h}_{t-1} + \mathbf{W}_{\text{in}}\mathbf{x}_t + \mathbf{b}_{\text{rec}}), \tag{1.20}$$

where $\mathbf{W}_{\text{out}}$, $\mathbf{W}_{\text{dir}}$, $\mathbf{W}_{\text{rec}}$ and $\mathbf{W}_{\text{in}}$ are matrices with trainable connection weights and $\mathbf{b}_{\text{out}}$ and $\mathbf{b}_{\text{rec}}$ represent vectors with trainable bias parameters. As Equation 1.20 shows, the value of a hidden state vector $\mathbf{h}_t$ is a function of both the current input vector $\mathbf{x}_t$ and its own value at the previous time step.

A well-trained RNN should be able to use its hidden states to represent a summary of the history of its input that is relevant for solving the task of interest. A nice property of RNNs is that, in contrast to methods that use a fixed number of previous input values, the networks should be able to learn by themselves how far back they should look to find relevant information. However, in practice the length of the time related dependencies RNNs can learn is limited (Bengio et al., 1994). Computing gradients through RNNs requires many multiplications that can drive their value to very small or very large values. Moreover, small changes in the recurrent weights can have a very large effect on the behavior of the system. Recently however, some promising long dependency learning has been demonstrated with RNNs that were trained using second order optimization methods (Martens and Sutskever, 2011). Other ways for learning long term dependencies are with gated memory units as done in long short-term memory networks (Hochreiter and Schmidhuber, 1997) or appropriately adapted reservoir computing methods (Jaeger, 2012).

1.3.2 Deep Learning

The recent success of the so-called *deep learning* approach (Bengio, 2009), has led to a revival of interest in neural network models for solving machine learning problems. Deep neural networks, consisting of many layers of processing units, have claimed state-of-the-art performance on various benchmark problems in areas such as speech recognition (Dahl et al., 2012), handwritten digit recognition (Hinton et al., 2006), and most notably image recognition (Krizhevsky et al., 2012).

Initially, this success was largely attributed to mechanisms for pre-training deep neural networks in a layer by layer fashion to solve the vanishing gradients problem that plagues RNNs as well. However most of the more recent successes seem to be attributable to other factors including new types of activation functions, computing machinery that allows for much larger networks to be used and regularization methods like dropout (Hinton et al., 2012). Nonetheless, the initial surge of interest in unsupervised methods for training neural networks has led to many new ideas for training and developing complex neural network based generative models.

The underlying idea of deep learning is that many real world modelling problems can be solved more efficiently by models that learn a series of transformations of the data that lead to representations that become increasingly abstract. In image processing, for example, the task of object recognition may benefit from a feature extraction process that generates a code that contains the presence and locations of edges in various orientations (i.e., horizontal, vertical and diagonal). At the next level, combinations of these features could be used to construct a coding of slightly more complex shapes like curves and geometrical shapes. At the highest level, features may even code for entire objects. Theoretically, a sufficiently large neural network with a single layer of hidden units can learn any desired mapping but deeper architectures may require fewer processing units to learn the types of mappings that are suitable for real-world phenomena. Recent theoretical work about networks with rectified linear units showed that deep networks can represent more complex functions (more precisely, functions with a larger number of distinct linear regions) than a network with a single large hidden layer (Pascanu et al., 2013b).

As most of the work on deep learning involved neural network architectures, it felt natural to discuss it in a section about neural networks. However, the principles behind deep learning are not limited by the precise architecture that is used and many other types of models have been investigated as multi-layer feature processing mechanisms. An interesting case of this is for example the work on deep kernel functions (Cho and Saul, 2009). Finally, deep learning principles have also successfully been applied to neural time-series architectures like RNNs (Graves et al., 2013; Hermans and Schrauwen, 2013).

In this thesis I don't investigate deep learning itself. However, many of the models I will discuss could potentially be applied in a deep learning setup. Some of the more complex time-series architectures were also inspired by models that come from the deep learning research field.

1.4 Approximate Inference

While for some probabilistic models inference is tractable, there are many types of architectures for which this is not the case. To do inference for such models or to train them, *approximate inference* methods are needed. As the name implies, these methods are not exact and each of them comes with certain advantages and disadvantages.

Since most of the work in this dissertation deals with models for which exact inference is not possible, approximations for inference or learning are a very important part of the research involved and sometimes the direct subject of it. In this section I will talk about some of the most commonly used stochastic and deterministic methods for approximate inference.

1.4.1 Sampling Methods

Often, one is not directly interested in the exact distribution over some of the variables of a probabilistic model but rather in some *expectation* with respect to that distribution. Expectations of variables can be used to make predictions. Other useful quantities that can be written as expectations are the partition functions of undirected graphical models and their gradients.

For a continuous distribution $p(\mathbf{x})$, the expectation of some function $f(\cdot)$

with respect to that distribution is given by

$$\mathbb{E}[f] = \int f(\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{1.21}$$

Assuming that one can obtain samples from the distribution $p(\mathbf{x})$, the expectation of $f(\mathbf{x})$ can be approximated as

$$\frac{1}{L}\sum_{l=1}^{L} f(\tilde{\mathbf{x}}_l), \tag{1.22}$$

where $\{\tilde{\mathbf{x}}_l\}_{l=1}^{L}$ are L samples from $p(\mathbf{x})$.

In many situations however, it is not possible to obtain samples from $p(\mathbf{x})$ directly. Often, it is only possible to evaluate $p(\mathbf{x})$ or a function that is proportional to it. In other situations it is only possible to obtain samples for a subset of the variables $\mathbf{x}$ conditioned on the remaining variables, but not for all the variables at the same time. Factors like these determine which sampling method is the best choice.

A common method for approximating expectations when it is not possible to sample from the distribution of interest directly, is importance sampling. In importance sampling, samples are taken from a so-called *proposal distribution* that is as similar as possible to the distribution to approximate. To compensate for the fact that one is sampling from the wrong distribution, the samples are re-weighted by the ratio of the densities they have under the target and proposal distributions q. The formula used in importance sampling is very similar to Equation 1.22 and given by:

$$\frac{1}{L}\sum_{l=1}^{L} f(\tilde{\mathbf{x}}_l)\gamma_l, \tag{1.23}$$

where $\gamma_l = p(\tilde{\mathbf{x}}_l)/q(\tilde{\mathbf{x}}_l)$ are so-called *importance weights* If p or q are not normalized, the importance weights are often rescaled to sum to 1 over all samples and the multiplication with $\frac{1}{L}$ is removed. This corresponds to a rescaling with the ratio of the partition functions of the distribution as approximated by the importance sampling algorithm itself.

Unfortunately, importance sampling is very vulnerable to the curse of dimensionality. As the dimensionality of the sample space increases, it becomes exponentially more likely that samples from the proposal distribution have a very low density under the target distribution if the proposal distribution is just slightly higher in variance. This means that the computed average will be dominated by a very small number of samples with far higher weights than the others. In other words: the number of samples required for a reliable approximation grows very fast as the dimensionality grows.

If the goal is not just to compute expectations but to actually generate samples from a distribution for which this is not directly possible, a similar algorithm called *rejection sampling* can be used. In this algorithm, a sample $\tilde{\mathbf{x}}$ is again generated from some proposal distribution $q(\mathbf{x})$. Subsequently, a uniform sample $\tilde{u}$ from the interval $(0, 1)$ is generated. Now it should hold, that the proposal distribution times a certain constant α is always larger than $p(\mathbf{x})$. Now the sample $\tilde{\mathbf{x}}$ is accepted if $\tilde{u} < \frac{p(\tilde{\mathbf{x}})}{\alpha q(\tilde{\mathbf{x}})}$. As is the case with importance sampling, rejection sampling is also very sensitive to differences between the proposal and targets distributions when the dimensionality of the data is high.

When the dimensionality of the sample space is high and it is not possible to sample from the target distribution directly, it may be a better idea to resort to Markov chain Monte Carlo (MCMC) methods. MCMC methods suffer less from a higher dimensionality of the sample space and some variants can also be used when it is not even possible to compute values of the target density (or a function that is proportional to it). However, unlike many other Monte Carlo methods, the samples from MCMC methods are not independent of each other anymore because each sample directly depends on the previous state of the chain. The usability of a MCMC method is therefore directly related to the rate at which consecutive samples become less dependent on each other. MCMC methods will be discussed in more detail in Section 2.2.

1.4.2 Approximate Variational Inference

Variational approximate inference methods are a deterministic alternative for sampling methods that can be computationally more efficient. The idea behind variational inference is to turn the inference problem into an optimization problem. An objective is defined in terms of a functional that reaches an optimum when a function is found that is identical to the target distribution $p(\mathbf{x})$. One way to obtain an *approximate* method is by constraining the set of functions one can choose from to solve the optimization problem. In practice, this is often done by selecting a set of probability density functions over $\mathbf{x}$ that are still tractable. Examples of approximate variational inference methods are loopy belief propagation, mean-field and expectation propagation (Wainwright and Jordan, 2008; Minka, 2001). In this section I will only introduce the variational mean-field method.

A popular variational approximate inference method that has origins in statistical physics, is variational *mean-field* (Wainwright and Jordan, 2008). In variational mean-field, the objective to be optimized is defined to be the

Kullback-Leibler divergence between the distribution to approximate $p(\mathbf{x})$ and some tractable approximating distribution $q(\mathbf{x})$, given by

$$D_{\mathrm{KL}}(q(\mathbf{x})||p(\mathbf{x})) = \int q(\mathbf{x}) \ln \left(\frac{q(\mathbf{x})}{p(\mathbf{x})} \right) \, \mathrm{d}\mathbf{x}. \tag{1.24}$$

As one can see, Equation 1.24 is zero if and only if the distributions $p(\mathbf{x})$ and $q(\mathbf{x})$ are identical.

When a model contains latent variables $\mathbf{h}$ and approximations of $p(\mathbf{x})$ and $p(\mathbf{h}|\mathbf{x})$ are sought, an approximate distribution $q(\mathbf{h}|\mathbf{x})$ can be defined to provide the following lower bound on the log likelihood:

$$\begin{aligned} \ln p(\mathbf{x}) &\geq \ln p(\mathbf{x}) - D_{\mathrm{KL}}(q(\mathbf{h}|\mathbf{x})||p(\mathbf{h}|\mathbf{x})) \\ &= \mathbb{E}_q(\ln p(\mathbf{x}, \mathbf{h})) + \mathcal{H}(q), \end{aligned} \tag{1.25}$$

where $\mathbb{E}_q$ is the expectations with respect to q and $\mathcal{H}$ is the entropy functional.

It follows from the above, that given an approximating distribution q that is flexible enough, the bound can be made identical to the true log-likelihood. Obviously, finding the parameters of such a distribution is as difficult as inference in the original distribution and in practice a form of q has to be chosen that is tractable. The most common choice for q is a fully factorized distribution. If $p(\mathbf{h}|\mathbf{x})$ is for instance a joint distribution over a set of binary variables, the choice for $q(\mathbf{h}|\mathbf{x})$ may be chosen to be a product of one dimensional Bernoulli distributions given by $q(\mathbf{h}|\mathbf{x}) = \prod_{i=1}^{N_h} \mu_i^{h_i}(1-\mu_i)^{1-h_i}$. Expectations with respect to this distribution are easy to compute. When a fully factorized distribution q is used, the method is referred to as *naive mean-field*.

The next step is to optimize the bound in Equation 1.25 with respect to the parameters of the approximating distribution q. In general, the bound can be differentiated with respect to these parameters. Setting the gradient to zero leads to a coordinate ascent algorithm that will converge to a local optimum.

Variational mean-field has a couple of weaknesses. First of all, the quality of the approximation depends heavily on the choice of the family of approximating distributions. If the distribution we want to approximate is for example multi-modal, a unimodal approximating distribution will, at best, only be able to identify one of the modes of this distributions. Another problem is that mean-field methods can be very sensitive to the initialization of the parameters of the approximating distribution and that they might get stuck in bad local optima. Nonetheless, variational mean-field can be a useful tool when sampling methods are too expensive and the number of dependencies in a graphical model is too large to efficiently perform other approximate inference methods.

1.5 Evaluating Intractable Models

While the evaluation of models and methods in Machine Learning is already a task that requires great care, evaluating intractable models is even more problematic. When approximations are required to train a model, this often means that the evaluation of this model will need to be approximated as well. This can make it difficult to assess the quality of new models and ideas in a systematic way. It is often a good idea to evaluate a model with a couple of different methods to ensure that the result is not caused by the particular evaluation method itself.

There are roughly two ways to evaluate intractable generative models that have been trained using some approximation to maximum likelihood learning:

1. The model is evaluated on some other task than density estimation for which it will also be used in practice. Examples of such tasks are denoising, feature extraction, classification and the initialization of the parameters of a supervised model.

2. The likelihood (or some measure that correlates with it) is approximated with, for example, sampling methods.

The first of these two methods has the advantage that the model is evaluated on a task for which it will be used but this is obviously only possible if that task itself is tractable. From a scientific perspective, this method also gives no true insight into the quality of an approximate maximum likelihood learning method. The second method can provide more insight in approximate likelihood learning methods, but should never be trusted completely because it is based on approximations itself. The quality of these approximations could be sensitive to the specific training methods or models that are being compared.

Another option that is sometimes sensible, is to make the model tractable by reducing its complexity. It can sometimes be insightful to see how a downsized version of a model behaves to get a better intuition about the behavior of the intractable model as well. However, it should always be kept in mind that statistical models and their learning algorithms may behave very differently when the model changes in size.

I will now discuss some approximate quantitative methods for the evaluation of intractable models. I will also discuss some qualitative methods for evaluation that can be useful when the approximate methods prove to be very unreliable.

1.5.1 Annealed Importance Sampling

A relatively popular method for estimating normalization constants of intractable models is the Annealed Importance Sampling algorithm (AIS; Neal, 1998a). This algorithm can be seen as a combination between MCMC and importance sampling, in which the ratio between the normalization constants of two distributions is estimated. As the name implies, AIS is based on importance sampling (see Section 1.4.1). The idea is that one of these two distributions is tractable and that samples are taken using MCMC from a series of models that interpolate between the tractable and the intractable model; this gradual settling to the model of interest is the *annealing* process. In practice, this is done by setting the tractable model equal to a version of the intractable model where some of the parameters are set to zero. The interpolation is done by setting the energy equal to a linear combination of the two models where a scaling parameter determines their relative weights. Running an MCMC chain until it is annealed only provides a single importance weighted estimate of the ratio of normalization constants. In practice, many chains have to be simulated in parallel to obtain a reliable estimate. AIS is suitable for evaluating RBMs (Salakhutdinov, 2008) for which MCMC sampling is relatively efficient.

A nice property of the AIS method, is that the MCMC method is likely to mix quite well during the early stages of annealing. Another nice property is that no information is thrown away and that all intermediate samples of the MCMC method contribute to the final estimate.

To apply AIS, one has to be able to evaluate the energy function with respect to the samples produced by the MCMC method. This is not always possible. Another downside is that AIS tends to be too optimistic when the MCMC sampler doesn't mix well. Since an estimate of the partition function is sought, this estimate will be too low if the Markov chain misses a mode of high probability density. This is especially undesirable when one wants to compare the likelihood scores with those of other models. Moreover, the estimates of the sampler may be sensitive to the shape of the distribution.

1.5.2 Evaluating Samples

Another branch of sampling methods aims to define an evaluation metric that is a lower bound on the log likelihood (Breuleux et al., 2011; Bengio and Yao, 2013). The idea is to use samples from the intractable model to construct a density estimator. The evaluation metric is now the log likelihood this density estimator assigns to the test set. Estimates can be expected to be relatively conservative if the sampling method is sub-optimal. A property that is desirable when comparing intractable models with the true log likelihood of a tractable baseline.

A commonly used non-parametric density estimation method for real-valued data, is the Parzen window kernel density estimator with Gaussian kernel. Let $\mathbf{Y}$ be a set of K samples $\{\mathbf{y}_1, \dots, \mathbf{y}_K\}$, taken from the model of interest. The density of some point $\mathbf{x}$ is now defined as

$$\frac{1}{K}\sum_{k=1}^{K}\mathcal{N}(\mathbf{x}|\mathbf{y}_k, \sigma^2\mathbf{I}), \tag{1.26}$$

where $\mathcal{N}(\cdot|\mathbf{y}_k, \sigma)$ is a Gaussian probability density function with mean $\mathbf{y}_k$ and variance σ^2. This density estimator has the nice property that as the number of samples goes to infinity, it converges to the true distribution (in this case the distribution of the intractable model). The parameter σ determines the so-called kernel width and needs to be determined with a method like cross-validation.

In practice, this method suffers from large variance. In other words, many samples are needed to obtain reliable estimates. This is unfortunate because the evaluation of the kernel density estimator becomes more computationally demanding as the number of samples grows. The fact that the parameter σ needs to be determined using some validation set or cross-validation is a nuisance as well.

Recently, an interesting variant of this approach was proposed in which no kernel bandwidth parameter has to be determined and lower variance can be expected. The method only works for models in which Gibbs sampling provides samples of latent variables and for which the density over the visible variables is tractable given that the latent variables are known. Say some collection of latent variable samples $\{\mathbf{h}_1, \dots, \mathbf{h}_K\}$ has been obtained and $p(\mathbf{x}|\mathbf{h})$ is easy to compute, the density estimator is now defined as

$$\frac{1}{K}\sum_{k=1}^{K}p(\mathbf{x}|\mathbf{h}_k). \tag{1.27}$$

This density estimator is also a more natural choice for discrete probability distributions for which a Parzen density estimator doesn't make sense. See Bengio and Yao (2013) for a proof that the expected value of this estimator is a lower bound on the true test log likelihood.

A nice property of evaluation metrics based on non-parametric density estimators is that it is never required to compute the energy of the model. This makes these methods particularly useful for models for which it is straightforward to obtain samples from both the latent and visible variables using MCMC but not to compute the actual free energy of a sample.

A downside of these methods is still that many samples will be needed to reduce the variance of the estimator and that this directly leads to a more computationally demanding density estimator. In practice, this may lead one to discard many of samples that are too similar. When MCMC is used, this may mean that only samples are used that are far away from each other in the chain. Recall that this is not an issue with an algorithm like AIS.

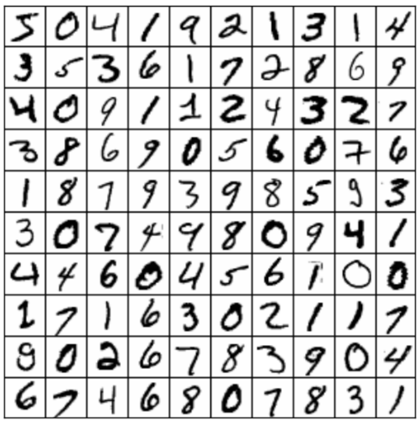
1.5.3 Qualitative Evaluation Methods

In addition to the evaluation methods described so far, it can be useful to choose a modelling problem that makes a qualitative evaluation of the model possible. This can be done by choosing a task that is visually interpretable by nature or by choosing a task that is simple enough that certain values of the parameters or the energy landscape can be visualized (i.e., can be shown in three or fewer dimensions).

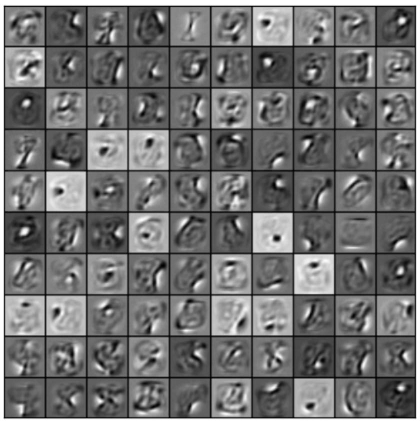
If, for example, one chooses to use easily recognizable image data like handwritten digits, a visual representation of the parameters of a model may be easier to interpret. This situation is most obvious when some of the parameters are applied to the data in a way that can be seen as a comparison operation. Figure 1.9a show 100 samples from the MNIST handwritten digits database and Figure 1.9b shows some of the weight parameters of a Restricted Boltzmann Machine that was trained on this dataset. While the features don't look like the data, they do appear to describe some of the more common shapes of the images. Doing a visual feature inspection of this kind is not as straightforward for features learned from audio data or network traffic.

Another way to asses the quality of a model visually, is by using a simple two dimensional data set so that the values of the model can be evaluated over a coarse grid of x and y coordinates. Again, this allows the structure learned by the model to be compared with the original data or the distribution the data was sampled from.

While qualitative assessments should not be used to draw strong conclusions about the behavior of a model or training method, they can provide insight into its weaknesses and strengths. When inference is difficult, one can still see if a training method is able to find features that are varied and somewhat similar to those found by more established methods. As with downsized models however, it is important to keep in mind that results on low dimensional data sets do not necessarily generalize to higher dimensions at all.



(a) MNIST digits.



(b) Features from an RBM trained on MNIST.

Figure 1.9: Some data sets make a visual inspection of the parameters of a model possible.

1.6 Research Contributions

In this section I will outline the main research contributions of this dissertation. The goal of the research was to find new or better ways to make use of complex statistical models for which normalization is not possible. The contributions involve both new training methods for existing model architectures and the investigation of new architectures.

The first part of the thesis is about improved or new methods for training generative models. Concerning stochastic methods for optimizing unnormalized models, I showed that two tricks from the statistical physics literature can be used to improve the training of restricted Boltzmann machines using parallel tempering. Subsequently, I showed how ideas from a couple of different statistical estimators and training techniques can be combined to construct a new estimator that displays potential as an alternative for existing methods.

The second part is more about the design of new model structures to solve problems inspired by methods for inference for generative models. I showed that backpropagation of error gradients through the steps that are normally taken to perform inference can be used to train various complex neural networks to perform missing-value imputation in time series data. I also showed that these models can be used to generate samples when they are used as so-called Generative Stochastic Networks. Finally I showed that a recurrent neural network architecture inspired by the way mean field inference iterations are carried out can be used to perform speech denoising. The systems I proposed outperformed previous neural network approaches to the task in question and some of the most common denoising techniques.

1.7 List of Publications

Journal Publications

1. Philémon Brakel, Dirk Stroobandt and Benjamin Schrauwen. Training energy-based models for time-series imputation. *Journal of Machine Learning Research*, Vol. 14, pp. 2771-2797 (2013)

2. David Verstraeten, Benjamin Schrauwen, Sander Dieleman, Philémon Brakel, Pieter Buteneers and Dejan Pecevski. Oger: modular learning architectures for large-scale sequential processing. *Journal of Machine Learning Research*, Vol. 13, pp. 2995-2998 (2012)

Conference Publications

1. Philémon Brakel, Dirk Stroobandt and Benjamin Schrauwen. Bidirectional Truncated Recurrent Neural Networks for Efficient Speech Denoising. *INTERSPEECH*, (2013)
2. Philémon Brakel and Benjamin Schrauwen. Energy-based temporal neural networks for imputing missing values. *ICONIP*, (2012)
3. Philémon Brakel, Sander Dieleman and Benjamin Schrauwen. Training Restricted Boltzmann Machines with multi-tempering: harnessing parallelization. *Proceedings of the 22nd international conference on Artificial Neural Networks and Machine Learning (ICANN)*, (2012)
4. Sander Dieleman, Philémon Brakel and Benjamin Schrauwen. Audio-based music classification with a pretrained convolutional network. *12th International Society for Music Information Retrieval Conference (ISMIR-2011)*, (2011)
5. Philémon Brakel, Stefan Frank. Strong systematicity in sentence processing by simple recurrent neural networks. *CogSci*, (2009)

2 Monte Carlo Methods for Training RBMs

In this chapter I will discuss optimization methods for Restricted Boltzmann Machines (RBMs) that are based on Monte Carlo methods. I will also describe some of my own research in this area and touch on some ideas for future work. While the discussion focusses on RBMs, one should keep in mind that most of the methods can be used for many other types of models as well; RBMs just happen to be the model of choice in many publications about stochastic optimization methods for unsupervised learning. To be more specific, this chapter is about optimization methods in which sampling methods are used to approximate the gradient of the normalization constant of a model. This distinguishes the methods in this chapter from other types of stochastic optimization like genetic algorithms, vanilla stochastic gradient descent and some of the methods that will be discussed in Chapter 3. The main contribution is a more efficient sampling method for gradient estimation. This should potentially lead to higher quality models when other sampling methods are not accurate enough.

2.1 Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs; Hinton, 2002; Freund and Haussler, 1994) are bipartite graphs in which a layer of latent (hidden) variables is connected to a layer of observed (visible) variables. As is shown in Figure 2.1, there are no connections among the variables that are in the same layer. This means that the hidden variables are independent when the visible variables are observed and vice versa.

Being part of the family of undirected graphical models with latent variables, RBMs are typically trained in an unsupervised way to explain a certain

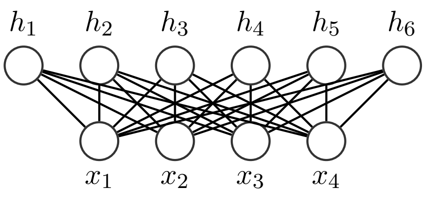


Figure 2.1: Graphical model structure of a Restricted Boltzmann Machine. The bipartite structure makes the hidden variables (top) independent given that the visible variables (bottom) have been observed and vice versa.

set of data samples as well as possible. Typically, this does not mean that the RBM is used to provide actual likelihood values for tasks like model comparison, but merely as an unsupervised feature extractor like PCA.

While RBMs have been around for more than a decade, they became a popular subject of research after it was shown that they can be used to find good initializations for supervised neural networks with multiple layers of processing units known as Deep Belief Networks (DBNs; Hinton et al., 2006). When it was shown that DBNs were successful at tasks including handwritten character recognition (Hinton et al., 2006), speech recognition (Dahl et al., 2012), information retrieval (Salakhutdinov and Hinton, 2009a), 3D object recognition (Nair and Hinton, 2009) and natural language processing (Deselaers et al., 2009), RBMs gained even more popularity. Finally, RBMs have also been used to perform some tasks directly without being part of some bigger system. RBMs have for example been used to perform matrix factorization for building recommender systems (Salakhutdinov et al., 2007). They have also been trained discriminatively to perform classification (Larochelle and Bengio, 2008).

The energy function that captures the structure of an RBM in which both the visible variables $\mathbf{x}$ and the hidden variables $\mathbf{h}$ are binary, is given by

$$E(\mathbf{x}, \mathbf{h}) = -\sum_{i=1}^{N_h} \sum_{j=1}^{N_x} W_{ij} h_i x_j - \sum_{i=1}^{N_h} h_i a_i - \sum_{j=1}^{N_x} x_j b_j, \tag{2.1}$$

where N_h and N_x are the number of hidden and the number of visible variables, respectively. The symbols W, a and b denote trainable weight and bias parameters. The likelihood of a configuration of the variables $\mathbf{x}$ and $\mathbf{h}$ is now given by the following Gibbs distribution:

$$p(\mathbf{x}, \mathbf{h}) = \frac{e^{-E(\mathbf{x},\mathbf{h})}}{\sum_{\mathbf{x}'} \sum_{\mathbf{h}'} e^{-E(\mathbf{x}',\mathbf{h}')}}. \tag{2.2}$$

The hidden units can be summed out analytically to define the energy as a function of the visible values only, often referred to as the *free energy*, given by

$$E(\mathbf{x}) = -\ln\left(\sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}\right), \tag{2.3}$$

$$E(\mathbf{x}) = -\ln\left(\sum_{h_1,\ldots,h_{N_h}} e^{\sum_{i=1}^{N_h}\sum_{j=1}^{N_x} h_i W_{ij} x_j + a_i + x_j b_j}\right), \tag{2.4}$$

$$= -\ln\left(\sum_{h_1,\ldots,h_{N_h}} \prod_{i=1}^{N_h} e^{\sum_{j=1}^{N_x} h_i W_{ij} x_j + a_i}\right) - \sum_{j=1}^{N_x} x_j b_j, \tag{2.5}$$

$$= -\ln\left(\prod_{i=1}^{N_h} \sum_{h_i\in\{0,1\}} e^{\sum_{j=1}^{N_x} h_i W_{ij} x_j + a_i}\right) - \sum_{j=1}^{N_x} x_j b_j, \tag{2.6}$$

$$= -\sum_{i=1}^{N_h} \ln\left(1 + e^{\sum_{j=1}^{N_x} W_{ij} x_j + a_i}\right) - \sum_{j=1}^{N_x} x_j b_j. \tag{2.7}$$

Due to the independence properties of the RBM model, it is possible to compute the probabilities of the visible variables conditioned on the hidden variables using

$$p(x_j = 1|\mathbf{h}) = \frac{e^{\sum_{i=1}^{N_h} h_i W_{ij} + a_i h_i}}{e^{\sum_{i=1}^{N_h} a_i h_i} + e^{\sum_{i=1}^{N_h} h_i W_{ij} + a_i h_i}}, \tag{2.8}$$

$$= \frac{e^{\sum_{i=1}^{N_h} h_i W_{ij}}}{1 + e^{\sum_{i=1}^{N_h} h_i W_{ij}}}, \tag{2.9}$$

$$= \operatorname{sigm}\left(\sum_{i=1}^{N_h} h_i W_{ij}\right), \tag{2.10}$$

where $\operatorname{sigm}(\cdot)$ is the logistic sigmoid function given by $\operatorname{sigm}(x) = (1 + \exp(-x))^{-1}$. Due to the independence of the variables, the conditional probability of the whole vector $\mathbf{x}$ is $p(\mathbf{x}|\mathbf{h}) = \prod_j \operatorname{sigm}(\sum_i w_{ij} h_i + b_i)$. Similarly, the probability of the hidden variables given the visible variables is given by $p(\mathbf{h}|\mathbf{x}) = \prod_i \operatorname{sigm}(\sum_j w_{ij} x_j + a_j)$.

Like any other energy-based model, when an RBM is trained to optimize the likelihood of a certain data set, it will aim to assign low energy to values of the visible variables that are similar to values it has seen before and high energy to values that are very different. The hidden variables allow the RBM to model correlations among the visible variables.

An RBM is not inherently limited to binary variables. The conditional distributions of the visible variables given the hidden variables and the hidden variables given the visible variables can be other univariate distributions depending on the type of data one wants to model and the type of hidden variables one expects to be most suitable for modelling the structure of the data. Especially RBMs with Gaussian visible units and binary hidden units have been the subject of research (Wang et al., 2012; Cho et al., 2011a). Furthermore, it was found that RBMs with hidden units that approximate a rectified linear mapping work better than binary units for modelling image data (Nair and Hinton, 2010). There are also models that include second order interactions between the visible units conditioned on the hidden units as well like the mean covariance RBM (Ranzato and Hinton, 2010) and the spike and slab RBM (Courville et al., 2011). Note that these are not RBMs in the strict sense because they do not[1] follow the required conditional independence properties of the visible and hidden units.

After they have been trained, RBMs can be used as feature extractors by treating the expected value of $p(\mathbf{h}|\mathbf{x})$ as a mapping from the data points to features that (hopefully) capture important structural properties. When RBMs are used to initialize Deep Belief Networks, the encoded data is used to train another RBM that tries to find regularities in the extracted features themselves. This process can be repeated to obtain a set of parameters that can be used to initialize a Deep Belief Network of many layers of hidden units.

An energy function can be used to define a Gibbs probability distribution of the form $p(\mathbf{x};\theta) = \sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h};\theta)}/Z(\theta)$, where $\theta = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$ and $Z(\theta)$ is the partition function which is given by $Z(\theta) = \sum_{\mathbf{h},\mathbf{x}} e^{-E(\mathbf{x},\mathbf{h};\theta)}$. The gradient of the logarithm of this likelihood function is given by

$$\frac{\partial \ln p(\mathbf{x};\theta)}{\partial\theta} = -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{x};\theta)\frac{\partial E(\mathbf{x},\mathbf{h};\theta)}{\partial\theta} + \sum_{\mathbf{x},\mathbf{h}} p(\mathbf{x},\mathbf{h};\theta)\frac{\partial E(\mathbf{x},\mathbf{h};\theta)}{\partial\theta}\ , \tag{2.11}$$

where θ is an element in the set of parameters $\{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$. The first term of this gradient can be evaluated analytically in RBMs but the second term needs to be approximated. This second term is the gradient of the partition function and will be referred to as the *model expectation.* When the hidden units are summed out analytically to obtain the free energy, the likelihood

[1]The spike and slab RBM is actually a borderline case because the visible units are independent when both the values of the spike variables and the slab variables are known.

gradient becomes

$$\frac{\partial \ln p(\mathbf{x};\theta)}{\partial \theta} = -\frac{\partial E(\mathbf{x};\theta)}{\partial \theta} + \sum_{\mathbf{x}} p(\mathbf{x};\theta)\frac{\partial E(\mathbf{x};\theta)}{\partial \theta}. \tag{2.12}$$

For a binary RBM, the gradient $\frac{\partial E(\mathbf{x};\theta)}{\partial \theta}$ can be computed using the following identities:

$$\frac{\partial E(\mathbf{x};\theta)}{\partial \mathbf{W}} = -\mathrm{sigm}(\mathbf{W}\mathbf{x}+\mathbf{a})\mathbf{x}^{\mathsf{T}}, \tag{2.13}$$

$$\frac{\partial E(\mathbf{x};\theta)}{\partial \mathbf{a}} = -\mathrm{sigm}(\mathbf{W}\mathbf{x}+\mathbf{a}), \tag{2.14}$$

$$\frac{\partial E(\mathbf{x};\theta)}{\partial \mathbf{b}} = \mathbf{x}. \tag{2.15}$$

When $\mathbf{h}$ is known, it is also possible to compute the gradient of $E(\mathbf{x},\mathbf{h};\theta)$ by replacing $-\mathrm{sigm}(\mathbf{W}\mathbf{x}+\mathbf{a})$ with $\mathbf{h}$ in Equations 2.13 and 2.14.

2.2 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods are based on simulating a Markov chain that has the desired distribution as its *equilibrium distribution*. This means that given an infinite number of steps in the Markov chain, the samples obtained after each step of the chain will be distributed according to some target distribution $p(\mathbf{x})$, regardless of the starting location of the chain. Many Markov chains with this property are defined as random walks but more sophisticated methods exist as well (Berg and Billoire, 2004). Intuitively one can think of most MCMC methods as a random walk that spends more time in regions where the probability density is higher. The samples can be used to approximate expectations of $p(\mathbf{x})$.

One of the most important reasons for the popularity of MCMC methods is that they are not as severely affected by the curse of dimensionality as many other sampling methods. If the regions of higher density lie on some narrow low dimensional manifold of the sample space, sampling these areas with some global method is incredibly unlikely to happen. MCMC methods on the other hand, are able to explore such a manifold by taking small steps without wandering away from it too often. Intuitively, this is somewhat comparable to playing the game battleship and trying locations near the last hit as opposed to random guessing.

Because every step in a Markov chain depends on the previous step in the chain, the samples obtained from an MCMC algorithm are not independent. Two successive samples are likely to be highly correlated. For this reason, it is often necessary to run the chain for many steps to obtain a sample that is practically independent of the starting location of the chain. Another problem with MCMC methods is that if the probability distribution contains multiple modes that are separated from each other by regions of very low probability density, it can take very long for a Markov chain to leave one of the modes and visit another one. The efficiency at which MCMC chains obtain independent samples and visit all important regions of the sample space, is called the *mixing rate* of the Markov chain.

One MCMC method of which many other MCMC methods are specific cases, is the Metropolis-Hastings algorithm (Hastings, 1970), which is a generalization of the original Metropolis algorithm. Let $g(\mathbf{x} \mapsto \hat{\mathbf{x}})$ be the probability density function of the event that a Markov chain jumps from some location $\mathbf{x}$ to a new location $\hat{\mathbf{x}}$. This density function defines a so-called *proposal distribution*. The Metropolis-Hastings algorithm now starts by taking a sample from this distribution. This new sample will be accepted with a probability that is given by

$$p_{\text{accept}} = \min\left(1, \frac{p(\hat{\mathbf{x}})g(\hat{\mathbf{x}} \mapsto \mathbf{x})}{p(\mathbf{x})g(\mathbf{x} \mapsto \hat{\mathbf{x}})}\right). \tag{2.16}$$

If the example is accepted, the new location of the chain becomes $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}$ is added to the set of collected samples. If the sample is rejected, the chain stays at location $\mathbf{x}$ and an extra observation of sample $\mathbf{x}$ is added to the set of collected samples. Note that because the distribution $p(\cdot)$ appears in both the denominator and numerator of Equation 2.16, its normalization constant cancels so it only has to be proportional to the desired density function; this makes the method applicable to unnormalized models like RBMs. The size of the steps proposed by the proposal distribution plays a large role in the behavior of the Metropolis-Hastings algorithm. If the steps are very small, many of proposed samples will be accepted but the chain will also behave very much like a random walk and stay in the same regions for a long time. If the steps are very big, the chain has a bigger chance to move to some distinct mode of the distribution but most of the samples will be rejected so it will take very long to collect enough samples for computing reliable expectations.

The acceptance criterion in Equation 2.16 ensures that the Markov chain satisfies the *detailed balance* condition. Detailed balance is a sufficient condition to ensure that a Markov chain has $p(\mathbf{x})$ as its equilibrium distribution. When the proposal distribution is symmetric so that $g(\hat{\mathbf{x}} \mapsto \mathbf{x}) = g(\mathbf{x} \mapsto \hat{\mathbf{x}})$, its occurrences in the numerator and denominator cancel out and Equation 2.16 becomes the acceptance criterion of the Metropolis algorithm.

If the proposal distribution is defined as the distribution of a subset of the variables in $\mathbf{x}$ conditioned on the remaining variables, one obtains the Gibbs sampling algorithm. In this case, the fraction in Equation 2.16 is always equal to 1, so all samples will get accepted. This sampling method of course requires that for every variable of $\mathbf{x}$, we can obtain a sample given that we know the values of the remaining variables. A nice property of the Gibbs sampler is that one doesn't need to be able to evaluate a function that is proportional to the target density.

2.2.1 MCMC for Optimization

Now that it is clear how MCMC methods can be used to approximate expectations with respect to their equilibrium distribution, the next question is how to combine those methods with optimization techniques. To answer this question, it may be useful to have a look again at the gradient of one of the most common objectives in statistical estimation: the log likelihood. The gradient of the log likelihood of a Gibbs distribution $p(\mathbf{x};\theta) = \exp(-E(\mathbf{x};\theta))/\int \exp(-E(\mathbf{x};\theta))\ \mathrm{d}\mathbf{x}$, with respect to the parameters θ, is given by

$$\nabla \ln p(\mathbf{x};\theta) = \mathbb{E}_{p_{\mathrm{model}}}\left[\nabla E(\mathbf{x};\theta)\right] - \mathbb{E}_{p_{\mathrm{data}}}\left[\nabla E(\mathbf{x};\theta)\right]. \tag{2.17}$$

The stochastic approximation will need to estimate the first term, often referred to as the *model expectation.* The second term is often called the *data expectation* and can be computed using a sample average. Since the model distribution and the expectations that depend on it are a function of the parameters of the model, the model expectation should be re-estimated every time the parameters are updated during training. Typically, the gradient estimates will be used to do parameter updates using the stochastic gradient descent method.

A naive algorithm for optimizing Equation 2.17 would first compute the data expectation, run a sampling method (like MCMC) until enough samples have been created to obtain a reliable estimate of the second expectation, update the parameters using this gradient estimate and repeat the whole procedure to obtain the next parameter update. This method is shown in Algorithm 1 for the specific case that the Metropolis algorithm with spherical Gaussian proposal distribution is used for sampling and gradient descent for updating the parameters of the model. Obviously, this puts the sampling procedure at the inner loop of the optimization algorithm. For most models and sampling algorithms this method would be far too slow to be of any practical use. However, it serves as the inspiration of many of the more practical methods I will discuss below.

Algorithm 1 Naive stochastic optimization algorithm using gradient descent and the Metropolis algorithm with spherical Gaussian proposal distribution.

```
Define data set $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and model $E(\mathbf{x}; \theta)$
Initialize parameter vector $\theta_0$, learning rate $\lambda$
**for** $l = 1$ **to** $L$ **do**
  Initialize chain state variables $\hat{\mathbf{x}}_0$
  {Generate $K$ samples}
  **for** $k = 1$ **to** $K$ **do**
    Sample $\hat{\mathbf{y}} \sim \mathcal{N}(\hat{\mathbf{y}} | \hat{\mathbf{x}}_{k-1}, \sigma^2 \mathbf{I})$
    Sample $r \sim \text{Uniform}(r|0, 1)$
    **if** $r < \exp(E(\hat{\mathbf{x}}_{k-1}; \theta_{l-1}) - E(\hat{\mathbf{y}}; \theta_{l-1})$ **then**
      $\hat{\mathbf{x}}_k \leftarrow \hat{\mathbf{y}}$
    **else**
      $\hat{\mathbf{x}}_k \leftarrow \mathbf{x}_{k-1}$
    **end if**
  **end for**
  {Update parameters}
  $\theta_l = \theta_{l-1} + \lambda \left( \frac{1}{K} \sum_{k=1}^{K} \nabla E(\hat{\mathbf{x}}_k; \theta_{l-1}) - \frac{1}{N} \sum_{n=1}^{N} \nabla E(\mathbf{x}_n; \theta_{l-1}) \right)$
**end for**
**return** $\theta_L$
```

Contrastive Divergence

A simple stochastic method that has been shown to work quite well for certain models is the Contrastive Divergence (CD; Hinton, 2002) algorithm. During training, an MCMC sampler is initialized at a data point and run for a just a couple of iterations. The data-based gradient is estimated using small batches of data points. The last sample of the chain is used to replace the intractable model expectation. In other words, CD training is a form of stochastic gradient descent where the model expectation is estimated using a Markov Chain that is only run for a small number of steps rather than until a reasonably independent sample has been obtained. The algorithm can be obtained from Algorithm 1 by setting the number of sampling steps K to some low value like 5 and using only a single randomly chosen data point $\mathbf{x}_n$ to estimate the model expectation in the parameter update. It is also important that $\hat{\mathbf{x}}_0$ is initialized at a data point (typically the same data point $\mathbf{x}_n$ as used to compute the data expectation). Keep in mind that the Metropolis algorithm is typically replaced with a Gibbs sampler or if necessary the Hamiltonian Monte Carlo algorithm from Section 2.2.2. It is also common practice to generate batches of samples in parallel and also use batches of data points to estimate the data expectations but the benefit of these alterations of the algorithm are very application specific. This strategy assumes that many of the low energy configurations, that contribute most to the model expectation, can be found near the data itself.

CD is easy to implement and computationally efficient but it has some drawbacks. While it is likely that much of the probability mass is located near the data, it is also very likely that there are many other valleys of low energy that will not be reached by a Markov Chain that is only run for a couple of steps. Furthermore, the algorithm does not optimize the true maximum likelihood criterion and may even diverge (Fischer and Igel, 2010; Sutskever and Tieleman, 2010). That said, there is an interesting connection between single step CD and the pseudo-likelihood (Hyvärinen, 2007a).

Persistent Contrastive Divergence

In Persistent Contrastive Divergence learning (PCD; Tieleman, 2008), a Markov chain is updated after every parameter update during training and used to provide samples that approximate the model expectation. The difference with normal CD is that the chain is not reset at a data point after every update, but keeps on running so it can find low energy regions that are far away from the data. Given infinite training time and a learning rate that goes to zero, this algorithm optimizes the true likelihood. The PCD algorithm can be obtained from Algorithm 1 by removing the outer loop, placing the parameter update inside the sampling loop and again use only a single data point for estimating the data expectation. This optimization method is shown in Algorithm 2. For brevity, I replaced the Metropolis sampling step with a more general MCMC transition distribution written as $p_{\mathrm{mcmc}}(\hat{\mathbf{x}}_k|\hat{\mathbf{x}}_{k-1})$

Algorithm 2 Persistent Contrastive Divergence with stochastic gradient descent

```
Define data set $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and model $E(\mathbf{x}; \theta)$
Initialize parameter vector $\theta_0$, learning rate $\lambda$
Initialize chain state variables $\hat{\mathbf{x}}_0$
for $k = 1$ to $K$ do
    {Generate $K$ samples}
    Sample $\mathbf{x}_k$ uniformly from the data
    Sample $\hat{\mathbf{x}}_\mathbf{k} \sim p_{\mathrm{mcmc}}(\hat{\mathbf{x}}_k|\hat{\mathbf{x}}_{k-1})$
    $\theta_k = \theta_{k-1} + \lambda \left(E(\hat{\mathbf{x}}_k; \theta_{k-1}) - \nabla E(\mathbf{x}_k; \theta_{k-1})\right)$
end for
return $\theta_K$
```

Typically, a far lower learning rate should be used for PCD than is common for standard CD learning. This makes sense since the gradient is based on the difference between the energy gradient at the data and the sampled values. In the case of a persistent chain these gradients should be expected to differ more than for samples obtained using CD.

2.2.2 Improving the Mixing Rate

In practice, training time is not infinite and it is important to consider the behavior of the PCD algorithm after a finite amount of time. As training progresses and the model parameters get larger, the energy landscape tends to become more rough. This is a direct result of the shape of the empirical data distribution we try to model. Ultimately, the empirical data distribution is a collection of point masses, a distribution in which many MCMC methods wouldn't work at all. This increased 'roughness' will decrease the size of the steps the chain takes and increase the time that the chain gets stuck in local modes of the distribution.

On the one hand, the parameter updates will improve the mixing rate during training by increasing the energy of regions that are visited very often by the Markov Chain. On the other hand, if the mixing becomes too slow, the estimated likelihood gradient may become so different from the true likelihood gradient that the training algorithm may fail to converge at all

(Fischer and Igel, 2010).

A common method from the literature for improving mixing times is *over-relaxation* (Neal, 1998b). The idea behind over-relaxation is to decorrelate the individual variables because the step size of the Gibbs sampler will be determined by the lowest value of the covariance of the variables. This is done by doing a Gibbs update and subsequently correlating the new variable value negatively with its previous value. Say a specific component x_i of a variable vector $\mathbf{x}$ would be updated using $p(x_i|\mathbf{x}_{/i})$ which has a Gaussian distribution with mean μ_i and variance σ_i^2, the over-relaxed update for the new $\hat{x}_i$ would be

$$\hat{x}_i = \mu_i + \alpha(x_i - \mu_i) + \sigma_i(1 - \alpha^2)^{\frac{1}{2}}\eta, \tag{2.18}$$

where η is a normally distributed variable with zero mean and unit variance. The parameter α should be between -1 and 1 and its optimal value depends on the specific problem. Intuitively, the method adds some memory about the previous value of the variable to increase the consistency of a series of steps. Unfortunately, over-relaxation can only be used if all the conditional distributions that are used during Gibbs sampling have a Gaussian distribution; this is typically not the case for RBMs. The remainder of this section will describe three more popular methods for improving the mixing rate of MCMC methods applied to RBMs: fast weights, Hamiltonian Monte Carlo and tempering.

Fast Weights

To obtain better mixing rates for the sampling chains in PCD, the Fast PCD algorithm was proposed (Tieleman and Hinton, 2009). This algorithm uses an additional set of parameters that are trained using a high learning rate and using a very aggressive decay penalty that drives them to zero. The idea behind this algorithm is to get the faster mixing that is caused by high learning rates that cause the energy to rise in regions that are often reached during sampling so one can move on to other modes of the distribution.

The data dependent expectation is still computed using only the actual parameters of the model. However, the sampling updates and the model dependent expectation are computed using the sum of the normal weights and the fast weights. Due to the strong decay of the fast weights, they should go to zero as the model parameters approach an optimum. While the learning rate for the normal parameters is decreased during training like in standard PCD training, the learning rate for the fast weight stays constant.

Hamiltonian Monte Carlo

When the distribution to be modelled is continuous, it may also be possible to use gradient information to obtain better mixing rates. A popular method that uses this strategy is called Hamiltonian Monte Carlo (HMC; Duane et al., 1987; Neal, 2011), sometimes also referred to as Hybrid Monte Carlo. The HMC algorithm is especially useful when efficient Gibbs sampling is not possible because it is often far more efficient than the standard Metropolis algorithm.

HMC simulates a particle that moves over an energy surface according to Hamiltonian dynamics. The Hamiltonian is a quantity that is defined as the sum of the kinetic energy of the system and its potential energy. The simulation keeps track of both the position of a particle and its momentum. To simulate the dynamics for a probability distribution, the position corresponds to a location in the sample space and the potential energy is the same as the energy of the model at that location (i.e., proportional to the negative log density). The momentum variables are introduced artificially as some vector $\mathbf{p}$ and the kinetic energy is defined as $\frac{\|\mathbf{p}\|^2}{2m}$ where m is the mass of the particle.

Intuitively, one can think of Hamiltonian dynamics as the simulation of a frictionless puck sliding over a surface where the potential energy is proportional to the hight of the surface. When the surface is flat, the puck will move at a constant velocity. When the hight of the surface increases, the puck will slow down until the kinetic energy becomes zero and it start sliding back again. By using these dynamics to generate proposals for a Metropolis sampler, the idea is that larger steps can be taken for which a decrease in energy (or at least a similar level of energy can be expected) than would be the case with a random walk.

Formally, Hamiltonian dynamics are defined by the following equations:

$$\frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}, \tag{2.19}$$

$$\frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}}, \tag{2.20}$$

where $\mathbf{q}$ is the position[2] of the particle. The Hamiltonian $\mathcal{H}$ is defined as $\mathcal{H}(\mathbf{p}, \mathbf{q}) = U(\mathbf{q}) + K(\mathbf{p})$, where U and K are the potential energy and the kinetic energy respectively. Since we will use the potential energy given by the model defined over $\mathbf{x}$, the Hamiltonian will often be defined as $\mathcal{H}(\mathbf{p}, \mathbf{x}) = E(\mathbf{x}) + \frac{1}{2}\|\mathbf{p}\|^2$.

[2]Normally the symbol $\mathbf{x}$ would be used for this variable but I chose to use notation that is more consistent with other papers about Hamiltonian Monte Carlo.

The continuous movement of the particle is approximated using a certain number of discrete time “leap frog” steps in a way that is similar to gradient descent with momentum. After this, the Hamiltonian at the start of the simulation is compared with the Hamiltonian at the end to determine the acceptance probability of the new location. Like in the Metropolis algorithm, a rejection of the newly proposed location entails a duplication of the last accepted sample. A precise description of the Hamiltonian Monte Carlo algorithm is shown in Algorithm 3.

Algorithm 3 Hamiltonian Monte Carlo.

```
Initialize $\hat{\mathbf{x}}_0$
for $k = 1$ to $N$ do
  Sample $\eta \sim \mathcal{N}(\eta|\mathbf{0}, \mathbf{I})$
  $\mathbf{q}_0 \leftarrow \hat{\mathbf{x}}_{k-1}$
  $\mathbf{p}_0 \leftarrow \eta - \frac{1}{2}\epsilon\nabla E(\hat{\mathbf{x}}_{k-1})$
  {Perform $K$ leap frog steps.}
  for $i = 1$ to $L$ do
    $\mathbf{q}_i \leftarrow \mathbf{q}_{i-1} + \epsilon\mathbf{p}_{i-1}$
    $\mathbf{p}_i \leftarrow \mathbf{p}_{i-1} - \epsilon\nabla E(\mathbf{q}_i)$
  end for
  $\mathbf{p}_{L+1} \leftarrow \mathbf{p}_L + \frac{1}{2}\epsilon\nabla E(\mathbf{q}_L)$
  $\mathcal{H}_{\text{old}} = E(\mathbf{q}_0) + \frac{1}{2}\|\eta\|^2$
  $\mathcal{H}_{\text{new}} = E(\mathbf{q}_L) + \frac{1}{2}\|\mathbf{p}_{L+1}\|^2$
  Sample $r \sim \text{Uniform}(r|0, 1)$
  {Metropolis acceptance decision.}
  if $r < \exp(\mathcal{H}_{\text{old}} - \mathcal{H}_{\text{new}})$ then
    $\hat{\mathbf{x}}_k \leftarrow \mathbf{q}_L$
  else
    $\hat{\mathbf{x}}_k \leftarrow \hat{\mathbf{x}}_{k-1}$
  end if
end for
return $\{\hat{\mathbf{x}}_0, \cdots, \hat{\mathbf{x}}_K\}$
```

One of the downsides of the HMC algorithm is that it can only be applied to continuous sample spaces. Another weakness of the algorithm is that it introduces two new hyper-parameters that need to be chosen: the step size ϵ and the number of leapfrog steps to take L. It is common practice to make the step size adaptive to achieve a certain desired acceptance rate. This is especially important for trained models in which the optimal step size can vary a lot during training. Finally, the HMC algorithm can be computationally expensive if the gradient of the energy with respect to the input variables is expensive to compute. This is why Gibbs sampling may sometimes still be preferred over HMC. If however, Gibbs sampling is not possible, the computational overhead caused by the gradient computations is often still to be preferred over the slow random walk behavior of the standard Metropolis algorithm. Examples of research in which HMC was used to train energy models are the work about the mean covariance RBM (Ranzato and Hinton, 2010) and energy models of multiple layers of nonlinear processing units (Ngiam et al., 2011).

Tempering Methods

Another way to improve the mixing rate is Replica Exchange Monte Carlo (Swendsen and Wang, 1986), also commonly referred to as Parallel Tempering (PT), a member of the family of extended ensemble Monte Carlo methods (Iba, 2001). Recently, PT has been applied to RBM training as well (Desjardins et al., 2010b; Cho et al., 2010). This algorithm runs various Markov chains in parallel that sample from replicas of the system of interest that operate under different temperatures. Chains that operate at lower temperatures can escape from local modes by jumping to locations of similar energy that have been proposed by chains that operate at higher temperatures.

Another method, that is based on a similar principle, is to run one Markov chain that occasionally is forced to transition through various temperatures (Salakhutdinov, 2009). In other words, at certain time intervals, Gibbs sampling is done while progressively increasing the temperature, subsequently it decreases again to the original temperature and the new sample is accepted using a criterion that takes the temperature changes into account.

One downside of PT for training energy-based models is that the number of parallel sampling chains that can be used by this algorithm is limited. One can use many chains in PT to cover more temperatures. This will cause more swaps between neighbouring chains to be accepted because they are closer together. However, it will also take more sequential updates before a certain replica moves back and forth between the lowest and the highest temperatures. Another disadvantage of PT is that only the chain with the lowest temperature is actually used to gather statistics for the learning algorithm. The serial tempered transitions method doesn't suffer from this problem but using more temperatures will directly lead to longer simulation times for this algorithm as well.

2.3 Training Restricted Boltzmann Machines with Multi-Tempering

I will describe two methods for improving both the mixing rate of an MCMC sampler and the quality of the gradient estimates at each sampling step. These two methods are extensions for the PT algorithm and were originally proposed for statistical physics simulations (Athènes and Calvo, 2008). The first extension allows every possible pair of replicas to swap positions to increase the number of sampling chains that can be used in parallel. The second extension is to use a weighted average of the replicas that are simulated in parallel. The weights are chosen in a way that is consistent with the exchange mechanism. The goal of my experiments was to see if these methods from statistical physics can also be used to improve the training of energy-based models like RBMs.

2.3.1 Parallel Tempering

The idea behind PT is to run several Markov chains in parallel and treat this set of chains as one big ensemble chain that generates samples from a distribution with augmented variables. Transition steps in this combined chain can now also include possible exchanges among the sub chains. Let $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_M\}$ be the state of a Markov chain that consists of the states of M sub chains that operate under inverse temperatures $\{\beta_1, \cdots, \beta_M\}$, where $\beta_1 = 1$ and indicative of the model we want to compute expectations for. The combined energy of this system is the weighted sum given by $E(\mathbf{X}) = \sum_{i=1}^{M} \beta_i E(\mathbf{x}_i)$.

The difference in total energy that results from switching two arbitrary sub chains with indices i, j, is given by

$$E(\hat{\mathbf{X}}(i,j)) - E(\mathbf{X}) = (\beta_i - \beta_j)(E(\mathbf{x}_j) - E(\mathbf{x}_i)) \ , \qquad (2.21)$$

where $\hat{\mathbf{X}}(\cdot)$ denotes the new state of the combined chain that results from the exchange indicated by its arguments[3]. If i and j are selected uniformly and forced to be neighbours, the Metropolis-Hastings acceptance probability is given by $r_{ij} = \exp(E(\mathbf{X}) - E(\hat{\mathbf{X}}(i,j)))$. This is the acceptance criterion that is used in standard Parallel Tempering.

In the case of a binary RBM, the conditional probabilities of $\mathbf{x}$ and $\mathbf{h}$

[3] So $\hat{\mathbf{X}}(i,j,k)$ would mean that i is first swapped with j and subsequently, the sample at position j is swapped with the one at position k.

depend on the inverse temperature in the following way:

$$p(\mathbf{x}|\mathbf{h};\beta_i) = \prod_r \operatorname{sigm}\left(\beta_i\left(\sum_q w_{qr}h_q + b_q\right)\right), \tag{2.22}$$

$$p(\mathbf{h}|\mathbf{x};\beta_i) = \prod_q \operatorname{sigm}\left(\beta_i\left(\sum_r w_{qr}x_r + a_r\right)\right). \tag{2.23}$$

In other words, running a Gibbs sampling chain at a certain inverse temperature β_i simply involves a multiplication of the connection weights and bias parameters with that value. Note that this means that a chain is simulated for the energy function $\beta_i E(\mathbf{x}, \mathbf{h})$ and not $\beta_i E(\mathbf{x})$. Since for $E(\mathbf{x})$ the energy differences are not linear in the inverse temperatures anymore, they need to be computed explicitly, a reason why the use of $E(\mathbf{x}, \mathbf{h})$ is more computationally efficient. Finally, note that when β_i goes to zero, the distribution defined by the RBM becomes uniform.

From Equation 2.21 it is clear that jumps from one chain to another are more probable when the energy values of the models corresponding to the two corresponding temperatures are similar. At the same time, most MCMC algorithms are unlikely to travel far through regions where the energy is very high if there is a region of lower energy nearby. This means that the choice of the temperatures presents a trade-off: when the temperatures of two chains are nearly identical, many jumps between the chains will occur but the chance to visit additional modes has not increased much; vice versa, when the temperatures are far apart, the chain with the highest temperature will be more likely to visit additional modes but may never exchange this information with the chain running at a lower temperature. A graphical illustration of this effect is shown in Figure 2.2.

2.3.2 Multi-Tempering

To increase the number of parallel chains that PT can effectively use, I proposed to use Multiple Replica Exchange methods for RBM training. These methods have already been shown to work well in statistical physics (Brenner et al., 2007; Athènes and Calvo, 2008). To prevent the use of very different names for similar algorithms, I will refer to this method as Multi-Tempering (MT).

In Multi-Tempering (Athènes and Calvo, 2008), an index i is selected uniformly and an index j is selected with a probability that is based on the difference in total energy the proposed exchange would cause (as given in

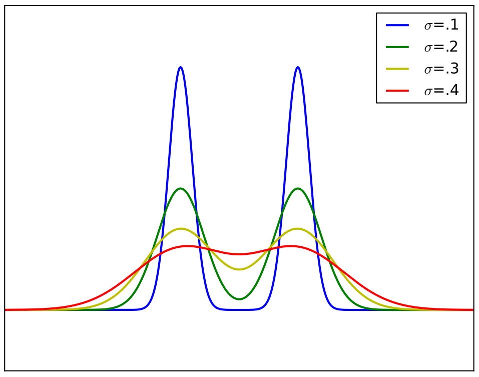


Figure 2.2: Overlapping plots of two component mixtures of Gaussians with different variance parameters. The plot illustrates that it is easier for a MCMC algorithm to visit both of the modes when the variance (and for that reason also the temperature) is higher. A particle has a higher chance to jump to a different temperature if the log densities are similar so a jump between the green and blue distributions is more likely than a jump between the red and blue distributions.

Equation 2.21):

$$p(j|i) = \frac{r_{ij}}{\sum_{j'=1}^{M} r_{ij'}} \ . \tag{2.24}$$

To ensure that the Markov chain is ergodic, the acceptance probability is chosen such that the detailed balance criterion is satisfied. In this case, the Metropolis-Hastings acceptance probability is given by

$$\begin{aligned}
A(i,j) &= \min\left\{1, \frac{p(i,j|\hat{\mathbf{X}}(i,j))p(\hat{\mathbf{X}}(i,j))}{p(j,i|\hat{\mathbf{X}})p(\hat{\mathbf{X}})}\right\} &(2.25)\\
&= \min\left\{1, \frac{\sum_{j'} e^{-E(\hat{\mathbf{X}}(i,j'))}e^{-E(\hat{\mathbf{X}}(i,j,i))}p(\hat{\mathbf{X}}(i,j))}{\sum_k e^{-E(\hat{\mathbf{X}}(i,j,k))}e^{-E(\hat{\mathbf{X}}(i,j))}p(\hat{\mathbf{X}})}\right\} &(2.26)\\
&= \min\left\{1, \frac{\sum_{j'} e^{-E(\hat{\mathbf{X}}(i,j'))}e^{-E(\hat{\mathbf{X}})}e^{-E(\hat{\mathbf{X}}(i,j))}}{\sum_k e^{-E(\hat{\mathbf{X}}(i,j,k))}e^{-E(\hat{\mathbf{X}}(i,j))}e^{-E(\hat{\mathbf{X}})}}\right\} &(2.27)\\
&= \min\left\{1, \frac{\sum_{j'} e^{-E(\hat{\mathbf{X}}(i,j'))}}{\sum_k e^{-E(\hat{\mathbf{X}}(i,j,k))}}\right\} . &(2.28)
\end{aligned}$$

The chance of two particles to trade places is still dependent on the change in energy this operation causes, but the selection procedure is now able to identify ‘lucky shots’ of chains that are further away in the temperature ranking as candidates for the exchange.

2.3.3 Using a Weighted Average of the Chains

While the Multi-Tempering algorithm has the potential to allow more of the sampling particles to contribute to the approximation of the model gradient indirectly, the way the different candidates for the exchange are selected suggests that more information can be extracted from the computed samples and their corresponding energy values.

Given the selection probabilities $p(j|i)$ from Equation 2.24 and the acceptance probabilities $A(i,j|\mathbf{X})$, one can compute a weighted average over the particles of all the Markov chains to estimate the gradient of the intractable likelihood term. This average is given by

$$\langle g \rangle_1 = \sum_{j=1}^{M} \left[(1 - A\,(1,j))\, g(\mathbf{x}_1) + A(1,j) g(\mathbf{x}_j) \right] p(j|1) \ , \tag{2.29}$$

where $g(\cdot)$ is short for $\frac{\partial E(\cdot)}{\partial \theta}$. This extension is originally called Information Retrieval but this term might lead to confusion in a Machine Learning context. I will refer to this version of the algorithm as Multi-Tempering with weighed averaging (MTw).

2.3.4 Experiments

Three experiments were done using the MNIST dataset. This dataset is a collection of $70,000$ 28×28 grayscale images of handwritten digits that has been split into a train set of 50000 images and test and validation sets of each 10000 images. The pixel intensities were scaled between 0 and 1 and interpreted as probabilities from which binary values were sampled whenever a datapoint was required.

First, it was investigated how the MT and the PT algorithms behave for different numbers of parallel chains by looking at the rate at which replicas travel from the highest temperature chain to the one with the lowest temperature. Ten RBMs with 500 hidden units were trained with PCD using a linearly decaying learning rate with a starting value of .002 for 500 epochs. Subsequently, both sampling methods were run for 10000 iterations and the number of times that a replica was passed all the way from the highest to the lowest temperature chain was counted. This experiment was done for different numbers of parallel chains. The inverse temperatures were uniformly spaced between .8 and 1. In preliminary experiments, I found that almost no returns from the highest to the lowest temperature occurred for any algorithm for much larger intervals.

The second experiment was done to get some insight in the mixing rates of the sampling methods and their success at approximating the gradient of the partition function. A small *tractable* RBM with 15 hidden units was trained on the MNIST dataset using the PCD algorithm. The different sampling methods were now run for 20000 iterations while their estimates of the gradient were compared with the true likelihood gradient which had been computed analytically. Because the success of the samplers partially depends on their random initialization, I repeated this experiment 10 times.

In the third experiment, to see how the different sampling algorithms perform at actual maximum likelihood training itself, a method called annealed importance sampling (AIS) (Neal, 1998a; Salakhutdinov and Murray, 2008) was used to estimate the likelihood of the data under the trained models. PCD, PT, MT, and MT with weighted averaging (MTw) were each used to train 10 RBM models on the train data for 500 epochs. Each method used 100 chains in parallel. The inverse temperatures for the Tempering methods were linearly spaced between .85 and 1, as expected, a slightly more conservative temperature range would be needed to make PT competitive. I used no weight decay and the order of magnitude of the starting learning rates was determined using a validation set. The learning rate decreased linearly

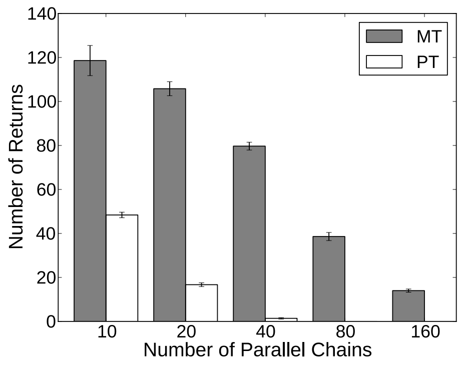


Figure 2.3: Number of returns for parallel tempering and multiple replica exchange as a function of the number of parallel chains that are used.

after every epoch.

2.3.5 Results

Figure 2.3 displays the results of the first experiment. The number of returns is a lot higher for MT at the start and seems to go down at a slightly slower rate than for PT. This allows a larger number of chains to be used before the number of returns becomes negligible.

As Figure 2.4 shows, the MT estimator was most successful at approximating the gradient of the partition function of the RBM with 15 hidden units. To my surprise, the MT estimator also performed better than the MTw estimator. However, it seems that the algorithms that used a single chain to compute the expectations (MT and PT), fluctuate more than the ones that use averages (MTw and PCD).

Table 2.1 displays the AIS estimates of the likelihood for the MNIST test set for each of the training methods. MTw outperforms all other methods

Table 2.1: Means and standard deviations of the AIS estimates of the likelihood of the MNIST test set for different training methods. Means are based on 10 experiments with different random initializations.

Epochs	MTw	MT	PT	PCD
250	−82.25(10.33)	−92.59(7.79)	−93.48(11.54)	−94.43(1.71)
500	−65.09(7.66)	−83.74(6.76)	−84.18(7.79)	−80.45(11.36)

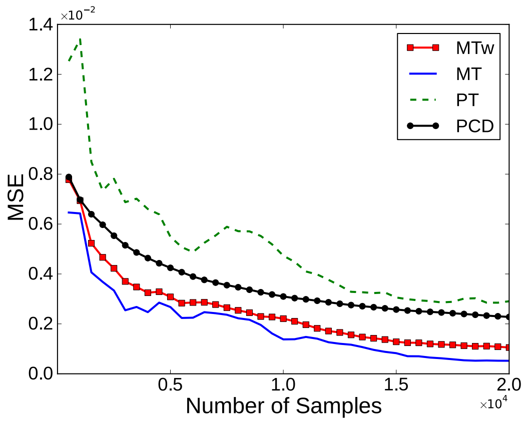
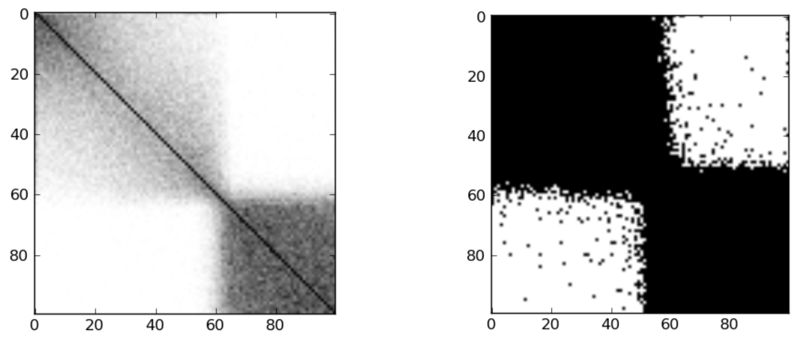


Figure 2.4: Mean Square Error (MSE) between the approximated and the true gradients of the partition function of an RBM with 15 units as a function of the number of samples.

on this task. The standard deviations of the results are quite high and MT, PT and PCD don't seem to differ much in performance. The fact that MT and PT use only a single chain to estimate the gradient seems to be detrimental. This is not in line with the results for the gradient estimates for the 15 unit RBM. It could be that larger RBMs benefit more from the higher stability of gradient estimates that are based on averages than small RBMs. The results suggest that PCD with averaged parallel chains is preferable to Tempering algorithms that use only a single chain as estimate due to its relative simplicity but that MTw is an interesting alternative. Note that the results in Table 2.1 are better than in most other papers where RBMs are trained on the MNIST dataset. While RBM training is sensitive to certain hyper parameters like the learning rate and the weight initialization, the most important reason for the high likelihood scores is that a 250K weight updates is a big number compared to, for example, the 4700 updates used in Cho et al. (2011b) or the 30K updates used in Schulz et al. (2010). Unfortunately I cannot directly compare with the results of the original PCD paper (Tieleman, 2008) because the training time in that work is reported in seconds rather in epochs or number of weights updates.

During MT training, I also recorded the transition indices for further inspection. There are clearly many exchanges that are quite large as can be seen in Figure 2.5a, which shows a matrix in which each entry $\{i, j\}$ represents the number of times that a swap occurred between chains i and j. While there seems to be a bottleneck that is difficult to cross, it is clear that some particles still make it to the other side once in a while. In Figure 2.5b, one can see that occasionally some very large jumps occur that span



(a) Matrix of exchange frequencies cut off at 100.

(b) Binarized matrix of exchanges.

Figure 2.5: Plot of inter chain replica exchanges for MT.

almost the entire temperature range.

2.4 Conclusion

I proposed two extensions to improve Parallel Tempering training for RBMs and showed that the combination of the two methods leads to improved performance on learning a generative model of the MNIST dataset. I also showed that the MTw algorithm allows more chains to be used in parallel and directly improves the gradient estimates for a small RBM. While the weighted average didn't seem to improve the rate at which the estimate of the gradient improved, it seemed to stabilize training.

Another advantage of Multi-Tempering methods over standard PT is that, given enough computational resources, one has to be less careful about choosing the temperature values. Because more chains can be used, a finer range of values can be chosen without sacrificing too much efficiency.

Limitations

A limitation of this work is obviously that I only evaluated the likelihood scores using the AIS method, which is of course itself a stochastic estimation procedure that suffers from weaknesses similar to those of MCMC methods. While I tried to reduce the variance of the likelihood estimates by running many AIS chains in parallel, one should keep in mind that the method may be sensitive to the roughness of the energy landscapes of the estimated models. If this is the case, it could be that the likelihood of a model with many very sharp modes is overestimated compared to a model that is more smooth.

At the moment, it is still difficult to beat the vanilla PCD algorithm because it can use so many chains in parallel. It is also by far the most computationally efficient method discussed because the other methods can be seen as extensions of it. It remains to be seen if the improved mixing rate of parallel tempering based methods will compensate for their higher computational complexity.

3 Alternatives to Maximum Likelihood Estimation

When maximum likelihood training is not tractable, it is possible to use stochastic approximations or to avoid the training of a generative model altogether but these are not the only solutions. This chapter delves deeper into methods for training generative models with alternative objectives, objectives that theoretically lead to similar or perhaps identical solutions as maximum likelihood learning but avoid the need for complex sampling schemes like MCMC.

Recall that the goal of maximum likelihood learning is to make some parametric density function equal to the distribution of a dataset. The *consistency* of the maximum likelihood estimator is a very important property because it guarantees that, as the number of available train samples increases, the estimator converges in probability to the data distribution. Even if an estimator is not efficient in that it achieves the Cramér-Rao bound (like the maximum likelihood estimator does), or isn't asymptotically normal, it may still be a *consistent* estimator. In situations where maximum likelihood learning is simply not possible without the use of elaborate approximation schemes, estimators that are easier to compute and can promise at least consistency, may be viable alternatives.

I will first discuss a couple of existing methods. These include quasi-likelihood methods, score matching and noise contrastive estimation. After that, I will shortly discuss recent work that relates many estimators to the Bregman divergence. Finally, I will discuss and evaluate a new statistical estimator that aims to combine some of the positive aspects of score matching and noise contrastive estimation. The section about Bregman divergences is basically a summary of the ideas introduced by Gutmann and Hirayama (2011) that help to understand the motivation behind the new estimator.

3.1 Quasi-Maximum Likelihood

An important class of estimators of this type are the so-called *quasi-maximum likelihood* estimators (Wedderburn, 1974), a class of methods that includes the *pseudo likelihood* (Besag, 1975) and the more general *composite likelihood* (Lindsay, 1988). Given certain sufficient conditions, some quasi-maximum likelihood estimators are both consistent and asymptotically normal, making them interesting alternatives for maximum likelihood learning. Unfortunately, this is not necessarily true for all types of models as pointed out in Hyvärinen (2005).

As the name suggests, the composite likelihood objective is a composition of objectives that only look at subsets of the sample space (often referred to as *events*). Let $\mathbf{x}$ be a set of stochastic variables with indices $S = \{1, \dots, D\}$ and $\{A_1, \dots, A_K\}$ and $\{B_1, \dots, B_K\}$ be subsets of S, with corresponding likelihood functions $L_k(\mathbf{x}; \theta) \propto p(\mathbf{x}_{A_k}|\mathbf{x}_{B_k}; \theta)$ and $A_k \cap B_k = \emptyset$ for any k. The symbol θ refers to the set of trainable model parameters. The composite likelihood[1] is now given by

$$CL(\mathbf{x}; \theta) = \prod_{k=1}^{K} L_k(\mathbf{x}; \theta)^{w_k}, \tag{3.1}$$

where $\{w_1, \dots, w_K\}$ are non-negative weights.

Composite likelihood objectives may differ in the precise way in which the index subsets A_k and B_k and the corresponding weights w_k are chosen. If there is only a single subset A_1 that contains all the indices, $B_1 = \emptyset$, and $w_1 = 1$, Equation 3.1 is equal to the standard likelihood. Two commonly used versions of the composite likelihood are the composite *marginal* likelihood where $B_k = \emptyset$ and the composite *conditional likelihood* for which $B_k = S \backslash A_k$. The pseudo likelihood is the special case in which all component functions $L_k(\mathbf{x}; \theta)$ are equal to conditional likelihoods in which each individual variable is conditioned on all the other variables on which it depends. In other words, only one-dimensional distributions are considered.

While quasi likelihood estimators have some nice properties, there are still many models for which they are impractical to use. Even in models as simple as RBMs, the pseudo likelihood is already quite intensive to compute because the hidden variables make the computation of the required conditional distributions more involved. For this reason, they are often more interesting to get more insight in other approximation methods. As discussed

[1]See Lindsay (1988) or Varin et al. (2011) for more detailed descriptions of the composite likelihood and its properties.

in Hyvärinen (2007a), there are interesting connections between the pseudo likelihood, and methods like contrastive divergence and score matching.

3.2 Score Matching

The estimation method called Score Matching (Hyvärinen, 2005) is consistent and in some cases easy to compute. The basic idea behind it is to train a model to match the gradient of its log likelihood with respect to the input variables $\nabla p_{\text{model}}(\mathbf{x}; \theta)$ with the gradient of the data distribution $\nabla p_{\text{data}}(\mathbf{x})$. Intuitively, normalization is not an issue when one trains a model to have the same slope as the target density everywhere. When two unnormalized density functions have the exact same shape, they will describe the same density function after normalization. Because the gradient with respect to the input variables is required, score matching can only be applied to models for continuous data. An extension of the algorithm to discrete data was introduced under the name 'ratio matching' (Hyvärinen, 2007b).

Formally, score matching aims to minimize the mean squared error between the 'scores' of the model one wants to estimate and the data distribution. The *score* is a term used to refer to the gradient of the log likelihood (or an unnormalized version of is) with respect to the input. This leads to the following objective function:

$$J(\theta) = \frac{1}{2} \int p_{\text{data}}(\mathbf{x}) \sum_{i=1}^{D} \left(\frac{\partial p_{\text{model}}(\mathbf{x}; \theta)}{\partial x_i} - \frac{\partial p_{\text{data}}(\mathbf{x})}{\partial x_i} \right)^2 \mathrm{d}\mathbf{x}, \tag{3.2}$$

where θ is the set of trainable parameters of the model distribution function p_{model}. While the score doesn't depend on the normalization constant, the new problem is now to estimate the score of the data distribution. This could possibly be done with a non-parametric method, but Hyvärinen (2005) showed that partial differentiation can be used to arrive at an expression of Equation 3.2 that does not require any approximation of this kind. The objective that is used in practice is given by

$$J(\theta) = \int p_{\text{data}}(\mathbf{x}) \sum_{i=1}^{D} \left(\frac{\partial^2 p_{\text{model}}(\mathbf{x}; \theta)}{\partial x_i^2} + \frac{1}{2} \left(\frac{\partial p_{\text{model}}(\mathbf{x}; \theta)}{\partial x_i} \right)^2 \right) \mathrm{d}\mathbf{x}. \tag{3.3}$$

The practical usefulness of score matching varies. Since Equation 3.3 requires the computation of second and third order derivatives, not all types of models may be suitable for score matching. For simple RBM-like models however, score matching seems to perform quite well (Swersky et al., 2011).

3.3 Noise Contrastive Estimation

Another approach to model estimation is called Noise Contrastive Estimation (NCE; Gutmann and Hyvärinen, 2010). The idea behind NCE is to turn density estimation into a classification problem. Given a chosen noise process that meets certain demands, a classification function is trained to discriminate between samples from the data distribution and the noise distribution. When unnormalized models are used, the normalization constant can be cast as an additional trainable parameter.

The objective used in NCE is very similar to the maximum likelihood objective of binary logistic regression and given for a single data sample $\mathbf{x}_n$ by

$$J(\mathbf{x}_n, \mathbf{y}_n, \theta) = \ln(h(\mathbf{x}_n; \theta)) + \ln(1 - h(\mathbf{y}_n; \theta)), \tag{3.4}$$

where $\mathbf{y}$ is sampled from the chosen noise process and the function $h(\cdot; \theta)$ is defined as

$$h(\mathbf{u}; \theta) = \operatorname{sigm}\left(\ln\left(\frac{p_{\text{model}}(\mathbf{u}; \theta)}{p_{\text{noise}}(\mathbf{u})}\right)\right), \tag{3.5}$$

where $\operatorname{sigm}(\cdot)$ is the standard logistic sigmoid function.

By inspecting Equations 3.4 and 3.5, one can see that the model will be trained to assign higher likelihood values to the data than the noise distribution does. Vice versa, it will also be trained to assign a lower likelihood to noise samples than the noise distribution does. Gutmann and Hyvärinen (2010) show that the estimator is consistent and that the model will learn the data distribution, given that the noise process assigns a non zero probability to all regions that are non zero in the data distribution.

NCE has a couple of nice properties that distinguish it from some other estimation methods like score matching and contrastive divergence. First, and like score matching, the method will eventually converge to the desired estimate (unlike contrastive divergence). Second, no higher order derivatives need to be computed like is necessary for score matching. This may allow NCE to be used for training more complicated models. NCE has been shown to be able to estimate models of natural images that consist of multiple layers of representation.

A weakness of NCE is that it is not very clear how the noise generating process should be chosen. While in the limit of infinitely many samples from both the data distribution and the noise distribution, NCE should converge, the number of samples that are required in practice probably depends on the choice of noise distribution. Gutmann and Hyvärinen (2010) claim that the noise distribution should ideally be as similar to the data distribution as possible. Obviously, the noise distribution should also be easy to sample from and evaluate. For these reasons, Gutmann and Hyvärinen (2010) suggest the use of simple distributions like for example, a Gaussian that has the same mean and covariance as the data.

3.4 Bregman Divergences for Estimation

Some more insight into the estimation methods discussed so far was provided by a paper in which both score matching and NCE were written as specific cases of Bregman divergences (Gutmann and Hirayama, 2011). It turns out that many new estimators can be designed by choosing different types of Bregman divergences and, potentially, auxiliary noise distributions.

Bregman divergences (Bregman, 1967) are similar to metrics but without necessarily satisfying the triangle inequality or symmetry. Common distance measures and divergences that can be written as a Bregman divergence are the squared Euclidean distance and the generalized Kullback-Leibler divergence. The relevance of Bregman divergences for statistical estimation problems is that they can be used to define objective funtions for matching probability distributions. The general Bregman divergence between two vectors **a** and **b** is defined as

$$d_\Psi(\mathbf{a}, \mathbf{b}) = \Psi(\mathbf{a}) - \Psi(\mathbf{b}) - \nabla\Psi(\mathbf{b})^\mathsf{T}(\mathbf{a} - \mathbf{b}), \tag{3.6}$$

where $\Psi(\cdot)$ is a differentiable function that is strictly convex. When $\Psi(\cdot)$ is set equal to $\|\cdot\|^2$, one obtains the squared Euclidean distance from Equation 3.6 in the following way:

$$\begin{aligned} d_{\|\cdot\|^2}(\mathbf{a}, \mathbf{b}) &= \|\mathbf{a}\|^2 - \|\mathbf{b}\|^2 - 2\mathbf{b}^\mathsf{T}(\mathbf{a} - \mathbf{b}), & (3.7)\\ &= \mathbf{a}^\mathsf{T}\mathbf{a} - \mathbf{b}^\mathsf{T}\mathbf{b} - 2\mathbf{b}^\mathsf{T}\mathbf{a} + 2\mathbf{b}^\mathsf{T}\mathbf{b}, & (3.8)\\ &= (\mathbf{a} - \mathbf{b})^\mathsf{T}(\mathbf{a} - \mathbf{b}), & (3.9)\\ &= \|\mathbf{a} - \mathbf{b}\|^2. & (3.10) \end{aligned}$$

Gutmann and Hirayama (2011) proposed to use the so-called *separable* Bregman divergence (Grünwald and Dawid, 2004), to measure the divergence between two vector valued functions **f** and **g** as

$$D(\mathbf{f}, \mathbf{g}) = \int d_\Psi(\mathbf{f}(\mathbf{x}), \mathbf{g}(\mathbf{x}))\mu(\mathrm{d}\mathbf{x}) = \int d_\Psi(\mathbf{f}, \mathbf{g})\ \mathrm{d}\mu, \tag{3.11}$$

where μ is a nondecreasing weighting function (for all practical purposes a cumulative probability mass function). Since probability density functions are scalar functions, I will follow Gutmann and Hirayama (2011) in using a notation that splits the terms from the separable Bregman divergence that depend on the target distribution and those that don't in two separate groups:

$$D(f,g) = \int \left(S_0(g) - S_1(g)f\right)\, \mathrm{d}\mu, \tag{3.12}$$
$$S_0(g) = -\Psi(g) + \Psi'(g)g, \tag{3.13}$$
$$S_1(g) = \Psi'(g). \tag{3.14}$$

This last form was also proposed as a generalization of NCE in Pihlaja et al. (2010). Noise contrastive estimation can be constructed by setting $\Psi(g) = g\ln(g) - (1+g)\ln(1+g)$ and setting μ to be the cumulative probability mass function of the noise process. The function g should also be defined as $g(\mathbf{x};\theta) = p_{\text{model}}(\mathbf{x};\theta)/p_{\text{noise}}(\mathbf{x})$ and similarly the target function f is given by $f(\mathbf{x}) = p_{\text{data}}(\mathbf{x})/p_{\text{noise}}(\mathbf{x})$. I will often omit the dependence of the function g on the parameters θ to avoid notational clutter and write the loss L directly as a functional of g. The NCE criterion can now be obtained from the negative Bregman divergence as follows:

$$S_0(g) = \ln(1+g), \tag{3.15}$$
$$S_1(g) = \ln(g) - \ln(1+g), \tag{3.16}$$
$$-D_{\text{nce}}(f,g) = -\int \left(S_0(g(\mathbf{x})) - S_1(g(\mathbf{x}))\frac{p_{\text{data}}(\mathbf{x})}{p_{\text{noise}}(\mathbf{x})}\right) p_{\text{noise}}(\mathbf{x})\, \mathrm{d}\mathbf{x}, \tag{3.17}$$
$$= \int S_1(g(\mathbf{x}))p_{\text{data}}(\mathbf{x})\, \mathrm{d}\mathbf{x} - \int S_0(g(\mathbf{x}))p_{\text{noise}}(\mathbf{x})\, \mathrm{d}\mathbf{x}, \tag{3.18}$$
$$= \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_{\text{noise}}(\mathbf{y})}\left[S_1(g(\mathbf{x})) - S_0(g(\mathbf{y}))\right], \tag{3.19}$$
$$= \mathbb{E}\left[\ln(g(\mathbf{x})) - \ln(1+g(\mathbf{x})) - \ln(1+g(\mathbf{y}))\right], \tag{3.20}$$
$$= \mathbb{E}\left[\ln(\text{sigm}(\ln(g(\mathbf{x})))) + \ln(1-\text{sigm}(\ln(g(\mathbf{y}))))\right], \tag{3.21}$$

where the last step uses the identity $\text{sigm}(\ln(x)) = x/(1+x)$.

Another interesting algorithm, that can be written as the minimization of a separable Bregman divergence, is the Importance Sampling algorithm as a method for estimating the partition function (Pihlaja et al., 2010). When $S_0(g) = g$ and $S_1(g) = \ln g$ or equivalently, when $\Psi(g) = g\ln g - g$, we get the following objective:

$$L(\theta) = \int \frac{p_{\text{model}}(\mathbf{x};\theta)}{p_{\text{noise}}(\mathbf{x})} p_{\text{noise}}(\mathbf{x})\, \mathrm{d}\mathbf{x} - \int \ln(p_{\text{model}}(\mathbf{x};\theta))p_{\text{data}}(\mathbf{x})\, \mathrm{d}\mathbf{x}, \tag{3.22}$$
$$\approx \sum_{k=1}^{K} \frac{p_{\text{model}}(\hat{\mathbf{x}}_k;\theta)}{p_{\text{noise}}(\hat{\mathbf{x}}_k)} - \sum_{n=1}^{N} \ln(p_{\text{model}}(\mathbf{x}_n;\theta)), \tag{3.23}$$

where $\{\hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_K\}$ are samples from the noise distribution. In this case, there is no weighting function μ and the objective has been split in two expectations: one with respect to the data distribution and one with respect to the noise distribution. In practice, the objective in Equation 3.23 is very unstable because of the division of densities and the curse of dimensionality. Nonetheless, it may give some insight into the reasons why NCE may be more successful.

3.5 Data Dependent Noise Contrastive Estimation

While promising results have been obtained using NCE for applications in both image and natural language processing (Gutmann and Hyvärinen, 2010, 2012; Mnih and Kavukcuoglu, 2013), it remains interesting to see how sensitive the method is to the choice of the auxiliary noise distribution. Gutmann and Hyvärinen (2010) show that the MSE of the NCE estimator (between the parameters of the estimated distribution and the true distribution) is twice the Cramér-Rao bound when the noise distribution is equal to the data distribution. This does indeed indicate that more complicated noise distributions than simple Gaussians or uniform densities may lead to better results.

It would theoretically be possible to train a separate parametric model to function as a noise model for NCE but this could lead to a noise model that is not as convenient to evaluate and obtain samples from. Non-parametric density estimators, like the Parzen Density Estimator (PDE), may provide a better starting point because they are often easy to sample from even though in general the computation of the likelihood scores of the estimators is costly. A PDE has the nice property that it converges to the true distribution in the limit of infinitely many samples (Parzen et al., 1962) (i.e., it is consistent).

The biggest limitation of non-parametric models for NCE-like training algorithms is that they are very costly to evaluate. It would be far more practical if it would only be required to sample from the non-parametric distribution without the need to evaluate density values directly. The ideas about Bregman divergences for estimating statistical models suggest some approaches that may solve this problem.

Gutmann and Hirayama (2011) suggested a possible estimator in which the noise distribution is conditioned on the data itself. They define the noise distribution as $p_{\text{noise}}(\mathbf{x}) = \alpha p_{\text{data}}(\mathbf{B}\mathbf{x} + \mathbf{v}) + (1 - \alpha) p_{\text{data}}(\mathbf{x})$, where α is from the interval $[0, 1]$. The matrix $\mathbf{B}$ is assumed to be orthonormal (for mathematical simplicity) and $\mathbf{v}$ is some random perturbation vector. The idea behind this noise distribution is that it is a mixture of both the data distribution itself and its density for a perturbed version of the data. The target function is given by $f(\mathbf{x}) = p_{\text{data}}(\mathbf{x})/p_{\text{noise}}(\mathbf{x})$ and the weighting function μ is set equal to the cumulative distribution function of the noise distribution. The function to train is given by $g(\mathbf{x}) = p_{\text{model}}(\mathbf{x})/(\alpha p_{\text{model}}(\mathbf{Bx} + \mathbf{v}) + (1 - \alpha)p_{\text{model}}(\mathbf{x}))$. Finally, after combining these definitions with the definition of the separable Bregman divergence for scalar functions in Equation 3.12, the objective to optimize is given by

$$J(g) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\alpha S_0(g(\mathbf{B}^\mathsf{T}\mathbf{x} - \mathbf{B}^\mathsf{T}\mathbf{v})) + (1-\alpha)S_0(g(\mathbf{x})) - S_1(g(\mathbf{x}))\right], \tag{3.24}$$

where the fact is used that $\mathbf{z} = \mathbf{B}^\mathsf{T}\mathbf{x} - \mathbf{B}^\mathsf{T}\mathbf{v}$ has the distribution $p_{\text{data}}(\mathbf{Bx}+\mathbf{v})$. To my knowledge, the objective in Equation 3.24 has not been investigated directly but was only used to show relations between the Bregman divergence and algorithms like score matching and ratio matching.

To obtain a practical estimator that is based on NCE and a non-parametric noise distribution I first simplified the objective in Equation 3.24 by setting $\mathbf{B} = \mathbf{I}$ and replacing the mixture noise distribution with $p_{\text{noise}}(\mathbf{x}) = p_{\text{data}}(\mathbf{x} + \mathbf{v})$. This corresponds to a specific version of the noise mixture distribution in which $\alpha = 1$. Naturally, the target and trainable functions become $f(\mathbf{x}) = p_{\text{data}}(\mathbf{x})/p_{\text{data}}(\mathbf{x} + \mathbf{v})$ and $g(\mathbf{x}) = p_{\text{model}}(\mathbf{x})/p_{\text{model}}(\mathbf{x} + \mathbf{v})$, respectively. The NCE definitions of S_0 and S_1 were used because of their numerical robustness (many other functions suffer from multiplications with exponential numbers):

$$S_0(x) = \ln(1 + x), \tag{3.25}$$

$$S_1(x) = \ln(x) - \ln(1 + x). \tag{3.26}$$

By choosing μ to be the cumulative distribution function of the noise distribution again and combining these definitions with Equation 3.12, one obtain the following integral:

$$\begin{aligned} J(g, \mathbf{v}) &= \int (S_0(g(\mathbf{x})) - S_1(g(\mathbf{x}))f(\mathbf{x}))p_{\text{data}}(\mathbf{x} + \mathbf{v})\ \mathrm{d}\mathbf{x}, && (3.27)\\ &= \int (S_0(g(\mathbf{x} - \mathbf{v})) - S_1(g(\mathbf{x})))p_{\text{data}}(\mathbf{x})\ \mathrm{d}\mathbf{x}, && (3.28)\\ &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[S_0(g(\mathbf{x} - \mathbf{v})) - S_1(g(\mathbf{x}))\right], && (3.29)\\ &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\ln\left(1 + e^{g(\mathbf{x}-\mathbf{v})}\right) - \ln(g(\mathbf{x})) + \ln\left(1 + e^{g(\mathbf{x})}\right)\right]. && (3.30)\end{aligned}$$

Note that the objective in Equation 3.27 is still defined as a function of the perturbation vector $\mathbf{v}$. Since the original motivation for the objective was to do NCE with a noise distribution in the form of a PDE, the vector $\mathbf{v}$ was assumed to be a stochastic variable with a spherical Gaussian distribution. To take this definition into account, the final objective was defined as the expectation of Equation 3.27 with respect to $\mathbf{v}$:

$$J(g) = \mathbb{E}_{\mathbf{v}\sim\mathcal{N}(\mathbf{0},\sigma^2\mathbf{I})}\left[\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\ln\left(1+e^{g(\mathbf{x}-\mathbf{v})}\right) - \ln(g(\mathbf{x})) + \ln\left(1+e^{g(\mathbf{x})}\right)\right]\right]. \tag{3.31}$$

In practice, the expectation over the data distribution can be approximated using the sample average. The expectation over the perturbation vector can be approximated by generating samples from a Gaussian distribution. To avoid overly long acronyms I will simply refer to this estimator as Data Dependent Contrastive Estimator (DDCE).

3.5.1 Relation to Other Methods

Since the DDCE objective will be minimized when $\frac{p_{\text{model}}(\mathbf{x})}{p_{\text{model}}(\mathbf{x}+\mathbf{v})} = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}+\mathbf{v})}$, the method will only make p_{model} *proportional* to the data distribution. This makes the method somewhat similar to score matching. Likewise, if the functions are perfectly proportional, they should define the same distribution after normalization.

An important difference between the objective in Equation 3.31 and standard NCE is that it is not necessary to evaluate the noise distribution directly. It is only required that samples from the perturbation distribution can be obtained efficiently. Another difference is that NCE also tries to learn the partition function when it is added as an additional parameter. Finally, the variance of the perturbation distribution has a similar role as the kernel width of a PDE; it is a hyper-parameter that should ideally be tuned using techniques like cross-validation.

Another estimation method that has some interesting similarities with DDCE is the training procedure for Denoising Autoencoders (Vincent et al., 2008). In a recent paper, it was shown that the act of training a neural network with a single hidden layer to reconstruct its input after applying Gaussian noise to it, is (under some circumstances) equivalent to a minimization of the distance between the gradient of the energy function of the model and the gradient of a PDE (Vincent, 2011). This is very similar to score matching. The method doesn't require an explicit evaluation of the gradient of the PDE because this gradient can be approximated by sampling from the PDE and averaging the local gradients. Unfortunately, this trick cannot be used for NCE because NCE also needs the density of the noise model, a quantity that cannot be approximated as easily as the gradient. What makes this approach interesting, is that it should be possible to apply it to other models than standard autoencoders. By sampling a data point, adding noise to it, and minimizing the MSE between the gradients with respect to the data point itself and the noisy version, this method should work for any model for which the gradient of the energy with respect to the input variables can be computed. DDCE has the advantage that it doesn't require the evaluation of energy gradients with respect to the input but it does require the energy of the model itself to be evaluated. The denoising autoencoder trick can also be used for model for which one can only compute the gradient of the energy and not the energy itself.

Finally, the method has some of the characteristics of Contrastive Divergence. Both methods use a corruption process to find locations near the data to alter the shape of the energy function of the model locally. The methods are also similar in that they only required evaluation of the energy gradient with respect to the parameters and not evaluation of the energy itself or its gradient with respect to the input variables. The applicability of Contrastive Divergence depends on the possibility to perform Gibbs sampling efficiently.

3.5.2 Consistency of the Estimator

To investigate the consistency of the DDCE empirically, I did experiments that were similar to those designed by Gutmann and Hyvärinen (2012) to illustrate the consistency of NCE. The goal of the experiments was to learn the parameters of a 5 dimensional zero-mean Gaussian distribution. The log probability density function to train is defined as $\ln p(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{\Lambda}\mathbf{x} + Z$, where $\mathbf{\Lambda}$ is a positive definite precision matrix and Z is the normalization constant of the distribution given by $-\frac{1}{2}\ln|\det\mathbf{\Lambda}| - \frac{D}{2}\ln(2\pi)$. The model to be trained was assumed to be unnormalized.

To evaluate the rate of convergence of the estimated parameters $\hat{\mathbf{\Lambda}}$ to the true parameters $\mathbf{\Lambda}$ of the target model, 150 random target precision matrices were created in the same way as described by Gutmann and Hyvärinen (2012): For each precision matrix, a set of 5 eigenvalues $\{\lambda_1, \dots, \lambda_5\}$ was sampled uniformly from the interval $[0.1, 0.9]$. Subsequently, a set of random eigenvectors $\mathbf{E}$ was created by sampling a random matrix $\mathbf{M}$ from a standard normal distribution and projecting it onto the space of orthonormal matrices using $\mathbf{E} = (\mathbf{M}\mathbf{M}^{\mathsf{T}})^{-\frac{1}{2}}\mathbf{M}$. The matrix defined by this set of eigenvalues and eigenvectors was used as the target precision matrix to learn. Note that it is by definition positive definite and symmetrical. The 150 precision matrices were used to define multivariate distributions from which samples were taken to generate data sets of various sizes. For comparison, the models were trained using both the NCE and DDCE objectives. These objectives were optimized with the BFGS algorithm. Finally, the results from the

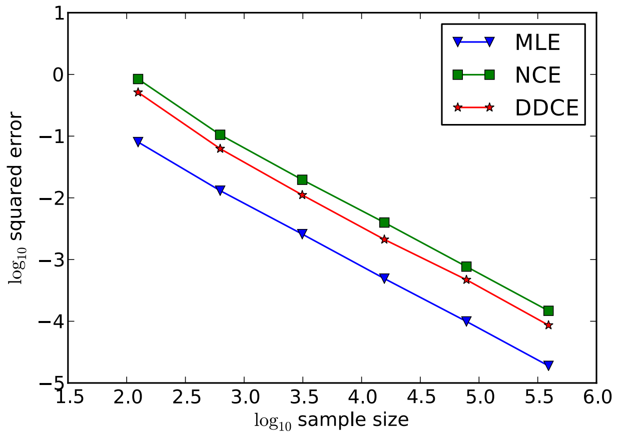


Figure 3.1: Line plots of the MSE between the parameters provided by the estimators and the true parameters for various sample sizes. The points in the graph represent the log of the average of 150 experiments.

analytically available maximum likelihood estimator (simply the inverse of the covariance matrix of the samples) were used for comparison as well. The performance of the estimators was evaluated by measuring the MSE between the estimated parameters and the true parameters of the model.

Figure 3.1 displays the logarithm of the averaged MSE values between the true parameters and the DDCE, NCE and MLE predictions as a function of the logarithm of number of available samples. As is clear from the graph, the MSE scores go down linearly as a function of the number of data points; this is indeed an indication that all the estimators are consistent. The MLE prediction, being an *efficient* estimator, displays the lowest MSE, followed by the DDCE and NCE predictions.

3.6 Learning Features for Image Patches with DDCE

It has been affirmed that the DDCE method can provide consistent estimates for a low dimensional toy example, the next step is to see if it can be applied to real world data. To this end, a Product of Student-t distributions (PoT; Welling et al., 2003) was trained to represent a data set of small natural image patches (i.e., randomly selected squares of neighbouring pixels). The goal of these preliminary experiments was simply to inspect the learned parameters of the models visually and not to make any claims about the relative performance of the estimation methods involved.

3.6.1 Products of Student-t Distributions

I chose to use a Product of Student-t distributions because it is a relatively simple energy-based model for images that I found to be relatively robust to different settings of its hyper-parameters with standard CD training. PoT models and especially Fields of Experts (Roth and Black, 2005) models (which are basically PoT models that employ 2D convolutions similar to convolutional neural nets) have been applied to various image processing tasks like denoising and inpainting. A reason for this may be that Student-t distributions have fatter tails than Gaussians distributions. Besides the robustness against potential outliers, this may also aid optimization because the gradient of the energy function is likely to have less extreme values.

A PoT model with K Student-t distributions has an energy function given by

$$E(\mathbf{x}) = \sum_{i=1}^{K} \gamma_i \ln \left(1 + \frac{1}{2}(\mathbf{w}_i^{\mathsf{T}}\mathbf{x})^2\right), \tag{3.32}$$

where $\mathbf{w}_i$ and γ_i are trainable parameters and $\gamma > 0$. To enforce the positivity constraint on γ it will in practise be defined as $\gamma_i = e^{z_i}$ such that z_i will be the actual trainable parameter. Note that the hidden units don't appear in the energy function because they have already been marginalized out.

The PoT can also be seen as a quadratic RBM in which the hidden units are Gamma distributed conditioned on the visible units and the visible units are Gaussian distributed given the hidden units. The energy for this interpretation of the model is

$$E(\mathbf{x}, \mathbf{h}) = \sum_{i=1}^{K} h_i \left(1 + \frac{1}{2}(\mathbf{w}_i^{\mathsf{T}}\mathbf{x})^2\right) + (1 - \gamma_i) \ln h_i. \tag{3.33}$$

Unlike an RBM however, the visible units are not independent conditioned on the hidden units; the weights and hidden units define a covariance matrix that allow second order interactions to be modelled directly (as opposed to being implicitly available by summing over the hidden variables). The distribution of the visible variables conditioned on the hidden variables is defined as $p(\mathbf{x}|\mathbf{h}) = \mathcal{N}(\mathbf{x}|\mathbf{0}, (\mathbf{W}^{\mathsf{T}}\boldsymbol{\Lambda}\mathbf{W})^{-1})$ where $\boldsymbol{\Lambda}$ is a diagonal matrix with $\Lambda_{ii} = h_i$.

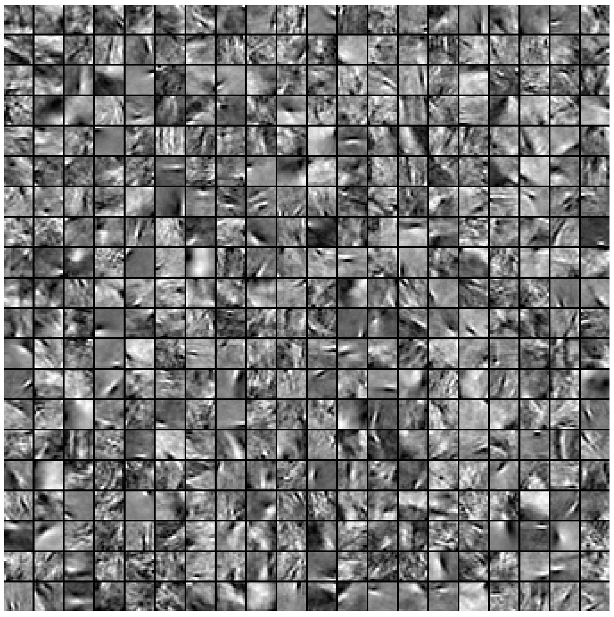
3.6.2 Qualitative Evaluation

The data consisted of $200,000$ patches of 16×16 pixels from the collection of 4000 gray-scale images taken by Hans van Hateren (Hateren and Schaaf, 1998). The patches were represented as 256 dimensional vectors which were reduced to 142 dimensions using PCA whitening. The data samples were split in a set of $150,000$ samples used for training and a set of $50,000$ samples for validation purposes.

PoT models were trained as density models of the image patches using different training methods. One model was trained using persistent contrastive divergence implemented through Gibbs sampling. Because each sampling step requires the inversion of a $D \times D$ matrix this was done with only a single Markov chain. Two other models were trained with NCE but with either a standard normal distribution or a PDE as the noise distribution. The last model was trained using DDCE. All models had 400 hidden units. Unlike previous work on PoT model, the weights of the models were not constrained to have unit norm. The noise distributions for NCE had a standard deviation or kernel width of 1. For DDCE a perturbation standard deviation of 0.1 was used.

Gradient descent was used to optimize the parameters of the models because of its computational efficiency for larger data sets and the ease with which new samples from the noise distributions can be integrated in the optimization scheme. Except for the model trained with PCD, all models were trained for 300 iterations over the data set using batches of 50 data points for each weight update. Learning rates were chosen such that the measurable objectives decreased as much as possible on the validation set. For PCD training, the choice of learning rate was simply based on monitoring a visualization of the model weights during the early iterations. Because PCD used single samples for each weight update it was only trained for 15 iterations over the dataset, resulting in more than twice as many weight updates for the PCD algorithm compared to the other methods. All models were trained with linearly decreasing learning rates. DDCE also seemed to benefit somewhat from the addition of a momentum term. All these setting were found in a rather ad hoc interactive way largely using prior intuitions about the behavior of the optimization procedure and should not be expected to be optimal for practical applications.

As both Figure 3.2 and Fig. 3.3 show, weights learned by the different optimization methods look quite similar. This visual inspection suggests that all of the algorithms could be viable alternatives to each other. The only visible difference seems to be that the features learned by the NCE algorithm with a Gaussian noise distribution look somewhat messier than those of the other methods. The features seem to implement edge detectors


(a) NCE with Gaussian noise

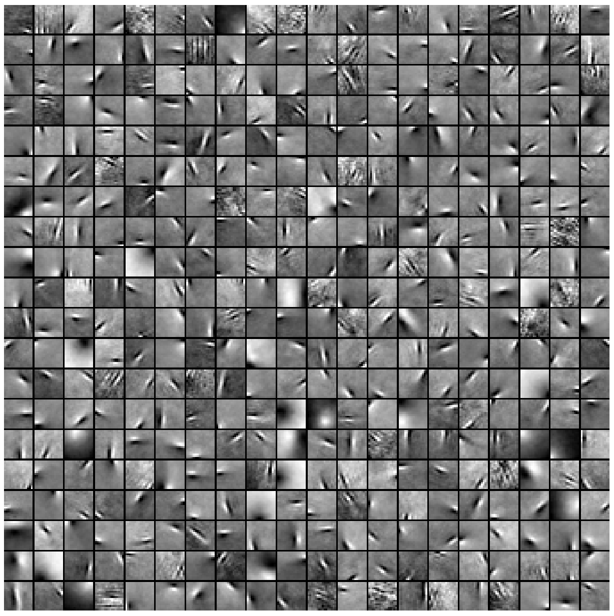
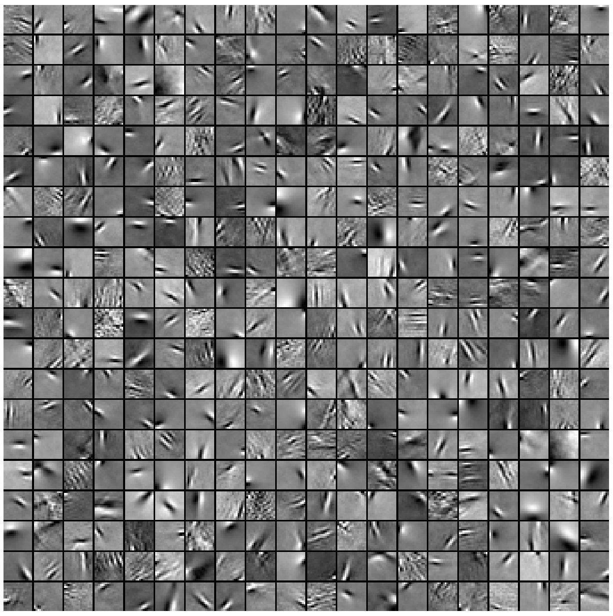
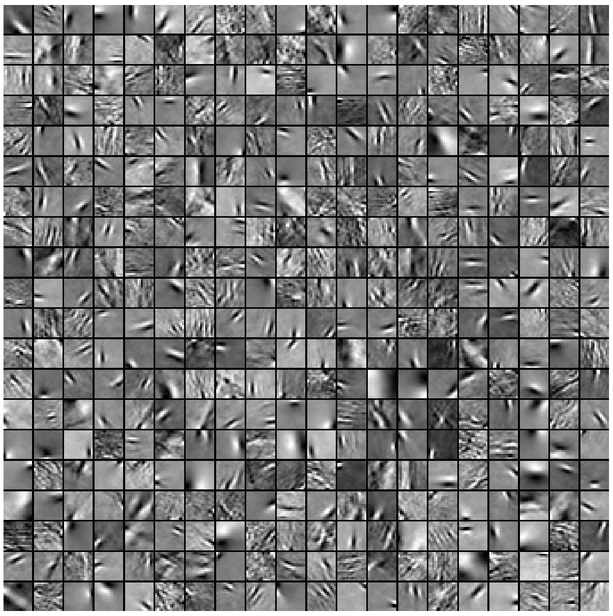

(b) NCE with PDE noise

Figure 3.2: Visualizations of the connection weights of the PoT models trained with NCE training using either a Gaussian distribution or a PDE as the auxiliary noise distribution.



(a) PCD



(b) DDCE

Figure 3.3: Visualizations of the connection weights of the PoT models trained with standard PCD training and those obtained using DDCE.

and high frequency detectors that look somewhat like Gabor filters.

3.7 Discussion

In this chapter, a new statistical estimator was introduced that combines ideas from Noise Contrastive Estimation, Bregman divergence minimization and non-parametric density estimators. Preliminary results indicate that the estimator is consistent and is able to learn image features that look similar to those obtained by more common optimization methods like Persistent Contrastive Divergence.

The estimator is relatively efficient from a computational perspective because no densities of noise distributions need to be computed and sampling from a spherical Gaussian is relatively cheap as well. However, the method does require the gradient of the energy of the model with respect to the parameters to be computed three times. All in all, the estimator seems most suitable for models for which Markov Chain Monte Carlo is impractical and second order gradients are tricky to compute. This rules out PCD and score matching, making NCE and the DDCE more viable alternatives.

An important weakness of the new estimator is its numerical behavior. The BFGS method, that was used for the consistency experiment, failed to find a good minimum of the objective when single precision floating point numbers were used. This is unfortunate because the optimization of big energy models often benefits from parallel processing units that are at this moment far more optimized for single precision computations. The NCE method seemed to suffer from this problem as well and of course it may have been a problem inherent in the specific task of estimating the parameters of a multivariate Gaussian using gradient methods. In general, I found that it helped to train relatively long with a small value for the variance of the corruption process to keep the gradients more stable. As more samples become available, the required perturbation variance should be allowed to become smaller, presumably making optimization easier.

4 Missing Value Imputation with Temporal EBMs

This chapter is about energy-based models for time-series that are trained directly to provide predictions for tasks that would typically be solved with generative models. The bottom line is that when maximum likelihood is not possible or highly impractical, it may be better to train models to provide good predictions directly. This can be done by considering the inference procedure that is used to be part of the model itself. The chapter focusses on inference procedures based on gradient descent and mean-field iterations. The proposed methods are used to estimate models that would be very difficult to train with maximum likelihood learning. The models are applied to missing value imputation tasks.

4.1 Optimizing Predictions

While the likelihood objective is defined in terms of the joint configuration of the observed variables in a model, it is also possible to define objective functions over values of variables after some form of inference has been performed. This means that unlike in energy-based learning, an objective is directly defined as a function of the predictions made by the model. In other words, we don't care about the likelihood a model assigns to a certain state of the variables as long as inference in the model leads to the values we want to see. While this is a normal state of affairs for discriminative models, where, for example, the loss is defined as a function of the predictions made by a classifier or a regression model, it may be less obvious that this principle can be generalized to many energy-based models that are normally used in a generative fashion.

Recall that inference in energy-based models is defined as the search for the lowest value of a function and can be seen as an optimization problem of the form:

$$\hat{\mathbf{X}}_{(i,j)\in\Omega} \leftarrow \arg\min_{\mathbf{X}_{(i,j)\in\Omega}} E(\mathbf{X};\theta). \tag{4.1}$$

It is analogous to state that in a probabilistic model, one searches for the configuration of variables with the highest likelihood with, for example, message passing algorithms like the Viterbi algorithm.

Given some initial set of values $\mathbf{X}_n$, and a subset of these indicated by a set of indices Ω, we can define the optimization in Equation 4.1 as a parameterized function $\mathcal{F}(\mathbf{X};\theta) = \hat{\mathbf{X}}_{(i,j)\in\Omega}$, where θ refers to the same parameters as those that define the energy function of the model. Say we want to minimize some loss objective that is defined as a function of the values predicted by the optimization mapping $L(\mathcal{F}(\mathbf{X};\theta))$, the chain rule now allows us to compute the gradient of this loss with respect to the parameters as

$$\frac{\partial L(\mathcal{F}(\mathbf{X};\theta))}{\partial\theta} = \frac{dL(\mathcal{F}(\mathbf{X};\theta))}{d\mathcal{F}(\mathbf{X};\theta)}\frac{\partial\mathcal{F}(\mathbf{X};\theta)}{\partial\theta}. \tag{4.2}$$

The point is, that if the derivative of $\mathcal{F}$ can be computed with respect to θ, it becomes possible to train the model to provide predictions that minimize the loss L directly.

It is important to note that methods of this type train a model for one specific task. While a very reliable generative model can be used for many different tasks like classification, data generation, and denoising, a model trained to optimize the outcome of an inference procedure for one specific task cannot necessarily be used for other applications. However, in Section 4.7 I will discuss an interesting exception to this rule.

4.2 Truncated Inference

In some cases, one wants to solve the optimization problem in Equation 4.1 with respect to a large number of variables. When the variables of interest are continuous, this optimization problem can be solved with gradient-based optimization methods (among others). Unfortunately, it can take many iterations to reach the nearest local minimum of the energy function. Since this optimization problem needs to be solved to compute the gradient of the loss function, it is within the inner loop of the training procedure and very likely to be the computational bottleneck. For this reason, the computational efficiency of the optimization algorithm is very important.

Since the model is trained to improve the quality of the answers that are found by the energy minimization procedure, one may wonder how harmful it is when this optimization is not entirely accurate. In fact, it has been shown that, in some cases, this is not much of a problem as long as the limitations of the inference procedure are taken into account during training, (Wainwright, 2006). This strategy was probably first proposed in Barbu (2009) to perform very fast image denoising. More recently, similar strategies have been applied to other tasks as well (Domke, 2012; Stoyanov et al., 2011). I will refer to this strategy as *truncated inference*. A proper name for its use in predictive modelling would be *truncated empirical risk minimization*, to clarify the relation to the term used in Stoyanov et al. (2011) for training models directly for prediction.

In truncated inference, the optimizer is not necessarily run until convergence and a prediction is now given by

$$\hat{\mathbf{X}}_{(i,j)\in\Omega} \leftarrow \underset{\mathbf{X}_{(i,j)\in\Omega}}{\text{opt-alg}}\, E(\mathbf{X};\theta). \tag{4.3}$$

In other words, the operator $\mathcal{F}$ has been simplified. Provided that this operator is differentiable, we can use the chain rule to construct backpropagation algorithms for it.

Keep in mind that this optimizer can take the form of message passing, gradient steps, coordinate descent and many other methods for energy based models. To choose the most appropriate optimization algorithm, one needs to consider both the limitations of the model and the properties of the prediction task we try to solve.

4.3 Missing Value Imputation for Time-Series

A common problem in statistical modelling, is that some measurements may be missing. Missing values can for example occur due to noise, malfunctioning sensors or data loss. Time-series can also suffer from this problem, but their temporal structure may be helpful for recovering missing values when this temporal structure is modelled properly. However, this can be a very challenging task when the time series of interest are generated by complex non-linear processes.

Simple techniques like nearest neighbour interpolation treat the data as independent and ignore temporal dependencies. Linear, polynomial and spline-based interpolation techniques tend to fail when variables are missing for extended periods of time. It appears that more complicated models are needed to make good predictions about missing values in high dimensional time-series.

4.3.1 Previous Work on Imputation

Given a set of observed variables, one can try to define a function that returns a set of predictions for the values that are missing. Models of this type belong to the discriminative family and are for example linear regression, support vector machines and multi-layer perceptrons (Bishop, 2006). Neural networks are interesting candidates for time-series tasks because their connection structure can be designed to capture prior beliefs about the temporal dependencies. Examples of neural networks that are able to deal with temporal sequences are Recurrent Neural Networks and one-dimensional Convolutional Neural Networks (Waibel et al., 1989). However, most models of the discriminative type assume that the ordering of known and unknown variables is fixed. It is not always clear how to use them if a variable that was known for one data point has to be predicted for another and vice versa.

Nonetheless, there has been some work on training neural networks for missing value recovery in a discriminative way. Nelwamondo et al. (2007) trained autoencoder neural networks to impute missing values in non-temporal data. They used genetic algorithms to insert missing values that maximized the performance of the network. Unfortunately it is not straightforward to apply this method to high dimensional time-series as the required models would be too large for genetic algorithms to remain computationally feasible. Gupta and Lam (1996) trained neural networks for missing value imputation by using some of the input dimensions as input and the remaining ones as output. This requires many neural networks to be trained and limits the available datapoints for each network to those without any missing input dimensions. This method is especially difficult to apply to high dimensional data with many missing dimensions. Convolutional neural networks have been trained to restore images (Jain et al., 2007) in a supervised way but in that work the task was not to impute missing values but to undo the effects of a contaminating process in a way that is more similar to denoising.

4.3.2 Generative Models for Time-Series

Probabilistic graphical models (Koller and Friedman, 2009) have often been used to model time-series. Examples of probabilistic graphical models for time-series are Hidden Markov Models (HMM) and Linear Dynamical Systems. For simple tractable models like these, conditional probabilities can be computed analytically. Unfortunately, these simple models have trouble with the long range dependencies and nonlinear structures in many of the more interesting datasets. HMMs have trouble modelling data that is the result of multiple underlying processes because only a single hidden state variable is used. The number of states that is required to model information about the past, grows exponentially as a function of the number of bits to represent. More complicated directed graphical models often suffer from the so-called *explaining away* phenomenon (Pearl, 1988).

Undirected graphical models (also known as Markov Random Fields) have been used as well but tend to be intractable. An example of an intractable model for high dimensional non-linear time series is the Conditional Restricted Boltzmann Machines (CRBMs; Taylor et al., 2007), which was used to reconstruct motion capture data.

There are some less conventional approaches to model nonlinear time series in a generative way as well. A combination of the expectation maximization algorithm and the Extended Kalman Smoother can be used to train certain classes of nonlinear dynamical systems (Ghahramani and Roweis, 1999). The difficulty with this approach is that fixed radial basis functions need to be used for approximating the nonlinearities to keep the model tractable. It is not clear how these would scale to higher dimensional state spaces where radial basis functions become exponentially less effective.

Non-parametric models like the Gaussian Process Latent Variable Model (Lawrence, 2004) have also been used to develop models for sequential tasks like synthesizing and imputing human motion capture data. A continuation of this work is the Gaussian Process Dynamical Model (Wang et al., 2008). While models of this type tend to display nice generalization properties for small datasets, their application to larger datasets is limited because of the need to invert a great number of kernel matrices that grow cubicly with the number of data points. There has been some work on improving the computational efficiency of these models by introducing sparsity (Lawrence, 2007) but parametric graphical models tend to remain more practical for larger datasets.

I argue, that while graphical models are a natural approach to time-series modelling, training them probabilistically is not always the best strategy when the goal is to use them for prediction tasks, especially, if the model is intractable.

- By trying to model the whole joint distribution of a data set, a large part of the flexibility of generative models is used to capture relationships that might not be necessary for the task of interest. A model might put too much effort into assigning low likelihoods to regions in the sample space that are very different to the patterns of observed values that will be encountered during the imputation task. An energy model with a deterministic inference method can make predictions that are directly optimized for the task of interest itself.

- Since the normalization constant of many generative models is intractable, inference needs to be done with methods like sampling or variational inference. Deterministic models circumvent this problem.

- Training is intractable as well for many generative graphical models and algorithms that approximate maximum likelihood learning have to be used.

There are some generative architectures that can handle sequential data with non-linear dependencies. Certain types of Dynamical Factor Graphs (Mirowski and LeCun, 2009) are still tractable when the energy function is designed in such a way that the partition function remains constant. Another tractable non-linear dynamical system is based on a combination of a recurrent neural network and the neural autoregressive distribution estimator (Larochelle and Murray, 2011; Boulanger-Lewandowski et al., 2012). Overall, however, discriminative energy-based models allow for a broader class of possible models to be applied to missing value imputation while maintaining tractability.

4.3.3 Training for Missing Value Imputation

I will mainly discuss the situation in which one wants to train a model on a train set in which no values are missing but missing values are expected to appear during test time. In this case, artificial missing values can be created by defining some probability distribution that selects values that will be considered absent during training. The ideal situation would be that this distribution over missing value locations is identical to the one that will be encountered during testing. This is not very likely to be the case so one would hope that a good imputation method that uses artificially selected missing values during training is robust to an inaccurate missing value selection procedure. When we select a distribution for the locations of the missing values, this distribution should represent our domain knowledge but a good model should learn enough about the internal structure of the data to not be fully reliant on the accuracy of this distribution.

The other possible situation is that there are also values in the train data itself that are missing. In this case a model should know how to do something sensible with the variables that represent these values and not simply set them to some arbitrary value. The models I proposed can also handle this problem but most of my research deals with 'healthy' train data in which no values are missing.

Given a sequence $\mathbf{X}_n$, represented as a matrix of data vectors $\{\mathbf{x}_1, \cdots, \mathbf{x}_T\}$, let Ω be the set of tuples of indices (i, j) that point to elements of data vectors that have been labelled as missing. For real-valued data, a sound error function to minimize is the sum of squared errors between the values we predicted $\hat{\mathbf{X}}_n$ and the actual values $\mathbf{X}_n$:

$$L = \frac{1}{2} \sum_{(i,j)\in\Omega} ((\mathbf{X}_n)_{ij} - (\hat{\mathbf{X}}_n)_{ij})^2. \tag{4.4}$$

Note that this loss function is only defined for a single data sequence. Since Ω will be sampled from some distribution, the actual objective that is minimized during training is the expectation of the sum squared error under a distribution over the missing values as defined for N_{data} sequences by

$$O = \frac{1}{2} \sum_{n=1}^{N_{\text{data}}} \sum_{\Omega} P(\Omega) \sum_{(i,j)\in\Omega} \Big((\mathbf{X}_n)_{ij} - (\hat{\mathbf{X}}_n)_{ij}\Big)^2 . \tag{4.5}$$

All models will be trained to minimize this objective.

The selection of $P(\Omega)$ during training is task dependent and should reflect prior knowledge about the structure of the missing values. If it is known that missing values occur over multiple adjacent time steps for example, this can be reflected in the choice of $P(\Omega)$. For tasks that contain missing values due to malfunctioning sensors or asynchronous sampling rates, this pattern is likely to be present. I expect that a good choice of $P(\Omega)$ is important but that the objective is robust to $P(\Omega)$ being somewhat imprecise.

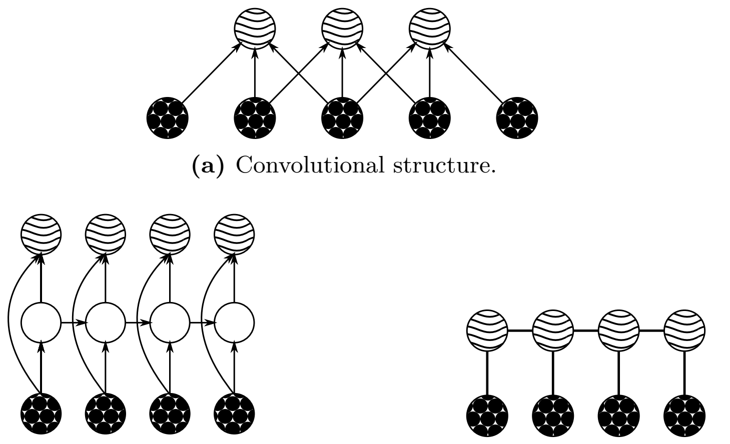
4.4 Models for Imputation

The first model I propose is based on a convolution over the data that is coupled to a set of hidden variables, as shown in Figure 4.1a. The second model is shown in Figure 4.1b and is a recurrent neural network that is coupled to a set of hidden variables as well. In both of these models, inference is done with gradient descent. The third model is a Markov Random Field with distributed hidden state representations for which the inference procedure consists of mean-field iterations. This model is shown in Figure 4.1c.

The first two models I will describe have a tractable free energy function $E(\mathbf{X})$ and I chose to use gradient descent to optimize it. The third model has no tractable free energy $E(\mathbf{X})$ and I chose to use coordinate descent to optimize a variational bound on the free energy instead.

4.4.1 The Convolutional Energy-Based Model

I will call the first model the Convolutional Energy-Based Model (CEBM). The CEBM has an energy function that is defined as a one dimensional



(a) Convolutional structure.

(b) Recurrent structure. **(c)** Undirected structure.

Figure 4.1: The three model structures that are used in this paper. The wavy circles represent the hidden units, the circles filled with dots the visible units and empty circles represent deterministic functions. The time dimension runs from the left to the right and each circle represents a layer of units.

convolution over a sequence of data combined with a quadratic term and is given by

$$E(\mathbf{X}, \mathbf{H}) = \sum_{t=1}^{T} \left(\frac{\|\mathbf{x}_t - \mathbf{b}_x\|^2}{2\sigma^2} - \mathbf{h}_t^\mathsf{T} \mathbf{g}_{\text{conv}}(\mathbf{X}, t; \mathbf{W}) - \mathbf{h}_t^\mathsf{T} \mathbf{b}_h \right), \tag{4.6}$$

where $\mathbf{b}_x$ is a vector with biases for the visible units and $\mathbf{H}$ is a set of binary hidden units. The function $\mathbf{g}_{\text{conv}}(\cdot)$ is defined by

$$\mathbf{g}_{\text{conv}}(\mathbf{X}, t; \mathbf{W}) = \mathbf{W}[\mathbf{x}_{t-k} \oplus \cdots \oplus \mathbf{x}_t \oplus \cdots \oplus \mathbf{x}_{t+k}], \tag{4.7}$$

for $k < t < T - k$ and equal to zero for all other values of t. The matrix $\mathbf{W}$ contains trainable connection weights and the operator $\oplus$ signifies concatenation. The value k determines the number of time frames that each hidden unit is a function of and σ is a parameter for the standard-deviation of the visible variables which will be assumed to be 1 in all experiments. The set of trainable parameters is given by $\theta = \{\mathbf{W}, \mathbf{b}_x, \mathbf{b}_h\}$. This model has the same energy function as the convolutional RBM (Desjardins and Bengio, 2008; Lee et al., 2009) and the main difference is the way training and inference are done.

The motivation behind this energy function is similar to that of regular RBM models. While correlations between the visible units are not directly parametrized, the hidden variables allow these correlations to be modelled implicitly. When they are not observed, they introduce dependencies among the visible units. The quadratic visual bias term serves a similar role to that of the mean of a Gaussian distribution and prevents the exponential of the negative energy function from being unbounded with respect to **X**.

To compute the free energy of a sequence **X**, one has to sum over all possible values of **H**. Fortunately, because the hidden units are binary and independent given the output of the function $\mathbf{g}_{\text{conv}}(\cdot)$, the total free energy can efficiently be calculated analytically and is given by

$$E(\mathbf{X}) = \sum_t^T \left(\frac{\|\mathbf{x}_t - \mathbf{b}_x\|^2}{2\sigma^2} - \sum_j \log\Big(1 + \exp\big(g_{\text{conv}j}(\mathbf{X}, t; \mathbf{W}) + \mathbf{b}_h\big)\Big) \right). \tag{4.8}$$

The index j points to the jth hidden unit. See Freund and Haussler (1994) for a derivation of the analytical sum over the hidden units in Equation 4.8.

The gradient of the free-energy function with respect to the function value $g_{\text{conv}j}(\mathbf{X}, t; \mathbf{W})$ is given by the negative sigmoid function:

$$\frac{\partial E(\mathbf{X})}{\partial g_{\text{conv}j}(\mathbf{X}, t; \mathbf{W})} = -\Big(1 + \exp\left(g_{\text{conv}j}(\mathbf{X}, t; \mathbf{W}) + \mathbf{b}_h\right)\Big)^{-1}. \tag{4.9}$$

The chain rule can be used to calculate the derivatives with respect to the input of the network. The derivative of the free energy with respect to the input variables is defined as

$$\frac{\partial E(\mathbf{X})}{\partial \mathbf{x}_t} = \frac{\partial E(\mathbf{X})}{\partial \mathbf{g}_{\text{conv}}(\mathbf{X}, t; \mathbf{W})} \frac{\partial \mathbf{g}_{\text{conv}}(\mathbf{X}, t; \mathbf{W})}{\partial \mathbf{x}_t} + \frac{\mathbf{x}_t - \mathbf{b}_x}{\sigma^2}. \tag{4.10}$$

4.4.2 The Recurrent Energy-Based Model

The second model I proposed, is the Recurrent Energy-Based Model (REBM). In this model, the energy is based on the dynamics that the data elicit in a non-linear dynamical system. In this case the non-linear dynamical system is instantiated as a Recurrent Neural Network (RNN).

RNNs are interesting models for time-series because they can process sequences of arbitrary length. Unlike Hidden Markov Models, RNNs have hidden states that are high dimensional distributed representations of the previous input patterns.

The energy function of the REBM is again defined by Equation 4.6 but

$\mathbf{g}_{\text{conv}}$ is replaced by

$$\begin{aligned}\mathbf{g}_{\text{rec}}(\mathbf{X}, t; \theta_{\text{RNN}}) &= \mathbf{A}\mathbf{r}_t + \mathbf{B}\mathbf{x}_t \\ \mathbf{r}_t &= \tanh(\mathbf{C}\mathbf{r}_{t-1} + \mathbf{D}\mathbf{x}_t + \mathbf{b}_r) \quad 1 < t \leq T,\end{aligned}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are matrices with network connection parameters and $\mathbf{b}_r$ is the bias vector of the recurrent state variables $\mathbf{R}$. The total set of trainable parameters for this model is given by $\theta = \{\mathbf{b}_h, \mathbf{b}_x\} \cup \theta_{\text{RNN}}$ with $\theta_{\text{RNN}} = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{b}_r\}$.

In most situations, predictions by an RNN only depend on the previous input patterns. For the REBM however, the energy function depends on the full input sequence. This allows predictions to be based on future observations as well.

This model is similar to the Recurrent Temporal Restricted Boltzmann Machine (Sutskever et al., 2008) but in the proposed model the units that define the energy are in a separate layer and the visible variables are not independent given the hidden variables. It is also similar to the Recurrent Neural Network Restricted Boltzmann Machine (Boulanger-Lewandowski et al., 2012) that will be used as a baseline in our experiments.

4.4.3 The Discriminative Temporal Boltzmann Machine

The third model is inspired by the work on Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009b) and variational inference. In this model, the hidden units are connected over time in a way that is similar to an undirected version of the factorial Hidden Markov Model (Ghahramani and Jordan, 1997). Unlike the factorial Hidden Markov Model however, the hidden variables are not just connected in temporal chains that would be independent given the input (note that this statement is only true for undirected graphs). In this model, every hidden unit at a certain time step t is connected to every hidden unit at time $t + 1$ and time $t - 1$. This allows the model to use distributed representations that are highly interconnected. The hidden units are again binary and take values from $\{-1, 1\}$. The energy of the model is defined by the following equation:

$$E(\mathbf{X}, \mathbf{H}) = \sum_{t=1}^{T} \left(\frac{\|\mathbf{x}_t - \mathbf{b}_x\|^2}{2\sigma^2} - \mathbf{h}_{t-1}^{\mathsf{T}} \mathbf{W} \mathbf{h}_t - \mathbf{h}_t^{\mathsf{T}} \mathbf{A} \mathbf{x}_t - \mathbf{h}_t^{\mathsf{T}} \mathbf{b}_h \right), \qquad (4.11)$$

where $\mathbf{h}_0$ is defined to be $\mathbf{0}$. Note that this energy function is very similar to Equation 4.6, but the convolution has been replaced with a matrix multiplication and there is an additional term that parametrizes correlations between hidden units at adjacent time steps. This model can also be seen as a Deep Boltzmann Machine in which every layer is connected to a different time step of the input sequence. Because this model has the structure of a Boltzmann Machine time-series model and will be trained in a discriminative way, I will refer to the model as the Discriminative Temporal Boltzmann Machine (DTBM). This model is very similar to the one proposed in Williams and Hinton (1991) for discrete data but the way in which I trained it is very different.

Inference in the DTBM

Since the hidden units of the DTBM are not independent from each other, I was not able to use the free energy formulation from Equation 4.8. Even when values of the visible units are all known, inference for the hidden units is still intractable. For this reason, I used variational mean-field inference to minimize the free energy. Compared to other approximate inference algorithms like loopy belief propagation, variational mean-field is relatively computationally efficient as the number of messages to compute is equal to the number of variables rather than their pairwise interactions. In the DTBM the number of pairwise interactions is very large compared to the actual number of variables.

The mean-field approximation can be motivated with ideas from variational inference. In the variational inference framework, inference is cast as an optimization problem (Wainwright and Jordan, 2008). Approximations can now be constructed by relaxing the optimization problem in some way. In the mean-field approach, the optimization is simplified by limiting the set of functions that are used to approximate the function of interest. For probabilistic models, this often means that a simple, tractable, distribution is optimized to be as close as possible to the more complicated, intractable distribution. This is done by optimizing a lower bound on the log likelihood. Optimizing this bound is equivalent to minimization of the Kullback-Leibler divergence between the approximating and target distributions. The simplest way of selecting an approximating distribution, is by dropping all dependencies between the variables. In other words, the approximating distribution takes the form of a product of one-dimensional distributions. This is commonly referred to as *naive mean-field*.

In the original training procedure for the Deep Boltzmann Machine (Salakhutdinov and Hinton, 2009b), the free energy is replaced by a variational lower-bound on the log likelihood. Given a set of known variables $\mathbf{x}$

and a set of unknown variables $\mathbf{y}$, this bound can be written as

$$\ln p(\mathbf{x}) \geq \sum_{\mathbf{y}} q(\mathbf{y}|\mathbf{x};\mathbf{u}) \ln p(\mathbf{x},\mathbf{y}) + \mathcal{H}(q), \tag{4.12}$$

where $q(\cdot)$ is an approximating distribution with parameters $\mathbf{u}$ and $\mathcal{H}(\cdot)$ is the entropy functional.

If the approximating distribution is chosen to be a fully factorized Bernoulli distribution of the form

$$q(\mathbf{H}) = \prod_{t=1}^{T} \prod_{j=1}^{N_h} q(h_{jt}|u_{jt}), \tag{4.13}$$

$$q(h_{jt} = 1|u_{jt}) = \frac{1}{2}(u_{jt} + 1), \tag{4.14}$$

$$q(h_{jt} = -1|u_{jt}) = \frac{1}{2}(1 - u_{jt}), \tag{4.15}$$

where N_h is the number of hidden units, a simple algorithm can be derived that optimizes the set $\mathbf{U} \in (0,1)^{N_h \times T}$ of variational parameters to maximize the variational bound. While the Deep Boltzmann Machine is originally trained by approximating the free energy component of the gradient with the variational method and the term of the gradient that comes from the partition function with sampling methods, I only used the variational optimization as an inference algorithm.

Because the distribution will now also be defined over some of the visible units that have been labelled as missing, the variational distribution is augmented with the appropriate number of one-dimensional Gaussian distributions of unit variance of the form $q(v|\hat{u}) = \mathcal{N}(v|\hat{u}, 1)$. The lower bound now has the following form:

$$\begin{aligned}
\ln p(\mathbf{X}_{\backslash\Omega}) \geq \mathcal{B}(\bar{\mathbf{U}}) = \sum_{t=1}^{T} \Bigg(& -\frac{\|\mathbf{x}_t - \mathbf{b}_x\|^2_{\backslash\Omega}}{2\sigma^2} + \mathbf{u}_{t-1}^{\mathsf{T}} \mathbf{W} \mathbf{u}_t + \mathbf{u}_t^{\mathsf{T}} \mathbf{b}_h \\
& - \frac{\|\hat{\mathbf{u}}_t - \mathbf{b}_x\|^2_{\Omega}}{2\sigma^2} + \sum_{k,i\in\Omega} A_{ki} u_{kt} \hat{u}_{it} + \sum_{k,i\notin\Omega} A_{ki} u_{kt} v_{it} \\
& - \sum_{j=1}^{N_h} \left(\frac{u_{jt}+1}{2} \ln \frac{u_{jt}+1}{2} + \frac{1-u_{jt}}{2} \ln \frac{1-u_{jt}}{2} \right) \Bigg) \\
& + \frac{1}{2} N_{\hat{u}} \ln(2\pi e \sigma^2) - \ln Z(\theta),
\end{aligned} \tag{4.16}$$

where Ω is the set of indices that point to the variables that have been labelled as missing, $N_{\hat{u}}$ is the number of Gaussian variables to predict (i.e., the number of missing values) and $\bar{\mathbf{U}}$ is the set of all mean field parameters

$\mathbf{U} \cup \hat{\mathbf{U}}$. Note that an upper bound on the free energy is now defined as

$$E(\mathbf{X}_{\setminus\Omega}) \leq -\mathcal{B}(\bar{\mathbf{U}}) - \ln Z(\theta) \ . \tag{4.17}$$

Optimizing this bound will lead to values of the variational parameters that approach a mode of the distribution in a similar way that a minimization of the free energy by means of gradient descent will. Setting the gradient of this bound with respect to the mean-field parameters to zero, leads to the following update equations:

$$\mathbf{u}_t \leftarrow \tanh(\mathbf{W}^\mathsf{T}\mathbf{u}_{t+1} + \mathbf{W}\mathbf{u}_{t-1} + \mathbf{b}_h + \mathbf{A}\hat{\mathbf{u}}_t) \tag{4.18}$$

$$\hat{\mathbf{u}}_t \leftarrow \mathbf{b}_x + \mathbf{A}^\mathsf{T}\mathbf{u}_t \ . \tag{4.19}$$

Note that only the variables $\hat{\mathbf{u}}$ that correspond to missing values get updated.

Ideally, the variational parameters for the hidden units should be updated in an alternating fashion. The odd units will be mutually independent given the visible variables and the even hidden variables and vice versa. Results about coordinate descent (Bertsekas, 1999) show that the algorithm is guaranteed to converge to a local minimum of the free energy surface because every individual update achieves the unique minimum for that update and the updates are linearly independent.

The model will be trained to make the variational inference updates more efficient for imputation. The naive mean-field iterations will approach a local mode of the distribution and the values of the parameters $\hat{u}$ can be directly interpreted as predictions. Note that Salakhutdinov and Larochelle (2010) proposed a method that hints in this direction by training a separate model to initialize the mean-field algorithm as well as possible. Independently, a variation of the same idea was proposed for training Deep Boltzmann Machines using a composite likelihood criterion (Goodfellow et al., 2013).

4.5 Training the Models

To train the models above, it is necessary to compute the gradient of the loss function with respect to the parameters. The loss gradients of both the models that use gradient descent inference can be computed in the same way: by doing backpropagation through the gradient descent steps. Since the DTBM uses mean-field iterations, its loss gradient is computed by backpropagating loss gradients through these mean-field updates instead.

4.5.1 Backpropagation Through Gradient Descent

To train the models that use gradient descent inference (i.e., the convolutional and recurrent models), I backpropagated loss gradients through the gradient descent steps like in Domke (2011). Given an input pattern and a set of indices that point to the missing values, a prediction was first obtained by doing K steps of gradient descent with step size λ on the free energy. Subsequently, the gradient of the mean squared error loss L, as defined in Equation 4.4 with respect to the parameters was computed by propagating errors back through the gradient descent steps in holder variables $\bar{\mathbf{X}}$ and $\bar{\theta}$ that in the end contained the gradient of the error with respect to the input variables and the parameters of the models (we use θ as a place holder for both the biases and weights of the models). Note that this procedure is similar to the backpropagation through time procedure for recurrent neural networks. The gradient with respect to the parameters was used to train the models with stochastic gradient descent. Hence, the models were trained to improve the predictions that the optimization procedure came up with directly.

Backpropagation is an application of the chain rule to propagate the error gradient back to the parameters of interest by multiplication of a series of derivatives. A single gradient descent inference step over the input variables is given by

$$\hat{\mathbf{X}}^{k+1} \leftarrow \hat{\mathbf{X}}^{k} - \lambda \nabla_{\mathbf{X}} E(\hat{\mathbf{X}}^{k}; \theta) \ . \tag{4.20}$$

By application of the chain rule, the gradient of the loss with respect to the parameters θ is given by

$$\frac{\partial L}{\partial \theta} = \sum_{k=1}^{K} \frac{\partial L}{\partial \hat{\mathbf{X}}^{k}} \frac{\partial \hat{\mathbf{X}}^{k}}{\partial \theta}. \tag{4.21}$$

Assuming a value of k that is smaller than $K - 1$ the gradient of the loss with respect to one of the intermediate states of the variables $\hat{\mathbf{X}}^{k}$ is given by

$$\frac{\partial L}{\partial \hat{\mathbf{X}}^{k}} = \frac{\partial L}{\partial \hat{\mathbf{X}}^{K}} \frac{\partial \hat{\mathbf{X}}^{K}}{\partial \hat{\mathbf{X}}^{K-1}} \cdots \frac{\partial \hat{\mathbf{X}}^{k+1}}{\partial \hat{\mathbf{X}}^{k}}. \tag{4.22}$$

To propagate errors back one needs to know $\partial \hat{\mathbf{X}}^{k+1} / \partial \hat{\mathbf{X}}^{k}$. Differentiating Equation 4.20 with respect to $\mathbf{X}$ gives

$$\frac{\partial \hat{\mathbf{X}}^{k+1}}{\partial \hat{\mathbf{X}}^{k}} = \mathbf{I} - \lambda \frac{\partial^2 E(\hat{\mathbf{X}}^{k}; \theta)}{\partial \mathbf{X} \partial \mathbf{X}^{\mathsf{T}}} \ . \tag{4.23}$$

Similarly, to complete the computation in Equation 4.21, one also needs to know $\partial\hat{\mathbf{X}}^{k+1}/\partial\theta$ to propagate the errors back to the model parameters. This partial derivative is given by

$$\frac{\partial\hat{\mathbf{X}}^{k+1}}{\partial\theta} = -\lambda\frac{\partial^2 E(\hat{\mathbf{X}}^k;\theta)}{\partial\theta\partial\mathbf{X}^\mathsf{T}} \ . \tag{4.24}$$

Since gradients of gradients are used, one needs to compute second order derivatives. Both of these second order derivatives are matrices that contain a very large number of values but fortunately there are methods to compute their product with an arbitrary vector efficiently. It is never required to explicitly store these values.

One way to compute the product of the second order derivative with a vector is by finite differences using

$$\frac{d\mathbf{f}}{d\mathbf{y}^\mathsf{T}}\mathbf{x} \approx \frac{1}{2\epsilon}\left(\mathbf{f}(\mathbf{y}+\epsilon\mathbf{x}) - \mathbf{f}(\mathbf{y}-\epsilon\mathbf{x})\right) \ . \tag{4.25}$$

The error of this approximation is $\mathcal{O}(\epsilon^2)$. Another way to compute these values is by means of automated differentiation or the $\mathcal{R}$ operator (Pearlmutter, 1994). In software for automated differentiation like Theano (Bergstra et al., 2010), the required products can be computed relatively efficiently by taking the inner product of the gradient of the energy with respect to the input and the vector to obtain a new cost value. The automated differentiation software can now be used to differentiate this new function again with respect to the input and the parameters.

I found that, even when applied recursively for three steps in single precision floating point arithmetic, the finite differences approximation was very accurate. For the CEBM, the mean squared error between the finite differences based loss gradients and the exact gradients was of order 10^{-3}, while the variance of the gradients was of order 10^2. In preliminary experiments, I didn't see any effect of the approximation on the actual performance for this model.

Since the finite difference approximation was generally faster than the exact computation of the second order derivatives, I used it for most of the required quantities. For the REBM I used finite differences with $\epsilon = 10^{-7}$ in double precision. For the CEBM I found that automatic differentiation for the second order derivative with respect to the parameters was faster so for this quantity I used this method instead of finite differences. The CEBM computations were done on a Nvidia Geforce GPU card so I had to use single precision with $\epsilon = 10^{-4}$.

See Algorithm 4 for more details about the backpropagation through gradient descent procedure. In the algorithm I omitted the missing value

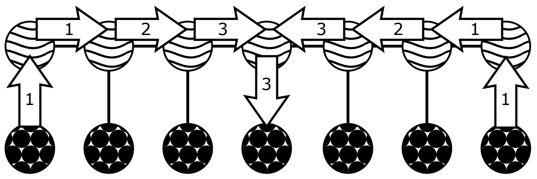


Figure 4.2: Flow of information due to mean-field updates. The numbers represent the separate iterations at which both the hidden units and the unknown visible units are updated.

indicator indices Ω that determine which variables should be updated at every iteration.

Algorithm 4 Compute the error gradient through gradient descent

```
Initialize $\hat{\mathbf{X}}^0$
**for** $k = 0$ **to** $K - 1$ **do**
    $\hat{\mathbf{X}}^{k+1} \leftarrow \hat{\mathbf{X}}^k - \lambda \nabla_{\hat{\mathbf{X}}} E(\hat{\mathbf{X}}^k; \theta)$
**end for**
$\bar{\mathbf{X}}^K \leftarrow \nabla L(\hat{\mathbf{X}}^K) = \hat{\mathbf{X}}^K - \mathbf{X}$
$\bar{\theta} \leftarrow \mathbf{0}$
**for** $k = K - 1$ **to** $0$ **do**
    $\bar{\theta} \leftarrow \bar{\theta} - \lambda \frac{\partial^2 E(\hat{\mathbf{X}}^k;\theta)}{\partial \theta \partial \mathbf{X}^\mathsf{T}} \bar{\mathbf{X}}^{k+1}$
    $\bar{\mathbf{X}}^k \leftarrow \bar{\mathbf{X}}^{k+1} - \lambda \frac{\partial^2 E(\hat{\mathbf{X}}^k;\theta)}{\partial \mathbf{X} \partial \mathbf{X}^\mathsf{T}} \bar{\mathbf{X}}^{k+1}$
**end for**
Return $\nabla_\theta L = \bar{\theta}$
```

4.5.2 Backpropagation Through Mean-Field

Computing error gradients for the mean-field based model is possible by backpropagating errors through the mean-field updates. Intuitively, this model can be seen as a bi-directional recurrent neural network with tied weights. The derivatives of the update Equations 4.18 and 4.19 are easy to compute using the derivatives of the hyperbolic tangent function and linear operations:

$$\frac{\partial \tanh(\mathbf{Wz})}{\partial \mathbf{z}} = \mathbf{W}^\mathsf{T}(1 - \tanh(\mathbf{Wz})^2)\ . \tag{4.26}$$

However, to make it easy to experiment with the number of mean-field iterations, I used automated differentiation for this.

While the number of gradient descent steps for the convolutional model and the recurrent neural network model can be very low, the number of mean-field iterations has a more profound influence on the behavior of the model. This is because the number of mean-field iterations directly determines the amount of context information that is available to guide the prediction of the missing values. If for example, five iterations of mean-field are used, the activation in the hidden variables at a certain time frame $\mathbf{h}_t$ will be influenced by the five frames of visible variables to the left and right of it and by the known values of $\mathbf{x}_t$; every vector of hidden units now depends on eleven frames of visible units. So a greater number of mean-field iterations increases the range of the dependencies the model can capture. This flow of information is displayed for three iterations in Figure 4.2.

Using backpropagation through variational optimization updates is referred to as variational mode learning (Tappen, 2007). In Tappen (2007), Fields of Experts were trained by backpropagating loss function gradients through variational updates that minimized a quadratic upper bound of the loss function. This specific approach would not work for the type of model I defined here.

4.6 Imputation Experiments

I did experiments on three datasets. The first dataset consisted of concatenated handwritten digits, the second dataset contained marker positions from motion capture recordings and the third dataset sensor readings from a mobile robot. The last experiment also investigated the robustness of the models when there were not only missing values in the test set but also in the train data. To compare my approach with generative methods, I also trained a Convolutional RBM with the same energy function as the Convolutional Energy-Based Model for all these datasets. Between these two models, the training method is the only thing that makes them different. Furthermore, I also trained a Recurrent Neural Network Restricted Boltzmann Machine (RNN-RBM; Boulanger-Lewandowski et al., 2012). This model is quite similar to the REBM as it also employs separate sets of deterministic recurrent units and stochastic hidden units. The structure of the RNN-RBM is the same as the architecture in Figure 4.1b but the hidden units are connected to the next time step instead of the current one and those connections are undirected. This makes it possible to use Contrastive Divergence learning but also renders the model unable to incorporate future information during inference because the information from the recurrent units is considered to be fixed. Training a recurrent model that incorporates this kind of information with Contrastive Divergence would require something like Hybrid Monte Carlo (Duane et al., 1987). Unfortunately, Hybrid Monte Carlo requires careful tuning of the number and size of leap frog steps and is known to be less efficient than block Gibbs sampling.

For the Energy-Based models I used the same inference procedure as during training. For the Convolutional RBM and RNN-RBM I used mean-field iterations that were run until the MSE changed less than a threshold of 10^{-5}. For the RNN-RBM this was done by initializing the missing values of a single time step with the values of the corresponding dimensions at the previous time-step, running mean-field to update them, passing these new values through the recurrent neural network and repeating this procedure for all remaining time-steps. This procedure proved to be more accurate than Gibbs sampling which was originally used to generate sequences with this model (Boulanger-Lewandowski et al., 2012).

4.6.1 Concatenated Handwritten Digits

To provide a qualitative assessment of the reconstruction abilities of the models, I used the USPS handwritten digits data set (`http://www.cs.nyu.edu/~roweis/data/usps_all.mat`). While the task we trained the models for is of little practical use, visual inspection of the reconstructions allows for an evaluation of the results that may be more insightful than just the mean squared error with respect to the ground truth.

Data

The USPS digits data set contains 8-bit grayscale 28×28 pixel images of the numbers '0' through '9'. Each class has 1100 examples. To turn the data into a time-series task, I randomly permuted the order of the digits and concatenated them horizontally. This sequence of 28 dimensional vectors was split into a train set, a validation set and a test set that consisted of respectively 80% and two times 10% of the data.

Training

Good settings of the hyper-parameters of the models were found with a random search[1] over 500 points in the parameter space, followed by some manual fine-tuning to find the settings that led to minimal error on the validation set. After this, the models were trained again on both the train and the validation data with these settings. Since the CEBM, DTBM and

[1]The set of hyper-parameters included the variances of the Gaussian distributions from which initial weight matrices were sampled, initial learning rates. The numbers of units in each layer were searched over in multiples of 50 up till 300 units.

Table 4.1: Parameter settings for training the models on the USPS data. The learning rates were always divided by the lengths of the sequences during training.

	Inference iterations	Learning rate	Hidden units
CEBM	3	0.005	300
REBM	5	0.001	50
DTBM	15	0.0005	500
Conv. RBM	N/A	0.0005	300
RNN-RBM	N/A	1.8	100

Convolutional RBM are very parallel in nature, I used a GPU for their simulations. All models were trained for 100,000 epochs on randomly selected mini batches of 100 frames. Linearly decaying learning rates were used. Table 4.1 shows the settings of the hyper-parameters. After some preliminary experimentation I found that I got better results for the CEBM by initializing the biases of the hidden units at -2 to promote sparsity. I used a step size of .03 for the gradient descent inference algorithm of the CEBM. The REBM had 200 recurrent units and I used a step size of .2 for the gradient descent inference. The CEBM and the Convolutional RBM had a window size of 7 time frames. The RNN-RBM had 200 recurrent units. Note that the best RNN-RBMs also had more hidden units than the best performing REBMs.

The Convolutional RBM was trained with the Contrastive Divergence algorithm (CD; Hinton, 2002). I found that I got better results with this model if I increased the number of CD sampling steps over time. Initially a single CD step was used. After 20,000 iterations this number was increased to 5, after 50,000 to 10 and after 75,000 to 20. I did not find any benefits from stimulating sparsity in this model. For the RNN-RBM I found that a fixed number of 5 CD sampling steps gave the best results.

Missing values were generated as 20 square shaped gaps in every data sequence. The gaps were positioned at uniformly sampled locations. The size of each square was uniformly sampled from the set $\{1, 2, \ldots, 8\}$. Figure 4.3a shows an example of missing values that were generated this way for six sequences. Figure 4.3c shows an example of data that has been corrupted by this pattern of missing values.

For this experiment, the Energy-Based Models that required missing values during training were provided with missing values from the same dis-

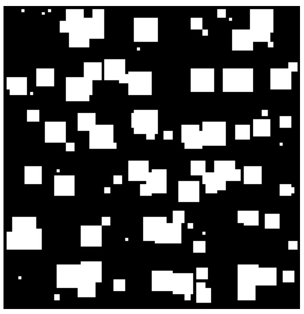
Table 4.2: Means and standard deviations of the results in mean squared error on the USPS data.

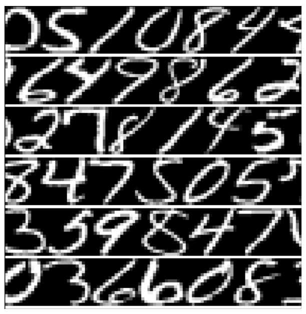
	Train	Test
CEBM	0.48(0.0082)	0.49(0.0089)
REBM	0.47(0.01)	0.48(0.0067)
DTBM	0.44(0.011)	**0.45**(0.0088)
Conv. RBM	0.66(0.011)	0.66(0.0085)
RNN-RBM	0.71(0.0056)	0.73(0.0062)

tribution that was used during training. To control for the random initialization of the parameters and the randomness induced by stochastic gradient descent, I repeated every experiment 10 times.

Results

Quantitatively, the DTBM achieved the best performance as can be seen in Table 4.2. The CEBM and REBM performed on a similar level, while the Convolutional RBM and the RNN-RBM performed far worse. Figure 4.4 shows how the models reconstructed six damaged sequences from the test data. The reconstructions by the Convolutional RBM in Figure 4.4d seem to be of a lower quality than those from the other models and look more blurry. This is consistent with the MSE scores. However, just looking at the MSE scores does not seem to give the full picture as the reconstructions of the CEBM look more smeared out and blurry than those of the REBM even though the MSE scores of these models are similar. The DTBM provided reconstructions that look similar in quality to those of the REBM. The RNN-RBM has a very bad MSE score while its reconstructions look very sharp when they are correct. Somehow, this model seems to be too sure in cases where it is not able to fill in the data well. This may be due to the fact that the reconstructions are generated on a frame-by frame basis while the deterministic models and the Convolutional RBM can update their predictions from previous iterations. This causes bad predictions to be propagated and possibly amplified. Unfortunately this problem is intrinsic to the choice to keep sampling tractable by treating the recurrent mapping as fixed context information.


(a) Mask


(b) Original data

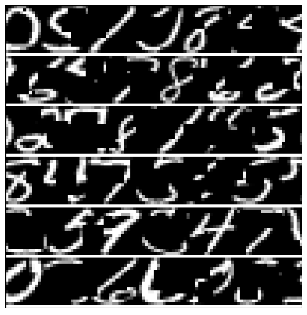
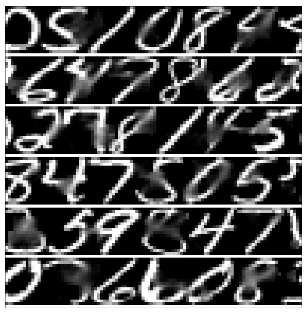
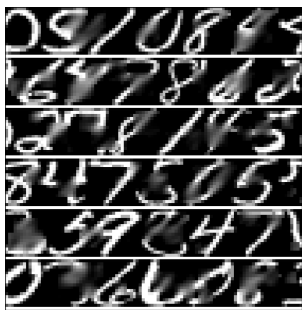

(c) Corrupted data

Figure 4.3: A visual depiction of the handwriting imputation task.


(a) CEBM

(b) REBM

(c) DTBM


(d) Conv. RBM

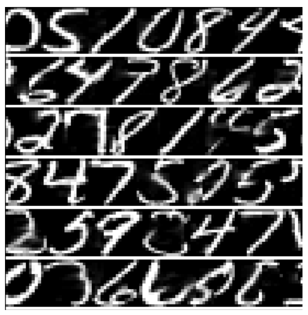

(e) RNN-RBM

Figure 4.4: Visual representations of the reconstruction of six sequences of handwritten digits. The reconstructions are produced by the CEBM, the REBM, the DTBM, the Convolutional RBM and the RNN-RBM.

4.6.2 Motion Capture

For the second experiment I applied the models to a motion capture dataset. In visual motion capture, a camera records movements of a person wearing a special suit with bright markers attached to it. These markers are later used to link the movements to a skeleton model. Missing values can occur due to lightning effects or occlusion. Motion capture data is high dimensional and generated by a non-linear process. This makes motion capture reconstruction an interesting task for evaluating more complex models for missing value imputation.

In previous work (Taylor et al., 2007), a Conditional Restricted Boltzmann Machine (CRBM) was trained to impute missing values in motion capture as well. For comparison I also trained a CRBM but note that a direct comparison of performance is not fair because the CRBM only uses information from a fixed number of previous frames to make predictions.

Data

The data consisted of three motion capture recordings from 17 marker positions represented as three 49-dimensional sequences of joint angles. The data was down sampled to 30Hz and the sequences consisted of 3128, 438, and 260 frames. The first sequence was used for training, the second for validation and the third for testing. The sequences were derived from a subject who was walking and turning and come from the MIT dataset provided by Eugene Hsu.[2] The data was preprocessed by Graham Taylor (Taylor et al., 2007) using parts of Neil Lawrence's Motion Capture Toolbox.

Training

Again, good settings of the hyper-parameters of the models were found with a random search on the parameter space, followed by some manual fine-tuning to find the settings that led to minimal error on the validation set. The models were trained on mini batches of 140 frames. Table 4.3 shows the hyper-parameter settings of the models. Note that the best RNN-RBMs had again more hidden units than the best REBMs. Additionally, the inference step sizes of the CEBM and the REBM were both set to .2. The CEBM and the Convolutional RBM had a window size of 15 time frames. All models were trained for $50,000$ iterations. The REBM had 200 recurrent units. To train the Convolutional RBM, single iteration CD training was used during the first $10,000$ epochs. Five iterations of CD were used during the remaining training epochs. The CRBM was trained with single iteration

[2] `http://people.csail.mit.edu/ehsu/work/sig05stf/`

Table 4.3: Parameter settings for training the models on the motion capture data.

	Inference iterations	Learning rate	Hidden units
CEBM	3	0.5	200
REBM	5	0.001	50
DTBM	10	0.002	200
CRBM	N/A	0.0001	200
Conv. RBM	N/A	0.001	200
RNN-RBM	N/A	2.14	100

CD. Finally, the RNN-RBM had 200 recurrent units and was trained with 5 CD iterations. All models were trained for a total of $50,000$ epochs.

The CEBM, REBM and DTBM were again trained by labeling random sets of dimensions as missing. The number of missing dimensions was sampled uniformly. The specific dimensions were then randomly selected without replacement. The duration of the data loss was sampled uniformly between 60 and 125 frames. This adds some bias towards situations in which the same dimensions are missing for a certain duration. This seems to be a sensible assumption in the case of motion capture data and it is an advantage of the training method that it is possible to add this kind of information. To control for the influence of the randomly initialization of the parameters, I repeated every experiment 10 times.

Finally, Nearest Neighbour interpolation was performed by selecting the frame from the train set with minimal Euclidean distance to the test frame according to the observed dimensions. The distances were computed in the normalized joint angle space.

Evaluation

To evaluate the models, a set of dimensions was removed from the test data for a duration of 120 frames (4 seconds). This was done for either the markers of the left leg or the markers of the whole upper body (everything above the hip). Because the offset of this gap was chosen randomly and because the CRBM had a stochastic inference procedure, this process was repeated 500 times to obtain average mean squared error values for both the train and the test data. Note that this distribution of missing values was quite different from the one that was used during training.

Table 4.4: Means and standard deviations of the results in mean squared error on the motion capture data.

	Left leg		Upper body	
	Train	Test	Train	Test
CEBM	0.18(0.0036)	0.29(0.011)	0.47(0.023)	0.46(0.017)
REBM	0.22(0.0047)	0.33(0.017)	0.47(0.014)	**0.42**(0.014)
DTBM	0.17(0.0069)	**0.28**(0.017)	0.43(0.021)	0.45(0.011)
CRBM	0.25(0.0036)	0.44(0.014)	0.51(0.023)	0.49(0.0065)
Conv. RBM	0.16(0.0038)	0.36(0.027)	0.48(0.015)	0.68(0.035)
RNN-RBM	0.18(0.0055)	0.36(0.023)	0.45(0.017)	0.60(0.055)
Nearest neighb.	N/A	0.45	N/A	0.76

The CRBM was used in a generative way by conditioning it on the samples it generated at the previous time steps while clamping the observed values and only sampling those that were missing as was done in Taylor et al. (2007). Preliminary results showed that this led to similar results to the use of mean-field or minimization of the model's free energy to do inference.

Results

Table 4.4 shows the mean squared error between the reconstructed dimensions of the data and their actual values. The convolutional and recurrent models clearly outperform the CRBM and nearest neighbour interpolation on the reconstruction of the left leg. The CEBM and the DTBM have the best performance but a comparison of the train and test error scores suggests that the REBM might display better generalization properties. The CRBM seems to suffer most from overfitting. The results of the Convolutional RBM are only slightly worse than the CEBM and better than those of the REBM for the reconstruction of the left leg, but far worse for the upper body where a greater number of variables were missing. The RNN-RBM performed similar to the Convolutional RBM when reconstructing the markers of the missing leg. It performed slightly better at reconstructing the missing upper body than the Convolutional RBM but still a lot worse than the three deterministic models. Figure 4.5 shows plots of the predictions made by the DTBM for two of the markers for a sequence from the test data.

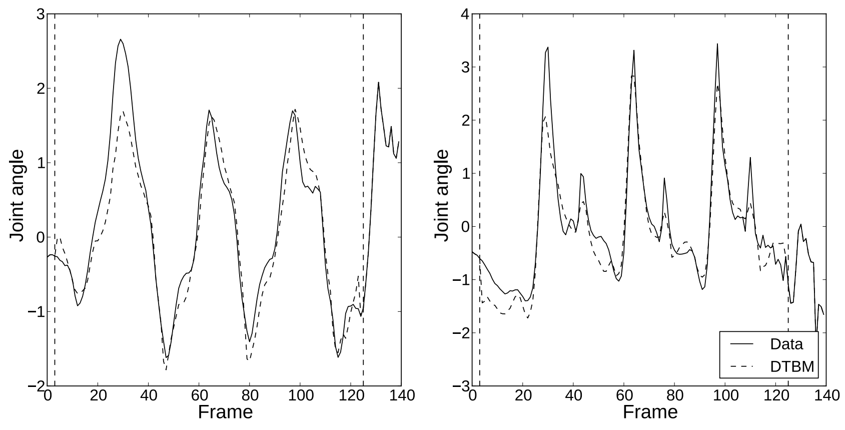


Figure 4.5: Plots of two dimensions reconstructed by the DTBM next to the actual data. For this sequence the markers of the left leg were missing in the region between the vertical striped lines.

4.6.3 Missing Training Data

So far, all experiments were done by training on data without actual missing values; values were only truly unknown during testing. In practice, a useful model for missing value imputation should also be able to deal with actual missing values in the train set. For generative models, missing values in the train data shouldn't pose a problem because they can be marginalized out. For intractable models like RBMs however, this marginalization can easily become infeasible. For the models I proposed, missing values in the train data are easily dealt with. During training, the energy is optimized with respect to the true missing variables for which no ground truth value is available and artificial additional missing values. In other words, the models will try to predict the true missing values during training. The loss however, is only computed for the artificially created missing values for which the actual value is known. A very similar method has been used in earlier work to train neural networks for classification when missing values are present (Bengio and Gingras, 1996).

To see how well the models deal with missing training data, I conducted an additional series of experiments.

Data

The data I used to investigate the effect of missing training data consists of the measurements of the 24 ultrasound sensors of a SCITOS G5 robot

Table 4.5: Parameter settings for training the models on the robot data.

	Inference iterations	Learning rate	Hidden units
CEBM	3	0.65	300
REBM	3	0.0007	100
DTBM	5	0.002	300
Conv. RBM	N/A	0.00012	300
RNN-RBM	N/A	0.15	50

navigating a room (Freire et al., 2009; Frank and Asuncion, 2010). The 5456 sensor readings were sampled at a rate of 9Hz and the robot was following the wall of the room in a clockwise direction, making four trips around the room. I used 80% of the data for training and split the remaining 20% up in a validation set and a test set.

Training

In the first experiment, I trained the models on fully intact training data to get an estimate of the optimal performance the models could achieve on it. In this experiment I also compare the results with those of the Convolutional RBM and the RNN-RBM. In the second experiment, I generated a mask for the whole train set. The mask was divided in regions of 100 frames and at each of these regions a randomly selected set of dimensions was labelled as missing. To train the models, I selected random batches of 100 frames from the train data and selected another set of variables as missing that were not already truly missing in the data. This way, the models never had access to the values that were labelled as missing by the training data mask. To investigate the robustness of the models, I varied the amount of data that was damaged in the train set. In both experiments, the number of dimensions that I pretended to be missing in order to train the models was uniformly sampled from $\{1, \dots, 5\}$. All models were trained for $100,000$ epochs.

The hyper-parameter settings were obtained in the same way as in the previous experiments and are displayed in Table 4.5. Additionally, the Convolutional RBM and CEBM had a window size of 5 and the REBM had 200 recurrent units. The step sizes for the CEBM and REBM were respectively .016 and .7. The CEBM and the Convolutional RBM had a window size of 5

Table 4.6: Results in mean squared error on the wall robot data without missing dimensions during training.

	Train	Test
CEBM	0.27	0.43
REBM	0.41	0.39
DTBM	0.29	**0.34**
Conv. RBM	0.53	0.47
RNN-RBM	0.38	0.54

time frames. The Convolutional RBM was trained with the same Contrastive Divergence scheme as in the handwritten digits experiment. The RNN-RBM had 250 recurrent units and was trained with 5 iterations of Contrastive Divergence. Somehow the best performing RNN-RBMs had this time fewer hidden units than the best performing REBMs. I used these settings for all the experiments, regardless of the number of missing training dimensions.

Results

Table 4.6 shows the results for the experiment without missing train data. The DTBM had the best performance, followed by the REBM, the CEBM, the Convolutional RBM, and the RNN-RBM respectively. Surprisingly, the CEBM has a far better score on the train data than the REBM while its test error is higher. This seems to be a sign of overfitting. The REBM is probably slightly underfitting as its training error was even a little bit higher than its test error. The RNN-RBM obtained a slightly better score on the train data than the REBM but performed a lot worse on the test set.

Figure 4.6 shows the error scores for the three energy-based models as a function of the number of missing input dimensions. Obviously, as more dimensions are missing, there is less data available to train on and the error scores of all the models are rising. For the DTBM and REBM, the error seems to increase at a modest pace initially and when just a single value is missing, the performance is similar to the first experiment. The CEBM has more trouble with missing values and is not learning to reconstruct the data well any more after about 10 dimensions are missing. I interpret these results as an upper bound on the error scores one could potentially achieve by doing more extensive hyper-parameter tuning for each individual level of data corruption.

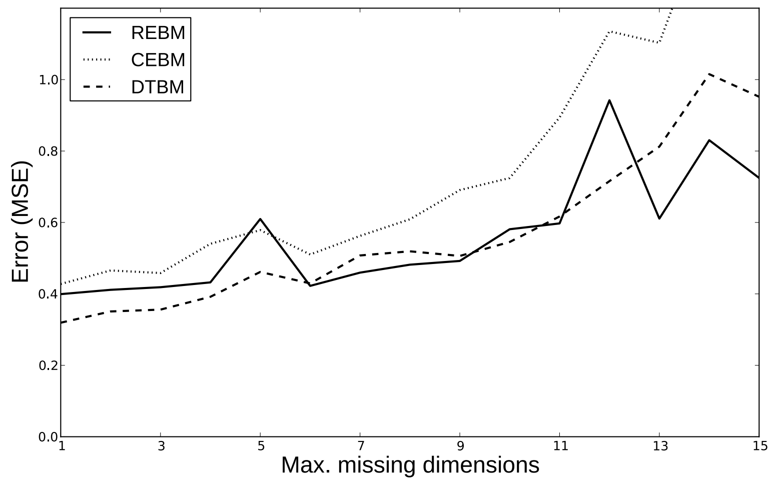


Figure 4.6: Mean square error as a function of the size of the interval from which the number of missing train data dimensions was sampled.

All in all, the training methods seem to handle missing training data quite well until

4.7 Imputation and Denoising for Training Generative Models

While the imputation models of this chapter are not trained as generative models, it is clear that they still need to learn some properties of the data to perform well. Even though the motivation for the training procedure was to perform imputation without the need for maximum likelihood learning, it is interesting to learn more about the potential overlap of the two methods. This may also provide more insight into the sensitivity of the models to a misspecification of the corruption process that selects the missing values.

First of all, Goodfellow et al. (2013) trained Deep Bolzmann Machines with a method that turns out to be nearly identical to the training algorithm of the Discriminative Temporal Boltzmann Machine. Their goal however, was to use the model for other tasks than imputation and they arrived at the method from the motivation to optimize a variational version of the so-called composite likelihood (see Section 3.1 and Lindsay, 1988). Pseudo likelihood and some other variants of the more general composite likelihood, have the interesting property that they are consistent estimators. However, when the composite likelihood and the standard likelihood have not been fully optimized, they tend to behave quite differently. The composite likelihood tends to converge faster to models that perform well at conditional queries of the variables, as is the case in missing value imputation. Optimization of the true likelihood, on the other hand, tends to lead faster to models that provide samples that look similar to the data they have been trained on (Goodfellow et al., 2013).

Secondly, the imputation models can be seen as so-called Generative Stochastic Networks (GSN; Bengio and Thibodeau-Laufer, 2013). Unlike most conventional parametric generative models, GSNs are not trained to learn the data density directly, but to learn a transition operator of a Markov chain that has the data density as its ergodic distribution. GSNs do this by learning the shape of the data distribution locally, conditioned on variables that are sampled from a certain noise distribution. This means that many denoising and imputation methods can be used to construct GSNs. It was shown that when models are trained with variational composite likelihood, they are able to provide better looking samples of the data when they are employed as GSNs rather than as standard density models. As pointed out by Yoshua Bengio during personal communication, the models I trained for imputation can also be used as GSNs to generate samples.

Drawing samples from a GSN is done by running a Markov chain that samples from the noise distribution and the denoising distribution in an alternating fashion. This works in the following way: First, the chain is initialized at some random location. Subsequently, this 'sample' is corrupted by the noise distribution. Finally, the model provides a denoised sample conditioned on the noisy input. These last two steps are repeated to run a Markov chain. If the denoising distribution is accurate enough, this Markov chain converges to an equilibrium distribution that should be close to the data distribution. Formally, if the denoising model provides a distribution $p(\mathbf{x}|\hat{\mathbf{x}})$, where $\hat{\mathbf{x}}$ is the corrupted data, created by a corruption process $q(\hat{\mathbf{x}}|\mathbf{x})$ (I used a different symbol to emphasize the fact that these conditional distributions don't need to match), the algorithm is defined by the following steps:

1. Start with some arbitrary value of $\mathbf{x}_t$.
2. Sample $\hat{\mathbf{x}}_\mathbf{t}$ from the corruption process $q(\hat{\mathbf{x}}_t|\mathbf{x}_t)$.
3. Sample $\mathbf{x}_{t+1}$ from $p(\mathbf{x}_{t+1}|\mathbf{x}_t)$.
4. Repeat steps 2 and 3 to obtain as many samples as desired.

To investigate whether the imputation method is able to learn a good

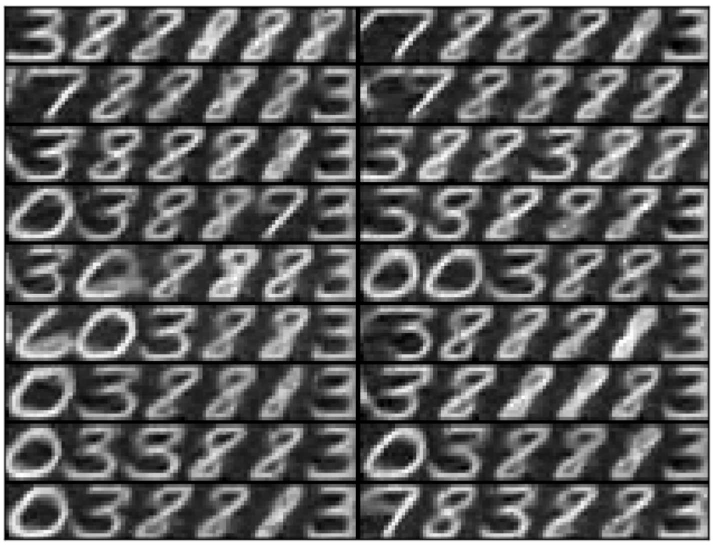


Figure 4.7: A collection of samples generated from the REBM model using the GSN sampling method. The samples are sorted in a column major fashion.

GSN, I drew samples from an REBM trained[3] with the settings that performed best on the handwritten digits task according to the GSN principle. The REBM provided the denoising distribution $p(\mathbf{x}|\hat{\mathbf{x}}, \Omega)$, conditioned on both the corrupted data and the locations of the missing values; the corruption process was simply the same as the one used during training, selecting squares of varying size to be marked as missing. Since the model was not trained to predict the variance of the distribution $p(\mathbf{x}|\hat{\mathbf{x}}, \Omega)$, I performed deterministic updates for the denoising process. I took $100,000$ MCMC steps and stored a sample after every 5000 updates.

Figure 4.7 displays 18 of the samples created with the GSN process. The process does not seem to mix very well. This is probably due to the deterministic updating and the relatively small proportion of the variables that is reconstructed at each step. However, the model does indeed generate

[3]A new model had to be trained because some of the input variables were never selected by the corruption process. For evaluation of the imputation performance this was no issue, but a GSN needs a corruption process that reaches all the variables.

samples that look like the data it was trained on. Moreover, the samples suggest that the model is invariant to horizontal (i.e., temporal) shifts of the data because samples that are relatively near each other in the sampling chain often contain very similar symbols that have been shifted a couple of pixels to the left or the right. Unfortunately, the model seems biased to generate the same patterns near the left and right edges of the sequences.

4.8 Discussion

In all the experiments I did, the discriminatively trained models outperformed the baseline methods. The most interesting result is that the CEBM often performed much better than the Convolutional RBM. This indicates that the proposed training strategy is more suited for missing value imputation than the Contrastive Divergence algorithm. The REBM also outperformed the RNN-RBM. While this comparison is less valid because the two models are not entirely the same, it still suggests that our training method also works better for recurrent architectures. The DTBM outperformed all other methods in all of the experiments except for reconstructing the upper body markers of the motion capture dataset where the REBM performed better. Given that it is also the easiest model to implement and generally (depending on the required context size) more computationally efficient than the REBM, this model seems to be the most promising candidate for practical applications.

I didn't show that my method outperforms maximum likelihood learning because Contrastive Divergence is not optimizing the actual likelihood of the model. However, it seems to be the most popular method for training these kind of models when true maximum likelihood learning is intractable. I actually suspect that, compared to other approximate maximum likelihood methods, Contrastive Divergence is still one of the best candidates for missing value imputation because it optimizes the energy landscape locally by pushing up wrong predictions that are near the data itself. When the number of missing values is not too large, inference would start near one of the regions in which the energy landscape has a shape that promotes good predictions. When the number of missing values is too high however, the inference algorithm might start in a region that was not explicitly shaped by the learning algorithm because it was too far away from the data. This might explain the bad performance of the Convolutional RBM when the whole upper body was missing in the motion capture task. The CRBM and (to a lesser extent) the RNN-RBM were more robust in this situation.

Finally, I also showed that it is possible to use at least one of the models in a generative setup as well by using it as a Generative Stochastic Network. While the samples looked decent, the mixing seemed to be rather slow. This is probably because of the relatively small number of variables that is reconstructed at each sampling step. The fact that the model had no parameters for modelling the variance is likely to have been detrimental as well but this was obviously no real priority for solving the original imputation tasks the model was optimized for.

4.8.1 Relation to Other Work

The idea of training Markov Random Fields in a discriminative way by using a simple deterministic inference procedure is not new and has been used in image and natural language processing. In image processing, the inpainting (Bertalmio et al., 2000) or denoising (Barbu, 2009) of pictures are thoroughly studied problems. Barbu (2009) proposed Active Random Fields for denoising images. Active random fields are Fields of Experts (Roth and Black, 2005) that are trained by doing inference with a couple of iterations of gradient descent. The model parameters are optimized to make the gradient descent inference procedure more efficient. In a more recent paper (Domke, 2012), this approach was extended to more advanced optimization methods like heavy ball (Polyak, 1964) and BFGS. In a similar fashion, gradients have been computed through message passing to train Conditional Random Fields more efficiently (Domke, 2011; Stoyanov et al., 2011). I extended their approaches to models for time-series and missing value imputation. To my knowledge, models that were used for image inpainting were either trained in a probabilistic way, or to do denoising. I showed that models can be trained for imputation directly and that the approach is not limited to gradient based optimization or loopy belief propagation. It can also be applied to models for which approximate inference is done using the mean-field method. Furthermore, I showed that quite complex models with recurrent dependencies, that would be very difficult to train as probabilistic models, can be learned this way.

This work is also related to the training of Dynamical Factor Graphs (DFG; Mirowski and LeCun, 2009). DFGs are used as generative models, but the way they are trained is more similar to the energy-based learning framework. The types of DFGs that have been studied so far are similar to the REBM model in that they employ both recurrent connections and latent variables. An important difference is that in the REBM the latent variables are not involved in the recurrent part of the model. By constraining the latent states of a DFG to operate under Gaussian noise with fixed covariance, the partition function of the model becomes constant so that a minimization of the energy for a certain state will automatically push up the energy values of all other possible states. The model is not fully probabilistic as it aims to align its maximum a posteriori hidden states with the observed data instead of the sum over all their possible values.

Inference for missing values in DFGs is done with gradient descent to find the minimum energy latent state sequence, ignoring the missing values in the gradient computations. Subsequently, the missing values are predicted as a function of this latent state sequence. There are a couple of important differences with my approach:

- The models I proposed don't aim to provide generative models of the data.
- The CEBM and REBM models use only a limited number of gradient descent updates while for DFGs this minimization is run until convergence.
- In the CEBM and REBM, energy minimization only takes place with respect to the missing values and not with respect to the latent variables which are marginalized out analytically.

It seems that some of the ideas behind DFGs are orthogonal to the proposed approaches and may be combined to design new interesting models.

Recently, a generative model called NADE (Larochelle and Murray, 2011) was proposed in which the mean-field algorithm inspired a model similar to the Restricted Boltzmann Machine (Freund and Haussler, 1994; Hinton, 2002), but with a tractable inference algorithm. In this model, a single iteration of mean-field is used to approximate a set of conditional distributions that are combined into a generative model. My mean-field based model is substantially different in that it doesn't try to learn a joint distribution over all the variables and that mean-field is also used to estimate the influence of connections between the hidden variables. NADE has been combined with a recurrent neural network to construct a sequential model for note patterns in music (Boulanger-Lewandowski et al., 2012). This model is practically the same as the RNN-RBM I used as a baseline but with the RBM replaced by the NADE model.

Adding artificial corruption to the data and training a model to reconstruct it is similar to the way denoising autoencoders are trained (Vincent et al., 2008). An important difference with traditional denoising autoencoders that the proposed models only focus on recovery of the missing values given the observed variables and not on reconstructing both. Of course, autoencoders could also be parameterized to perform a 1-to-1 mapping for the uncorrupted input variables.

The way the DTBM model dealt with missing training data in Section 4.6.3, is very similar to a method for dealing with missing values in neural networks that was presented more than a decade ago (Bengio and Gingras, 1996). In this method, missing values are also filled in by updating them as if they are part of a recurrent neural network. An important difference with my work is that in Bengio and Gingras (1996) the goal was not to predict the missing values themselves, but to perform better on classification tasks when missing values in the inputs are present.

It turns out that the architecture of the DTBM has also been proposed for learning to discriminate between sequences (Williams and Hinton, 1991). In this work, a version of the mean field algorithm that is inspired by simulated annealing is ran until convergence and the training algorithm aims to maximize the lower bound on the likelihood for one class of sequences, while lowering it for the remaining classes. Running mean-field iterations until convergence can make training very inefficient so it might be interesting to see how backpropagation through mean-field performs for tasks of this type.

4.8.2 Limitations

A downside of my models is that they can take quite long to train. For the CEBM and the DTBM, this problem can be alleviated by using GPU parallelization but unfortunately this is not possible for the REBM. It would probably not have been feasible to train this model with a more generic approach in which the energy optimization would have to be executed until convergence for every training sample during training. However, once training is done, inference for new sequences of data is quite fast. The models took far less time to evaluate than the Convolutional RBM for which mean-field had to be run until convergence to make predictions.

Another downside of my approach is that it introduces new hyper-parameters to tune. These are the number of inference iterations and for the CEBM and the REBM also the step size of the gradient descent algorithm. Because of this, I found it easier to find good settings for the DTBM than for the CEBM and REBM. This problem could be solved to some extent by using an optimizer for inference in the CEBM and the REBM that doesn't need a step size parameter. LBFGS has been shown to work quite well for optimization based inference (Domke, 2011) but is a lot more complicated to implement.

The REBM was the most difficult model to train even though it sometimes performed better than the CEBM. It seems to be more sensitive to the initial hyper-parameter settings than the other models. As more hidden units were used, these models became more difficult to optimize and this explains why the optimal number of hidden units was generally lower than for the other models. I think this problem occurs because the energy gradient is computed with backpropagation through time over relatively long sequences. The gradients of recurrent neural networks are known to be prone to exponential growth or decay and a single bad gradient can lead to a divergence of the learning algorithm.[4] Larger numbers of hidden units tend to lead to larger values in the gradients, making problematic weight updates more frequent. In preliminary results I found that this problem can be solved to some extent by normalizing the loss gradient before updating the weights. The DTBM might suffer less from this problem because the number of backpropagation steps that are used is smaller.

Since the number of mean-field iterations of the DTBM is directly related to the length of the temporal dependencies it can model, it might not be very suitable for problems in which these dependencies are very long. That being said, this model benefits a lot from parallel computing machinery like GPUs. As the performance of parallel computing hardware increases, the computational time required to model dependencies that are as long as the sequence itself might become more comparable to simulating a regular recurrent neural network. It remains to be seen what kind of effect this has on the quality of the computed gradients.

[4]There are some remedies to this problem like clipping gradient values that become too large (Pascanu et al., 2013a).

5 Speech Enhancement

Speech processing (and audio processing in general) is an interesting domain for investigating complex temporal machine learning models. First, audio data is temporal, which makes it less straightforward to process than data that consists of instances of a fixed size. Second, audio recordings can contain multiple sources of both signals of interest and noise. Speech is interesting in particular because the temporal structure of the audio signal is also determined by the language involved, the voice of the speaker, and emotional content of the utterance among many other factors. Finally, there are many practical applications for speech processing like speaker identification, speech synthesis and speech recognition. In this chapter I will describe how some of the ideas from the previous chapters can be used for feature enhancement applications in speech recognition.

5.1 Automatic Speech Recognition

Automatic Speech Recognition (ASR), is the task of translating spoken language into text. ASR systems are becoming more and more ubiquitous due the advent of mobile devices and increasing computing power. Speech recognition is a complicated task that requires both knowledge about language and advanced audio processing abilities. Most contemporary ASR systems consist of various modules that can often be split into *acoustic models* and *language* models. During processing, these modules work together to predict the final sequence of words.

Language models provide an ASR system with predictions for words given their context (i.e., the previous words). The term ‘word’ should interpreted in a very general way because language models may also be used to predict shorter chunks like syllables. The simplest type of language model that is still very commonly used, is the *n*-gram model.

Acoustic models assign likelihood scores to audio signals. While this by itself is a generative task, acoustic models are often trained on chunks of speech from specific classes (like words, syllables or phonemes). When a new chunk of speech is presented, model comparison can be used to see which model assigns the highest likelihood to it and to predict to which class the chunk of speech is most likely to belong. HMMs are the most popular acoustic models in ASR (Gales and Young, 2008). Even if other models, like neural networks, are used to provide acoustic information, HMMs nearly always form the backbone of the final ASR system.

One other problem that needs to be solved by an ASR system for continuous speech, is the *segmentation* of the speech signal. Chunks of speech cannot simply be assigned to phonemes, syllables or words without knowing at what point in time one symbol ends and another symbol starts. Even if train data like the TIMIT dataset (Garofolo et al., 1993) is used, where humans went through the tedious process of annotating the speech signal directly, the test data will still need to be segmented by the system itself. One of the reasons that HMMs are such an important component of ASR systems is that they provide a tractable distribution over all possible segmentations of a sequence. In many systems, a hierarchy is used in which lower level HMMs represent sub-word level units like phones (or more commonly triplets of phones called triphones) while higher level HMMs represent longer units of which there typically more classes like words.

If the scores provided by both the language models and the acoustic models are all properly normalized likelihood values, they can be combined directly to make a prediction that takes both properties of the language and the speech signal into account. In practice, heuristics may be required to combine the different components of the system.

Representing Audio

Acoustic models rarely work with raw audio waveforms directly. Especially HMMs with Gaussian Mixture emission densities tend to work better with data that is represented in the frequency domain and has been decorrelated. Hence, the raw audio signal is commonly first converted into a spectrogram that contains the powers of frequencies taken over small overlapping time windows of the signal. This processing step is commonly followed by a logarithmic transformation. This step is often preceded by a mapping of the frequencies to the so called Mel scale, which is based based on knowledge about the human ear (Stevens et al., 1937). If this is indeed done, and the discrete cosine transform is taken afterwards, one obtains the commonly used Mel-frequency Cepstral Coefficients (MFCCs). While many other audio features for speech processing exist, like Linear Predictive Coding analysis (Tremain, 1982) and Linear Discriminant Analysis on the Mel-frequency features, a discussion of those feature extraction methods is beyond the scope of this dissertation.

In practice, MFCCs are often combined with a couple of other features. Among those are often the log-energy of the signal, the derivatives of the MFCCs (deltas) and the derivatives of those (delta-deltas). The derivatives are added to supply later processing modules with more context information about the neighbouring frames. In many speech related applications, this leads to a dataset of vectors of between roughly 15 and 150 elements (depending on the number of MFCCs chosen) that represent time windows with an order of magnitude of tens of milliseconds. It is important to keep in mind that most of these feature extraction steps have been optimized for systems that work with HMMs and Gaussian Mixtures with diagonal covariance matrices.

5.2 Speech Enhancement

While the performance of state-of-the-art ASR systems has reached very high levels under controlled circumstances, there is still plenty of room for improvement under suboptimal recording conditions. One of the key features of a robust system for automatic speech recognition (ASR) is the ability to handle background noise. Methods for improving noise robustness either try to improve the quality of the features that are presented to the recognition system (ETSI, 2007), or to improve the robustness of the recognition system itself (Gales, 1995). In this dissertation I focus on the first of these two approaches.

To enhance the quality of features that have been derived from a speech signal, one can either use expert knowledge about the characteristics of human speech or resort to data-driven systems. As more data is becoming available, the latter approach seems to become increasingly viable.

The most direct approach to feature enhancement, is to train a system that filters the noise out of the signal directly. Statistical denoising methods can be roughly divided in discriminative and generative approaches. Discriminative methods try to learn a distribution over clean sequences conditioned on noisy sequences. Generative models try to model both the clean data and the noise in order to decompose a noisy signal. Generative methods tend to generalize better because they are more robust when the noise model is unreliable but require good models of clean speech data, which can be more difficult to learn than conditional noise characteristics. Which type of approach will work best also depends on the nature of the noise and how it has been combined with the input signal. If the noise is for example known to be additive, this greatly simplifies the construction of generative models.

A successful data driven approach to speech denoising is the Stereo-based Piecewise Linear Compensation for Environments (SPLICE) method (Deng et al., 2001), which models the joint distribution between clean and noisy utterances. In this approach, the clean signal is modelled as a piecewise linear function of the noisy signal. This function is constructed from models that have been separately trained on different types and levels of noise.

Neural Networks for Speech Enhancement

Another method that is gaining popularity for speech enhancement problems is non-linear regression using artificial neural networks. What makes neural networks interesting for speech processing tasks, is that they can be applied to large datasets and that they can process information relatively fast after they have been trained. When both noisy and clean versions of utterances are available, neural networks can be trained to perform speech denoising directly (Tamura and Waibel, 1988).

Recently, there has been a revival of interest in neural networks for speech processing in general due to the success of the so-called deep learning approach (Bengio, 2009) on a variety of machine learning tasks. Most importantly, deep architectures have been shown to perform very well as acoustic models for automatic speech recognition (Dahl et al., 2012). The idea behind deep learning is to use neural networks with multiple layers of hidden units that learn increasingly complex representations of the input patterns.

A neural network architecture that is particularly interesting for speech processing is the Recurrent Neural Network (RNN). RNNs are able to process information over time by updating a set of state variables that are a function of both their previous state and the current input pattern. One of the first applications of RNNs in speech recognition was phoneme prediction (Bengio, 1991; Robinson et al., 1994). Recently there have also been good results on both phoneme recognition and large vocabulary continuous speech recognition tasks with very large *reservoir computing* networks (Triefenbach et al., 2010, 2014), which are RNNs for which only the output weights are trained using a linear solver.

For speech enhancement, it has recently been shown that a combination of deep learning with RNNs (Maas et al., 2012) can outperform well-known noise reduction methods like the SPLICE algorithm (Deng et al., 2001) and the ETSI2 advanced front-end (ETSI, 2007). It was also shown that RNNs can improve robust speech recognition in a Tandem setup (Vinyals et al., 2012).

5.3 Bidirectional Truncated RNNs

As datasets are becoming increasingly large and parallel processing units like GPUs are getting more commonly available, it becomes increasingly beneficial for computational methods to perform computations in parallel. Unfortunately, RNNs are quite slow and the computations involved cannot be parallelized when the recurrent computations are the bottleneck. Moreover, they are difficult to train because of the so called *vanishing gradients* problem (Bengio et al., 1994). The vanishing gradients problem occurs because the backpropagation of gradients through time includes a large number of multiplications which have a tendency to either drive the gradients towards zero so learning occurs very slowly, or to make them very large so learning can become unstable.

I proposed two variants of a recurrent neural network architecture that inherits some of the properties of RNNs but allows for more parallelization. The models can also use information from future speech frames and seem to be easier to optimize because the number or required backpropagation iterations is smaller. The way the models process information is similar to the type of information processing in deep neural networks. I demonstrated their performance on the Aurora2 robust ASR task when they are trained with an advanced second order optimization method to remove noise from MFCC speech features. Under matched noise conditions, my models outperformed the SPLICE method and the ETSI2 advanced front-end.

5.3.1 Model Architecture

Let $\mathbf{X}_n$ be a matrix that represents a noisy input sequence of column vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, where N is the length of the sequence, and let $\mathbf{Y}$ be the corresponding clean sequence we want to predict. The goal is to learn a function $\mathbf{X} \mapsto \hat{\mathbf{Y}}$ with a minimal difference between the predicted sequence $\hat{\mathbf{Y}}$ and the true clean sequence $\mathbf{Y}$ as measured in the squared error given by

$$\|\hat{\mathbf{Y}} - \mathbf{Y}\|^2, \tag{5.1}$$

where $\| \cdot \|$ is the Frobenius norm.

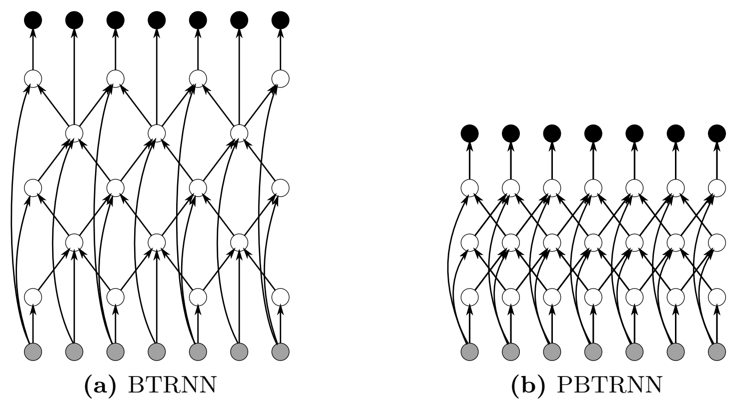


Figure 5.1: The two variants of the bidirectional truncated recurrent neural network for two iterations of hidden unit updates. Each circle represents a layer of neural network units. Grey shading indicates that a unit is part of the input pattern. Black shading signifies output units. Time goes from the left to the right so each input unit represents an MFCC vector at a different time step.

The architecture I proposed, processes information in two directions through time. Unlike a standard recurrent neural network, it also uses information about future frames. I chose to call the architecture 'truncated', because the number of iterations that is carried out to process information is much smaller than the length of the input sequence.

Like a multilayer perceptron, the bidirectional truncated recurrent neural network computes a sequence of hidden unit activations $\mathbf{H}$ and maps these activations to a prediction sequence $\hat{\mathbf{Y}}$. The activation of a hidden unit is computed as a weighted sum of the activations of the units it is connected to, followed by a non-linear transformation (in this case the hyperbolic tangent function).

The temporal component is introduced in the computation of these hidden unit activations. For a predetermined number of iterations K, the hidden units are updated with the information they receive from both their neighbours and the input pattern $\mathbf{X}$ (see Figure 5.1a). At the start of an iteration, the odd hidden state vectors $\{\mathbf{h}_m | m \in 2\mathbb{N} + 1\}$ receive a weighted sum of the activations of their neighbours with an even index and the current input frame. Subsequently, the even units are updated based on the activations of the odd ones. Algorithm 5 shows this procedure in detail. The trainable parameters of the model are the input connection weights $\mathbf{W}_{\text{in}}$, the recurrent weights $\mathbf{W}_{\text{rec}}$, the output weights $\mathbf{W}_{\text{out}}$ and the bias parameters for the recurrent and output layers $\mathbf{b}_{\text{rec}}$ and $\mathbf{b}_{\text{out}}$.

The two inner for-loops in Algorithm 5 can both be replaced by two matrix multiplications. On parallel computing architectures, this modification can lead to significant speedups. This kind of parallelization is not possible for standard recurrent architectures where $N-1$ matrix-vector products are required. The architecture of this first variant of the model is shown in Figure 5.1a. From now on I will refer to this model as the bidirectional truncated recurrent neural network (BTRNN).

Algorithm 5 Predict clean sequence $\hat{\mathbf{Y}}$ given noisy sequence $\mathbf{X}_n$

```
Initialize $\mathbf{H} \leftarrow \mathbf{0}$
Define $\mathbf{H}_{\cdot,0} = \mathbf{H}_{\cdot,N+1} = \mathbf{0}$
$\mathbf{A} \leftarrow \mathbf{W}_{\text{in}}\mathbf{X}_n + \mathbf{b}_{\text{rec}}\mathbf{1}^\mathsf{T}$
for $k = 1$ to $K$ do
  for $j = 1$ to $N$ step 2 do
    $\mathbf{H}_{\cdot,j} \leftarrow \tanh(\mathbf{W}_{\text{rec}}\mathbf{H}_{\cdot,j-1} + \mathbf{W}_{\text{rec}}^\mathsf{T}\mathbf{H}_{\cdot,j+1} + \mathbf{A}_{\cdot,j})$
  end for
  for $j = 2$ to $N$ step 2 do
    $\mathbf{H}_{\cdot,j} \leftarrow \tanh(\mathbf{W}_{\text{rec}}\mathbf{H}_{\cdot,j-1} + \mathbf{W}_{\text{rec}}^\mathsf{T}\mathbf{H}_{\cdot,j+1} + \mathbf{A}_{\cdot,j})$
  end for
end for
$\hat{\mathbf{Y}} \leftarrow \mathbf{W}_{\text{out}}\mathbf{H}_{\cdot,1..N} + \mathbf{b}_{\text{out}}\mathbf{1}^\mathsf{T}$
```

Updating the odd and even units separately was motivated by the mean-field algorithm for Markov Random Fields (Wainwright and Jordan, 2008). If the hidden units would be stochastic variables, this updating scheme would be equivalent to a mean-field approximation of their posterior distribution given the data.

A nice property of this equivalency is that the updates are guaranteed to converge to a fixed point that is a local optimum of an optimization problem that aims to approximate the distribution over the hidden units given the data. To see if this alternating updating is actually necessary, I also looked at a simpler version of the model in which all hidden units are updated in parallel. Updating the hidden units in parallel is computationally more efficient but theoretically this could lead to oscillations. However, since the number of iterations I intended to use was quite small, I didn't expect this to be a problem. The architecture of this second variant of the model is shown in Figure 5.1b. I will refer to this model as the parallel bidirectional truncated recurrent neural network (PBTRNN).

Note that the way the models process information from both previous and future frames is not the same as was done in previous work where the term bidirectional RNN was used (Schuster and Paliwal, 1997; Graves and Schmidhuber, 2005). In these approaches, two different hidden states captured past and future information separately while the models I proposed directly integrate future and past information into a single state representation.

The number of previous and future frames that are taken into account is determined by the number of iterations K. The prediction for a frame receives information about $2K + 1$ frames from the input sequence. The number of iterations to use presents a trade-off between context size and computational efficiency. It is also clear that as the models run for a greater number of iterations, the number of times that information flows through a non-linear activation unit increases. At every iteration, the information can be said to pass through an additional layer of a deep neural network with tied weights and a sparse connection structure. This is similar to the way convolutional neural networks process information.

5.4 Experiments

To evaluate the RNN models for speech denoising, I used the Aurora2 dataset (Pearce et al., 2000). This is a well-known benchmark for robust ASR. The dataset is a collection of connected digits from the TIDigits corpus. The train set contains $8,440$ clean sentences of which duplicates exist that have been corrupted by four noise types at seven different levels of signal to noise ratio (clean, 20dB, 15dB, 10dB, 5dB, 0dB, -5dB). The test set consists of $56,056$ corrupted and clean sentences and is divided in three subsets. Test set A contains the same four noise types as the train data. Test sets B and C contain different types or noise and can be used to evaluate the performance of a model under mismatched training and testing conditions. All systems were trained to map the noisy train sentences to their clean counterparts. Since the train data only contains a small number of noise types, I did not expect models that were trained on this data to perform very well on new noise types in general. For this reason I will focus on the results on test set A for which the noise types are identical to those in the train set.

A recognition system was trained on the clean train data only. Subsequently, the noisy utterances from the test set were processed by the various denoising models I compared and these denoised features were presented to the recognizer during testing. To generate MFCC features, I used the standard HTK scripts that were supplied with the Aurora2 dataset (Pearce et al., 2000). The models were trained on 13 dimensional vectors that contained 12 MFCC features and the log energy of the speech signal. First and second order derivatives (delta and delta-delta features) were appended after the denoising process during testing. To keep my results comparable with those in Pearce et al. (2000) I did not apply cepstral mean normalization to the signals. The recognizer was also trained and evaluated with the standard Aurora2 scripts for HTK. This means that whole-word HMMs with 16 states and for each state a Gaussian mixture with 3 components were used.

5.4.1 Model Settings

The BTRNN and PBTRNN both had 500 hidden units. I set the number of iterations at which the hidden units were updated to 6. This gave each model an input context size of 13 frames and $263,513$ trainable parameters.

I compared the proposed models with a deep recurrent denoising auto-encoder (DRDAE), a model which has been shown to work well for this task (Maas et al., 2012). This model is a neural network with three layers of hidden units of which the second layer of hidden units has recurrent connections. Like in the paper by Maas et al. (Maas et al., 2012), I set the number of units in each hidden layer to 500 and presented the network with small time windows of three frames: the current time frame and its two neighbours. The number of trainable parameters for this model was $777,513$.

To investigate the importance of the recurrent data processing in the networks I proposed, I also trained a standard multi-layer perceptron with roughly the same number of trainable parameters. This network had a single layer of 1450 hidden units. To make the comparison as fair as possible, I presented this network with time-windows of 13 frames so that it received the same amount of context information as the truncated recurrent neural networks at each time step. The number of trainable parameters of this model was $265,363$.

Finally, I compared the models with the features generated by the ETSI2 advanced front-end (AFE) and the SPLICE algorithm. (ETSI, 2007). The WER scores of these methods were not replicated but taken from the results reported by the authors who also used the standard AURORA scripts.

5.4.2 Training

The neural network models were trained to minimize the mean squared error (MSE) between their predictions and the actual clean utterances in the train set. All utterances were normalized by subtracting the mean and dividing by the square root of the variances of the noisy utterances from the train set.

To train the models I used Hessian-Free optimization (Martens, 2010). This is a second order optimization method which has been shown to be ef-

Table 5.1: Mean squared error for the neural network models on the train and validation data. The scores are averaged over all different levels and types of noise.

Model	Train	Validation
DRDAE	17.30	17.55
MLP	16.35	17.62
BTRNN	15.38	16.39
PBTRNN	15.18	16.89

fective for training deep neural networks and RNNs (Martens and Sutskever, 2011). Earlier results about Hessian-Free optimization also suggest that it is not that important anymore to use any form of unsupervised pre-training when this optimization method is used (Martens, 2010). Because there doesn't seem to be a straightforward way to apply pre-training to the models, this property of the optimization algorithm seems to be particularly appealing.

The Hessian-Free optimizer is a truncated Newton method and needs the gradient of the error objective with respect to the parameters of a model. It also needs a function that provides it with the product of the Gauss-Newton matrix of the model with an arbitrary vector. If the objective is twice differentiable these gradients and matrix vector products can easily be obtained by means of automated differentiation. I used the Theano toolbox (Bergstra et al., 2010) for python for this and to simulate the more parallelizable models with a GPU. I used the standard settings for the Hessian-Free algorithm as described in the original paper by James Martens (Martens, 2010). In preliminary experiments, I found that this algorithm outperformed LBFGS and stochastic gradient descent.

The dampening parameters λ was initialized at 100. The number of sequences to use for computing the matrix vector products was set to 300. All weight matrices where initialized with values sampled from a zero mean Gaussian with variance .01. All bias parameters were initialized at 0.

I reserved 20% of the train data for validation. I applied early stopping and selected the parameters that achieved the lowest MSE scores on the validation data after 100 iterations of training for denoising the MFCCs of the test set. I did not apply any form of weight decay during training.

5.4.3 Results

Table 5.1 displays the MSE scores for the models on both the sentences from the train set and those that I left for validation. The BTRNN and PBTRNN have the lowest validation scores. The train error of the PBTRNN is slightly lower than that of the BTRNN but it performs slightly worse on the validation set. Interestingly, the MLP suffers more from early overfitting than the DRDAE model as its train error is quite a bit lower but its validation error is slightly higher. Table 5.2 shows the word error rates (WER) scores for test set A. This test set had the same types of noise as the train set. Except for the clean utterances, where the AFE got the best performance, the BTRNN and the PBTRNN outperformed all other methods for all noise conditions. The PBTRNN performed slightly better under moderate noise conditions while the BTRNN a performed a little bit better under heavier noise conditions.

It is surprising how badly the MLP performed on the clean data. This indicates that either recurrent or deep processing, or a combination of them, is important for good neural denoising models. The performance of the DRDAE was worse than the BTRNN and the PBTRNN but the rate at which its performance decreases as a function of increasing levels of noise seems similar. I suspected that it should be able to achieve similar performance to my models but is more difficult to optimize.

I also averaged across noise conditions while leaving out the clean and -5dB scores as described in the ETSI standard. The BTRNN now achieves a WER of 9.59% and the PBTRNN a WER of 10.03%. These results are better than the WER rates reported for the AFE (ETSI, 2007) and the SPLICE algorithm (as reported by Droppo et al. (Droppo et al., 2002)) which are 12.26% and 11.67%, respectively. The DRDAE gives a WER of 11.57%, which is higher than the result reported by Maas et al. (Maas et al., 2012) of 10.85%, which is still higher than the WER scores of both of the proposed models. Finally, the MLP gives an average WER of 13.09%, performing worse than the AFE baseline.

While it was not my intention to construct models for mismatched noise conditions, I also report the averaged results for test set B. As expected and like in earlier work (Maas et al., 2012), the results of the neural models are a lot worse for the mismatched noise types. The BTRNN and PBTRNN achieved WER scores of 25.50% and 24.66%, respectively. This is worse than the SPLICE algorithm which gives a WER of 12.25% on this test set. The MLP and DRDAE gave WER scores of 19.84% and 18.29%, respectively. All models still improved on the raw MFCC features which lead to a WER of 44.42%. It seems that the models that performed worse on the matching test set suffer less from the mismatching conditions but are still worse than the

Table 5.2: Word error rate scores for the neural models and the ETSI2 advanced front-end on test set A. The results are averaged over the four different noise types.

SNR	MFCC	AFE	DRDAE	MLP	BTRNN	PBTRNN
Clean	1.06	**0.77**	1.36	4.14	1.37	1.41
20dB	5.02	1.70	1.85	3.04	1.82	**1.68**
15dB	13.11	3.08	2.93	3.77	2.36	**2.19**
10dB	32.81	6.45	5.14	6.08	**3.91**	4.03
5dB	60.74	14.16	12.49	14.00	**9.87**	10.31
0dB	82.98	35.92	35.45	38.59	**29.98**	31.97
-5dB	91.62	68.70	73.70	72.79	**65.37**	67.42

SPLICE baseline. This indicates that the BTRNN and PBTRNN learned to represent the noise in the train data very well but did not have enough noise data available to generalize well to other noise types. Note that the ability of the SPLICE algorithm to perform well on mismatched noisy data may be caused by the fact that separate models are used for each type of noise and noise condition. During testing, a special routine was used to select the most probable noise environment type. The fact that the splice algorithm knew which noise was from which type and condition during training can be seen as an additional annotation that may have helped for generalization.

5.4.4 Computational Efficiency

It is difficult to perform a fair assessment of the computational efficiency of the models because this is implementation dependent. Nonetheless, I measured the time it took for the models to process the first 1000 utterances from the train set. The simulations were done on a system with an Intel i7 quadcore CPU and a Geforce GTX 480 GPU card installed. For all models, I report the computation time for both the GPU and the CPU. Computations on the CPU made use of the Intel MKL BLAS library with multi-threading enabled. Since all methods were implemented in the Theano toolbox, their efficiency depends on the optimizations this software is able to perform for the various computational graphs. These implementations could probably be made more efficient by optimizing them manually.

As Table 5.3 shows, the BTRNN and PBTRNN benefit a lot more from the use of the GPU than the DRDAE model. A direct comparison is not fair

Table 5.3: Time in seconds it took for each model to process the first 1000 utterances of the train set using either the CPU or a GPU to perform computations.

Model	CPU	GPU
DRDAE	15.4	20.6
MLP	7.7	1.2
BTRNN	56.0	10.2
PBTRNN	25.1	5.1

because of the different numbers of parameters but these results still suggest that the truncated RNNs may scale better when larger numbers of units are used.

5.5 Discussion

I introduced two variants of a bidirectional truncated recurrent neural network architecture for speech denoising that shares properties with deep learning and convolutional neural networks. The two models outperform similar neural architectures and the ETSI2 advanced front-end for noise types that were present in the training data. Since the deep recurrent denoising auto-encoder I compared my models with can use more context information, I suspect that my models performed better because they are easier to optimize.

In contrast to more common recurrent architectures that only have access to information about the past, the proposed models also use information about future frames. The models could still be applied in a setup where the speech frames arrive in an online fashion but in that case it would probably be best to re-update the hidden units after more information has become available.

The fact that the models were overfitting on the specific noise types suggests that their generalization properties could be improved by extending the training data with more types of noise at the potential cost of performing slightly worse on test set A. It should be straightforward to construct such a dataset. I also showed that the models I proposed are more computationally efficient than similar architectures. This increases their potential to scale to larger datasets.

6 Conclusion and Future Work

In this final chapter I will summarize the most important research conclusions and discuss some of the limitations and ideas for future research. First I will discuss my personal view on what the most interesting directions for future research are in general, followed by more specific research ideas that are directly related to my own work.

6.1 Summary

As data sets are becoming larger and the problems we want to solve in statistical modelling are getting more complicated, it may also be necessary to use more complicated statistical models. Unfortunately, many of the more complicated types of models cannot be normalized analytically. In this dissertation I investigated ways to improve the practical applicability of unnormalized models.

After giving a brief overview of some of the most popular contemporary stochastic approximate maximum likelihood methods for training unnormalized models, I investigated ways for making the underlying sampling methods more efficient. Taking ideas from the statistical physics literature, I was able to show that the mixing time of Markov Chain Monte Carlo methods used for training Restricted Boltzmann Machines (RBMs) can be improved by extending a method called Parallel Tempering. Parallel Tempering employs a collection of interacting Markov chains that operate at different temperatures to improve the efficiency at which the Markov chain with the lowest temperature visits separated regions of the sample space. This strategy is wasteful with computational resources in that only a single chain is providing useful samples while multiple chains have to be simulated and kept track of in memory. I proposed to use a larger set of possible interactions between the chains and to use a weighted average of the different chains during training. The first method did improve the mixing behavior and the combination of the two seemed to lead to better performance when integrated in a training algorithm. Unfortunately, the methods also introduced additional computational costs and it is still difficult to say how much practical value they have compared to more conventional training methods.

Subsequently, I looked at methods that serve as alternatives to maximum likelihood estimation. Examples of these are quasi-likelihood methods, score matching and noise contrastive estimation. The idea behind such alternative estimators is that they may still posses some of the most desirable properties of maximum likelihood estimation while potentially circumventing some of its computational difficulties. I showed how ideas from previous work about Bregman divergences for training unnormalized models can be used to construct a new estimator that combines some of the properties of score matching and noise contrastive estimation. The estimator is somewhat comparable to a version of noise contrastive estimation in which a non-parametric noise distribution is used to be more similar to the data distribution than generic parametric models tend to be. I demonstrated that the estimator appears to be consistent when trained on synthetic problems. When used to train Products of Student-t distributions on natural image data, the estimator seemed to find features that look very similar to those found by other estimation methods. This indicates that the estimator provides a viable alternative when the goal is to estimate models for which other methods turn out to be impractical.

In the second part of the dissertation, I looked at methods that abandon the generative aspect of the training procedure entirely. When the goal is to solve some specific prediction task (e.g., regression) and a approximate inference method is used, it may be a better idea to train the model such that the inference method provides the desired predictions rather than training the model to model the data generatively. I applied previous ideas in this direction to the specific case of missing value imputation for time-series where inference was done using either gradient-based method or naive mean field. I was able to show that these methods allowed relatively complex unnormalized energy models to be trained to do missing value imputation and outperform several other methods on this task. Even though generative modelling was not the purpose of these models, I found they could still be used to generate interesting looking samples when used as so-called Generative Stochastic Networks (Bengio and Thibodeau-Laufer, 2013).

Finally, I took the architecture that seemed most practical for the imputation tasks and applied it to a speech processing task. Instead of missing value imputation, the model was now trained for supervised regression but the computational structure of the model was still inspired by the computations involved in naive mean field inference. This led to an architecture that can be seen as a bidirectional recurrent neural network. The architecture turned out to be more computationally efficient on hardware specialized for parallel processing. Two versions of the architecture were compared on a speech denoising task and outperformed both standard and other neural network based methods. However, the models did not generalize very well to noise types that were very different from those used during training.

6.2 Future Perspectives

The estimation of unnormalized statistical models is a quite general problem and one with a lot of potential for future research. In this dissertation, most of the more conceptual research was done using small synthetic problems and image modelling tasks. The more application based research mainly involved time-series problems because they seem to require architectures that quickly become too complex to remain tractable. Generally speaking, most of the ideas that have been tested on small problems should be tested on larger more realistic problems. For the models that were used for time-series problems, the emphasis of future research in this area should be more on the potential improvements that can be obtained by improving the architectures and the specific data processing pipelines.

6.2.1 Multi-Tempering

The Multi-Tempering method, may prove to be more interesting for very large and complex problems than for the problems on which it has been tested so far. After all, the number of models and the roughness of the energy landscape are important causes of potential mixing problems. For relatively simple problems, the mixing problem may not be severe enough to reward computationally demanding tempering methods.

In the experiments about multi-tempering for RBMs one could see that there was often a very distinct location in the range of temperatures where transitions were very rare. This suggests that adapting the temperature values can greatly improve the return time of any PT-based algorithm. It has indeed been shown that RBM training can be improved using a temperature adaptation algorithm that aims to optimize return times directly (Desjardins et al., 2010a). It would be interesting to see how the application of such an adaptation method influences the comparisons between different tempering algorithms.

A general problem for MCMC methods for maximum likelihood training is that the number of modes may be so high, that even a relatively fast mixing chain will never be able to visit all of them (Bengio, 2013). If this turns out to be the case, it may be difficult to verify because most methods for evaluating intractable energy-based models would suffer from the same problem. However, if this means that only the modes that are easy enough to reach by approximate inference algorithms are adapted to the data, the approximations used for evaluating these models may still have practical usefulness. An interesting experiment would be to train an RBM on a real-world data set, run an efficient MCMC sampler for a very long time and investigate the robustness of the estimated expectations over time.

Some recent work (Bengio et al., 2013) showed that MCMC sampling mixes more efficiently for deep models if it takes place at higher level variables. In other words, the idea is that models with multiple layers of representations learn to map the input space to a space in which the data is probably distributed more uniformly. Conceptually this is comparable with the way in which PCA whitening transfers the data to a space in which it has a simpler distribution. Perhaps it is more fruitful to investigate how models and perhaps their parameters can be altered in such a way that they either become easier to sample from in general or remain easier to sample from during later stages of training.

There is clearly plenty of work to be done to improve sampling methods for training unnormalized models like RBMs but it will be interesting to see how prominent MCMC methods will be in the future when new types of hardware are available. This applies especially to recent work on hardware that allows for efficient parallel computation based on quantum annealing (Johnson et al., 2011). Future research should investigate both how quantum annealing based algorithms can be used to train interesting models and focus on parallel methods for approximating expectations in general.

Finally, it would be very useful to have better measures of the quality of sampling algorithms. This would obviously make comparisons for research purposes more reliable and conclusive. Furthermore, it could be very useful to have ways to predict during which stages of training the mixing rate starts to become problematic so that one can switch to a slower training algorithm with better mixing properties.

6.2.2 Alternative Estimators

The proposed data dependent contrastive estimator its most promising asset seems to be that it can be applied to models for which Gibbs sampling is not possible and for which it is not practical to apply methods like score matching either. For this reason it would be interesting to see how this method compares with noise contrastive estimation for models that are very large and trained on relatively large data sets. I foresee that both methods would have numerically stability issues during training. This provides yet another subject for possible improvement and research.

As mentioned before, the qualitative validation of the estimator described in this thesis should be regarded as a preliminary sanity check. There is still a lot of work to be done to learn more about its practical value. A first step would be to investigate its ability to learn features that prove to be useful for classification. Evaluating estimators of this type using measures like AIS may prove to be problematic because a measure of the likelihood score may not be optimal for estimators that may behave quite differently for finite amounts of data and training time. A more thorough comparison with vanilla NCE is also needed.

Since a Parzen density estimator with a Gaussian kernel served as the inspiration for the estimator, only continuous data has been considered. An estimator for discrete distributions would require a Parzen density estimator with a suitable smoothing kernel.

Finally, there is also more theoretical work to be done. A theoretical proof of the consistency of the estimator would provide additional reason to investigate it empirically as well. Other properties, that have not been investigated yet, are the efficiency of the estimator and the distribution of its variance. Since it has been formally shown that NCE is both consistent and normally distributed (Gutmann and Hyvärinen, 2012), there is some reason to expect that the proposed estimator can be formally shown to posses these properties as well.

6.2.3 Backpropagation-Through-Inference

The backpropagation-through-inference based training ideas that were used for the imputation already seem to work quite well on some practical problems. Of course the size of these problems could still be increased to provide more substantial evidence for the practical usefulness of the methods but it would be more interesting to look at more conceptual properties. One example of this would be to investigate the relative performance of different inference algorithms and their sensitivity to the number of iterations for which they are ran. As Justin Domke also pointed out on his weblog[1], methods of this kind are turning the inference algorithm in some sort of black box and it would be interesting to see to what extent the inference iterations

[1]At this moment online at `http://justindomke.wordpress.com` and certainly worth a visit.

need to be identical to those of the original inference algorithm or whether they still work if they are only loosely based on them or even are applied to different sets of parameters at each iteration.

Future work should investigate the performance of different inference methods. Interesting candidates would be variants of loopy belief propagation, expectation propagation and structured mean-field. It would also be interesting to see if models of this type can be designed for other task domains like image inpainting.

The use of energy-based imputation models as Generative Stochastic Networks is an interesting venue for future research as well. First it should be investigated if the mixing behavior of the sampling process can be improved. Possible options for this would be to include variance parameters and train with a larger average number of missing values. The walk-back algorithm might help as well (Bengio and Thibodeau-Laufer, 2013).

Finally, the relation of the imputation learning framework with composite likelihood learning suggests that in the limit of infinitely many training examples, a misspecification of the noise process may become less influential. This is an assumption that requires both more theoretical foundation and empirical validation.

6.2.4 Speech Denoising

The models I proposed for speech denoising worked quite well for the task they were trained on. The over fitting on the specific noise types used during training seems more inherent to the task than a problem with the models themselves. Future work should investigate the applicability of these models on larger datasets for which the training data has been augmented with artificially generated noise as has been done in other work on robust speech recognition (Gemmeke and Virtanen, 2010). It would also be interesting to combine this approach with other techniques for noise robust ASR. It was beyond the scope of this work to investigate the influence of the number of iterations for which the hidden units are updated. The value we chose was based on prior intuitions about the task and the influence of this parameter on the performance of the models should be investigated in more detail.

6.3 Final Notes

This thesis presented a variety of approaches for training unnormalized statistical models. Now it would of course be interesting to know which of the presented ideas is the most useful one to use for practical applications. Or, if that question cannot be answered, what kind of properties of a modelling problem determine which approach should be preferred in that situation. Should one always prefer to turn generative models into deterministic black-box methods because this eliminates the need to approximate partition functions? Should one use sampling methods or alternative estimators when a generative model is required but inference is intractable? These are difficult questions to answer because the number of types of possible modelling problems is virtually infinite but I will try to present what I personally think are some of the more important considerations to keep in mind when choosing how to train a model and what kind of model to use.

First, I think it is important to get a very precise idea about the kind of task the model is supposed to be used for. If it is necessary to obtain a generative model, one may either try to approximate maximum likelihood learning or to use suitable alternative estimator. If the goal is to make direct predictions, it may be a better idea to abandon the whole idea of generative training altogether and use a more direct training approach. The bottom line is that it is important to keep alternatives to generative modelling in mind instead of choosing it out of habit.

Given that a generative model *is* desired, there are still many options to choose from. Approximate methods for maximum likelihood estimation may be most suitable when it is relatively easy to obtain samples from the model. When MCMC methods need to be used, this means that one needs to consider how efficient this method is going to be for the specific model. If it is computationally demanding to do a single sampling step, or if one has reasons to believe that the model distribution contains many modes that are separated by regions of very low probability density, it may be a better idea to look at alternative estimators instead. Again, it is important to keep in mind what the final application of the model will be. Is it important that the model can answer precise questions about a couple of its modes, or is it important that samples from the model are very similar to the true data? Given that the data to train on is finite, different estimators can provide models with very different strengths and weaknesses. Maximum likelihood may be preferred when one is looking for a model that provides high quality samples while certain alternative estimators may lead to models that are better for denoising because they put more effort into optimizing the shape of the distribution around the data while caring less about distant modes.

Finally, when models are trained to be components of a bigger system, as is done in deep learning approaches, the number of considerations to keep in mind increases even further. Component models that are very good at providing representations that have to be interpreted by other models are not necessarily those with the best likelihood scores. Different training methods may lead to models that produce very different types of representations in this setting. To find out what kind of approaches lead to good representations for deep learning models is not only an interesting venue for future research, but illustrates why I don't think any of the research directions I've discussed should be abandoned in favor of one that appears to be more practical at first sight.

Bibliography

Athènes, M. and Calvo, F. (2008). Multiple-Replica Exchange with Information Retrieval. *Chemphyschem*, 9(16):2332–9.

Barbu, A. (2009). Training an active random field for real-time image denoising. *Trans. Img. Proc.*, 18(11):2451–2462.

Bengio, Y. (1991). *Artificial Neural Networks and Their Application to Sequence Recognition*. PhD thesis, Montreal, Que., Canada, Canada. UMI Order No. GAXNN-72116 (Canadian dissertation).

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127.

Bengio, Y. (2013). Deep learning of representations: Looking forward. In *Statistical Language and Speech Processing*, pages 1–37. Springer.

Bengio, Y. and Gingras, F. (1996). Recurrent neural networks for missing or asynchronous data. *Advances in Neural Information Processing Systems*, pages 395–401.

Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2013). Better mixing via deep representations. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 552–560.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

Bengio, Y. and Thibodeau-Laufer, É. (2013). Deep generative stochastic networks trainable by backprop. *arXiv preprint arXiv:1306.1091*.

Bengio, Y. and Yao, L. (2013). Bounding the test log-likelihood of generative models. *CoRR*, abs/1311.6184.

Berg, B. A. and Billoire, A. (2004). *Markov Chain Monte Carlo Simulations.* Wiley Online Library.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: A cpu and gpu math compiler in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 3 – 10.

Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C. (2000). Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 417–424, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Bertsekas, D. P. (1999). *Nonlinear Programming.* Athena Scientific, Belmont, MA.

Besag, J. (1975). Statistical analysis of non-lattice data. *The statistician*, pages 179–195.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML.* icml.cc / Omnipress.

Bregman, L. M. (1967). The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 7(3):200–217.

Brenner, P., Sweet, C. R., VonHandorf, D., and Izaguirre, J. A. (2007). Accelerating the Replica Exchange Method through an Efficient All-Pairs Exchange. *The Journal of Chemical Physics*, 126(7):074103.

Breuleux, O., Bengio, Y., and Vincent, P. (2011). Quickly generating representative samples from an rbm-derived process. *Neural Computation*, 23(8):2058–2073.

Cho, K., Ilin, A., and Raiko, T. (2011a). Improved learning of gaussian-bernoulli restricted boltzmann machines. In *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 10–17. Springer.

Cho, K., Raiko, T., and Ihler, A. T. (2011b). Enhanced gradient and adaptive learning rate for training restricted boltzmann machines. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 105–112.

Cho, K., Raiko, T., and Ilin, A. (2010). Parallel tempering is efficient for learning restricted boltzmann machines. In *IJCNN*, pages 1–8. Citeseer.

Cho, Y. and Saul, L. K. (2009). Kernel methods for deep learning. In *NIPS*, volume 9, pages 342–350.

Courville, A. C., Bergstra, J., and Bengio, Y. (2011). A spike and slab restricted boltzmann machine. In Gordon, G. J., Dunson, D. B., and Dudík, M., editors, *AISTATS*, volume 15 of *JMLR Proceedings*, pages 233–241. JMLR.org.

Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42.

Deng, L., Acero, A., Droppo, J. L., and Huang, J. X. (2001). High-performance robust speech recognition using stereo training data. In *Proc. ICASSP*, pages 301–304, Salt Lake City, UT.

Deselaers, T., Hasan, S., Bender, O., and Ney, H. (2009). A deep learning approach to machine transliteration. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 233–241. Association for Computational Linguistics.

Desjardins, G. and Bengio, Y. (2008). Empirical evaluation of convolutional RBMs for vision. Technical Report 1327, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

Desjardins, G., Courville, A., and Bengio, Y. (2010a). Adaptive parallel tempering for stochastic maximum likelihood learning of rbms. In *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning.*

Desjardins, G., Courville, A. C., Bengio, Y., Vincent, P., and Delalleau, O. (2010b). Tempered markov chain monte carlo for training of restricted boltzmann machines. *Journal of Machine Learning Research - Proceedings Track*, 9:145–152.

Domke, J. (2011). Parameter learning with truncated message-passing. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 2937–2943, Washington, DC, USA. IEEE Computer Society.

Domke, J. (2012). Generic methods for optimization-based modeling. *Journal of Machine Learning Research - Proceedings Track*, 22:318–326.

Droppo, J., Deng, L., and Acero, A. (2002). Evaluation of splice on the aurora 2 and 3 tasks. In *International Conference on Spoken Language Processing*, pages 29–32.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid monte carlo. *Physics letters B*, 195(2):216–222.

ETSI (2007). Advanced front-end feature extraction algorihtm. Technical Report ETSI ES 202 050.

Fischer, A. and Igel, C. (2010). Empirical analysis of the divergence of gibbs sampling based learning algorithms for restricted boltzmann machines. In *Artificial Neural Networks–ICANN 2010*, pages 208–217. Springer.

Frank, A. and Asuncion, A. (2010). UCI machine learning repository.

Freire, A., Barreto, G., Veloso, M., and Varela, A. (2009). Short-term memory mechanisms in neural network learning of robot navigation tasks: A case study. In *Robotics Symposium (LARS), 2009 6th Latin American*, pages 1–6.

Freund, Y. and Haussler, D. (1994). Unsupervised Learning of Distributions on Binary Vectors Using Two Layer Networks. Technical report, Santa Cruz, CA, USA.

Gales, M. and Young, S. (2008). The application of hidden markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304.

Gales, M. J. F. (1995). *Model-Based Techniques, for Noise Robust Speech Recognition*. PhD thesis, University of Cambridge.

Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S., and Dahlgren, N. L. (1993). DARPA TIMIT acoustic phonetic continuous speech corpus CDROM.

Gemmeke, J. F. and Virtanen, T. (2010). Artificial and online acquired noise dictionaries for noise robust ASR. In *INTERSPEECH*, pages 2082–2085.

Ghahramani, Z. and Jordan, M. I. (1997). Factorial hidden markov models. *Machine Learning*, 29(2-3):245–273.

Ghahramani, Z. and Roweis, S. T. (1999). Learning nonlinear dynamical systems using an em algorithm. *Advances in neural information processing systems*, pages 431–437.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Goodfellow, I. J., Mirza, M., Courville, A., and Bengio, Y. (2013). Multi-prediction deep Boltzmann machines. In Burges, C. J. C., Bottou, L., Ghahramani, Z., and Weinberger, K. Q., editors, *NIPS*.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE.

Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610.

Grünwald, P. D. and Dawid, A. P. (2004). Game theory, maximum entropy, minimum discrepancy and robust bayesian decision theory. *Annals of Statistics*, pages 1367–1433.

Gupta, A. and Lam, M. (1996). Estimating missing values using neural networks. J. Operat. Res. Soc. 47, 229-238 (1996).

Gutmann, M. and Hirayama, J. (2011). Bregman divergence as general framework to estimate unnormalized statistical models. In *Proc. Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 283–290, Corvallis, Oregon. AUAI Press.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*, pages 297–304.

Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13:307–361.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109.

Hateren, J. H. v. and Schaaf, A. v. d. (1998). Independent component filters of natural images compared with simple cells in primary visual cortex. *Proceedings: Biological Sciences*, 265(1394):359–366.

Hermans, M. and Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 190–198.

Hinton, G. E. (2002). Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800.

Hinton, G. E., Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Hinton, G. E. and Sejnowski, T. J. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning and Relearning in Boltzmann Machines, pages 282–317. MIT Press, Cambridge, MA, USA.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580.*

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hyvärinen, A. (2005). Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709.

Hyvärinen, A. (2007a). Connections between score matching, contrastive divergence, and pseudolikelihood for continuous-valued variables. *Neural Networks, IEEE Transactions on*, 18(5):1529–1531.

Hyvärinen, A. (2007b). Some extensions of score matching. *Computational statistics & data analysis*, 51(5):2499–2512.

Iba, Y. (2001). Extended ensemble monte carlo. *International Journal of Modern Physics C*, 12(05):623–656.

Jaeger, H. (2012). Long short-term memory in echo state networks: details of a simulation study. Technical Report 27.

Jain, V., Murray, J., Roth, F., Turaga, S., Zhigulin, V., Briggman, K., Helmstaedter, M., Denk, W., and Seung, H. (2007). Supervised learning of image restoration with convolutional networks. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

Johnson, M., Amin, M., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A., Johansson, J., Bunyk, P., et al. (2011). Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4. IEEE.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques.* MIT Press.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4.

Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, pages 536–543. ACM.

Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. *JMLR: W&CP*, 15:29–37.

Lawrence, N. D. (2004). Gaussian process latent variable models for visualisation of high dimensional data. *Advances in neural information processing systems*, 16(329-336):3.

Lawrence, N. D. (2007). Learning for larger datasets with the gaussian process latent variable model. In *Proceedings of the Eleventh International Workshop on Artificial Intelligence and Statistics*, pages 21–24. Omnipress.

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. In Bakir, G., Hofman, T., Schölkopf, B., Smola, A., and Taskar, B., editors, *Predicting Structured Data.* MIT Press.

LeCun, Y. and Huang, F. (2005). Loss functions for discriminative training of energy-based models. In *Proc. of the 10-th International Workshop on Artificial Intelligence and Statistics (AIStats'05).*

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference* *on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA. ACM.

Lindsay, B. G. (1988). Composite likelihood methods. *Contemporary Mathematics*, 80:221–239.

Maas, A. L., Le, Q. V., O'Neil, T. M., Vinyals, O., Nguyen, P., and Ng, A. Y. (2012). Recurrent neural networks for noise reduction in robust asr. In *INTERSPEECH*. ISCA.

Martens, J. (2010). Deep learning via hessian-free optimization. In Fürnkranz, J. and Joachims, T., editors, *ICML*, pages 735–742. Omnipress.

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *Proc. 28th Int. Conf. on Machine Learning*.

Minka, T. (2001). Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 362–369, San Francisco, CA. Morgan Kaufmann.

Mirowski, P. and LeCun, Y. (2009). Dynamic factor graphs for time series modeling. In *Proc. European Conference on Machine Learning (ECML'09)*.

Mnih, A. and Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273.

Nair, V. and Hinton, G. E. (2009). 3d object recognition with deep belief nets. In *NIPS*, pages 1339–1347.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814.

Neal, R. (2011). Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162.

Neal, R. M. (1998a). Annealed importance sampling. *Statistics and Computing*, 11:125–139.

Neal, R. M. (1998b). Suppressing random walks in markov chain monte carlo using ordered overrelaxation. In *Learning in graphical models*, pages 205–228. Springer.

Nelwamondo, F. V., Mohamed, S., and Marwala, T. (2007). Missing data: A comparison of neural network and expectation maximisation techniques. *arXiv preprint arXiv:0704.3474*.

Ngiam, J., Chen, Z., Koh, P. W., and Ng, A. (2011). Learning deep energy models. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning*, pages 1105–1112, New York, NY, USA. ACM.

Nocedal, J. and Wright, S. J. (2000). *Numerical Optimization*. Springer.

Parzen, E. et al. (1962). On estimation of a probability density function and mode. *Annals of mathematical statistics*, 33(3):1065–1076.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013a). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*. ACM.

Pascanu, R., Montufar, G., and Bengio, Y. (2013b). On the number of inference regions of deep feed forward networks with piece-wise linear activations. *CoRR*, abs/1312.6098.

Pearce, D., gÃŒnter Hirsch, H., and Gmbh, E. E. D. (2000). The aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions. In *in ISCA ITRW ASR2000*, pages 29–32.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural Computation*, 6:147–160.

Pelikan, M., Goldberg, D. E., and Cantu-Paz, E. (1999). Boa: The bayesian optimization algorithm. pages 525–532. Morgan Kaufmann.

Pihlaja, M., Gutmann, M. U., Hyvärinen, A. J., et al. (2010). A family of computationally efficient and simple estimators for unnormalized statistical models. In *Conference on Uncertainty in Artificial Intelligence*.

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17.

Ranzato, M. and Hinton, G. E. (2010). Modeling pixel means and covariances using factorized third-order boltzmann machines. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2551–2558. IEEE.

Robinson, T., Hochberg, M., and Renals, S. (1994). Ipa: Improved phone modelling with recurrent neural networks. In *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, volume 1, pages I–37. IEEE.

Roth, S. and Black, M. J. (2005). Fields of experts: A framework for learning image priors. In *In CVPR*, pages 860–867.

Salakhutdinov, R. (2008). Learning and Evaluating Boltzmann Machines. Technical report UTML tr 2008-002, Department of Computer Science, University of Toronto.

Salakhutdinov, R. (2009). Learning in markov random fields using tempered transitions. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A., editors, *NIPS*, pages 1598–1606. Curran Associates, Inc.

Salakhutdinov, R. and Hinton, G. (2009a). Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978.

Salakhutdinov, R. and Hinton, G. E. (2009b). Deep Boltzmann Machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics.*

Salakhutdinov, R. and Larochelle, H. (2010). Efficient Learning of Deep Boltzmann Machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics.*

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM.

Salakhutdinov, R. and Murray, I. (2008). On the quantitative analysis of Deep Belief Networks. In McCallum, A. and Roweis, S., editors, *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 872–879. Omnipress.

Schulz, H., Müller, A., and Behnke, S. (2010). Investigating convergence of restricted boltzmann machine learning. In *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning.*

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681.

Stevens, S., Volkmann, J., and Newman, E. (1937). A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190.

Stoyanov, V., Ropson, A., and Eisner, J. (2011). Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. *Journal of Machine Learning Research - Proceedings Track*, 15:725–733.

Sutskever, I., Hinton, G. E., and Taylor, G. W. (2008). The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems*, volume 21, Cambridge, MA. MIT Press.

Sutskever, I. and Tieleman, T. (2010). On the convergence properties of contrastive divergence. In *International Conference on Artificial Intelligence and Statistics*, pages 789–795.

Swendsen, R. H. and Wang, J. S. (1986). Replica Monte Carlo Simulation of Spin-Glasses. *Physical Review Letters*, 57(21):2607–2609.

Swersky, K., Buchman, D., Freitas, N. D., Marlin, B. M., et al. (2011). On autoencoders and score matching for energy based models. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1201–1208.

Tamura, S. and Waibel, A. (1988). Noise reduction using connectionist models. In *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, pages 553–556. IEEE.

Tappen, M. F. (2007). Utilizing variational optimization to learn markov random fields. In *CVPR*. IEEE Computer Society.

Taylor, G. W., Hinton, G. E., and Roweis, S. (2007). Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems*, volume 19, Cambridge, MA. MIT Press.

Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the International Conference on Machine Learning.*

Tieleman, T. and Hinton, G. (2009). Using Fast Weights to Improve Persistent Contrastive Divergence. In *Proceedings of the 26th international* *conference on Machine learning*, pages 1033–1040. ACM New York, NY, USA.

Tremain, T. E. (1982). The government standard linear predictive coding algorithm: Lpc-10. *Speech Technology*, 1(2):40–49.

Triefenbach, F., Demuynck, K., and Martens, J.-P. (2014). Large vocabulary continuous speech recognition with reservoir-based acoustic models. *IEEE SIGNAL PROCESSING LETTERS*, 21(3):311–315.

Triefenbach, F., Jalalvand, A., Schrauwen, B., and Martens, J.-P. (2010). Phoneme recognition with large hierarchical reservoirs. In *NIPS*, pages 2307–2315.

Varin, C., Reid, N. M., and Firth, D. (2011). An overview of composite likelihood methods. *Statistica Sinica*, 21(1):5–42.

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P. A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine learning*, ICML '08, pages 1096–1103, New York, NY, USA. ACM.

Vinyals, O., Ravuri, S., and Povey, D. (2012). Revisiting Recurrent Neural Networks for Robust ASR. In *ICASSP*.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37(3):328–339.

Wainwright, M. J. (2006). Estimating the wrong graphical model: Benefits in the computation-limited setting. *The Journal of Machine Learning Research*, 7:1829–1859.

Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1-2):1–305.

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2008). Gaussian process dynamical models for human motion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):283–298.

Wang, N., Melchior, J., and Wiskott, L. (2012). An analysis of gaussian-binary restricted boltzmann machines for natural images. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 287–292.

Wedderburn, R. W. (1974). Quasi-likelihood functions, generalized linear models, and the gauss-newton method. *Biometrika*, 61(3):439–447.

Welling, M., Hinton, G. E., and Osindero, S. (2003). Learning sparse topographic representations with products of student-t distributions. In *Advances in Neural Information Processing Systems 15*, pages 1359–1366, Cambridge, MA. MIT Press.

Williams, C. K. and Hinton, G. E. (1991). Mean field networks that learn to discriminate temporally distorted strings. In *Connectionist models: Proceedings of the 1990 summer school*, pages 18–22. San Mateo, CA: Morgan Kaufmann.