On Layer-Wise Representations in Deep Neural Networks

vorgelegt von Dipl.-Ing.
Grégoire Montavon

Von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Manfred Opper
Gutachter:	Prof. Dr. Klaus-Robert Müller
	Prof. Dr. Yoshua Bengio
	Dr. Léon Bottou

Tag der wissenschaftlichen Aussprache: 29. Oktober 2013

Berlin 2013
D83

Abstract

On Layer-Wise Representations in Deep Neural Networks It is well-known that deep neural networks are forming an efficient internal representation of the learning problem. However, it is unclear how this efficient representation is distributed layer-wise, and how it arises from learning. In this thesis, we develop a kernel-based analysis for deep networks that quantifies the representation at each layer in terms of noise and dimensionality. The analysis is applied to backpropagation networks and deep Boltzmann machines, and is able to capture the layer-wise reduction of noise and dimensionality. The analysis also reveals the disrupting effect of learning noise, and how it prevents the emergence of highly sophisticated deep models.

Zusammenfassung

Schichtweise Repräsentationen in Tiefen Neuronalen Netzen Es ist bekannt, dass tiefe neuronale Netze eine effiziente interne Repräsentation des Lernproblems bilden. Es ist jedoch unklar, wie sich diese effiziente Repräsentation über die Schichten verteilt und wie sie beim Lernen entsteht. In dieser Arbeit entwickeln wir eine Kernel-basierte Analyse für tiefe Netze. Diese Analyse quantifiziert die Repräsentation in jeder Schicht in Bezug auf Rauschen und Dimensionalität. Wir wenden die Analyse auf Backpropagation-Netze und tiefe Boltzmann-Maschinen an und messen die schichtweise Reduzierung von Rauschen und Dimensionalität. Die Analyse zeigt auch den störenden Einfluss des Lernrauschens: Dieses verhindert die Entstehung komplexer Strukturen in tiefen Modellen.

Acknowledgements

First and foremost, I would like to thank Klaus-Robert Müller for being an exceptional advisor and for providing me with great and challenging opportunities. This includes the overall idea of this thesis, but also the possibility to take part to an interdisciplinary project on quantum chemistry. In addition, Klaus drive and enthusiasm have been an invaluable source of motivation and inspiration.

Special thanks also go to Mikio Braun and O. Anatole von Lilienfeld for advising me. I particularly appreciated their patience at teaching me the basics of kernel-based learning and quantum chemistry, and their willingness to reach beyond their own field of research. I am very grateful to the multiple authors of the second edition of the book "Neural Networks: Tricks of the Trade" for contributing high quality chapters. I thank my colleagues in TU Berlin for inspiring discussions. This includes my former office mates: Pascal Lehwark, Marius Kloft, Nico Görnitz, Gunnar Kedenburg, Duncan Blythe, and colleagues: Katja Hansen, Franziska Biegler, Matthias Rupp, Tammo Krüger, Siamac Fazli, Andreas Ziehe, Stefan Chmiela and Danny Panknin. On the personal side, I thank my family for their continuing support. I thank Hanna for her patience and support.

The first year of my thesis was supported by a scholarship from Nokia, which was later extended as a TU Berlin research stipend. The second part of my thesis was supported by Deutsche Forschungsgemeinschaft (DFG grant MU 987/17-1).

Contents

1. Introduction

A large number of machine learning problems are believed to benefit from hierarchical deep representations. Unlike simple similarity-based methods, deep learning aims to solve the prediction problem by applying a sequence of global nonlinear transformations to the input. This provides a more efficient way to fight the curse of dimensionality (Bengio and LeCun 2007).

In many applications, we need to predict a small set of output variables (e.g. a binary outcome) from a large set of input variables. The learning algorithm has to find and extract the small subset of variations in the data, that are relevant for prediction. A first example is handwritten digit recognition: In such problem, the symbol to be recognized (0–9) can take a variety of forms in the input space (translations, rotations). This variability has to be filtered out. A second example in the field of quantum chemistry is the prediction molecular atomization energy: From a high-dimensional geometric description of the molecule (Rupp et al. 2012), the system is progressively coarse-grained to produce the desired molecular electronic property.

Deep networks have been proposed to solve these dimensionality reduction problems and have been applied with success (LeCun et al. 1998, Ciresan et al. 2010, Montavon et al. 2012b). Among proposed methods, we can distinguish (1) supervised methods such as backpropagation networks (Rumelhart et al. 1986) where the mapping between input and output is learned explicitly by providing labels and backpropagating errors, and (2) unsupervised methods such as deep Boltzmann machines (Salakhutdinov and Hinton 2009), where the unsupervised learning algorithm creates as a by-product a low-dimensional representation of task in top layers.

An important requirement of deep learning algorithms for dimensionality reduction is to be able to progressively reduce the dimensionality of data without introducing noise. The convolutional neural network (LeCun 1989) is an example of a model that structurally incorporates several levels of dimensionality reduction through spatial pooling. Stacked autoencoders (Hinton et al. 2006) are able to reduce the dimensionality of data by progressively simplifying the representation in a greedy layer-wise fashion. Unfortunately, such progressive dimensionality reduction is more delicate to achieve in a setting where all layers of representation are learned jointly and where no predefined structure is available.

The goal of this thesis is to develop new methods based on the analysis of the representation at each layer, in order to better understand the intrinsic difficulties of learning multiple layers of representation at the same time. The main contribution of this thesis is to quantify representations at each layer of a deep network

in terms of noise and dimensionality. Viewing a deep network as a composition of functions $y = f_L \circ \ldots \circ f_1(x)$, we can construct a kernel at each layer:

$$\begin{aligned} k_0(x, x') &= k_{\mathrm{RBF}}(x, x') \\ k_1(x, x') &= k_{\mathrm{RBF}}(f_1(x), f_1(x')) \\ &\vdots \\ k_L(x, x') &= k_{\mathrm{RBF}}(f_L \circ \ldots \circ f_1(x), f_L \circ \ldots \circ f_1(x')). \end{aligned}$$

We then apply for each kernel the method of Braun et al. (2008), that looks at the correlation between the vector of labels and the kernel principal components. Of these correlations, only a few (corresponding to the leading eigenvectors) contain signal, and the remaining ones are essentially noise. This observation provides the basis for computing an estimate of the noise and dimensionality of the problem at each layer.

Our kernel-based analysis aims to provide a richer feedback on the training process than a simple top-layer error estimate. By this, we aim to identify statistical and computational bottlenecks in practical implementations of deep networks. For example, a sharp dimensionality reduction in the first layers combined with a noise increase may make learning more difficult by forcing subsequent layers to denoise the low-dimensional representation.

The kernel analysis is applied to test several hypotheses on the layer-wise evolution of the representation in backpropagation networks and deep Boltzmann machines. These hypotheses arise from the careful examination of small deep networks made of few units, and are extended by deduction to larger models. After showing that these hypotheses can be framed in terms of noise and dimensionality within our kernel-based framework, we train large deep networks on real data and estimate the noise and dimensionality of representations at each layer. Our analysis is then able to test the validity of these hypotheses at real scale.

Description of Chapters

This thesis consists of four main chapters:

- **Chapter 2 (Kernels and Deep Networks):** In this chapter, we review the kernel principal component analysis and the relevant dimension estimates, an analysis that measures the noise and dimensionality of a kernel with respect to a learning problem. Then, we explain how the analysis can be used to analyze representations at each layer of a deep network, and how it can help to better understand deep representations.

- **Chapter 3 (Backpropagation Networks):** In this chapter, we first review the backpropagation network and its training procedure. Then, several hypotheses on the layer-wise evolution of the representation are formulated,

including the layer-wise denoising and dimensionality reduction of the learning problem, and an early discrimination effect caused by the presence of learning noise. Hypotheses are tested with the kernel analysis of Chapter 2.

- **Chapter 4 (Deep Boltzmann Machines):** In this chapter, we apply the same kernel analysis to deep Boltzmann machines (DBMs) and stacked restricted Boltzmann machines (stacked RBMs). We first review the DBM and how the learned solution can be understood in terms of modes of interaction. Then, based on these modes of interaction, several hypotheses are formulated on the layer-wise evolution of the representation. In particular, we predict the emergence of invariant top-layer representations in DBMs, and a layer-wise oscillation effect in stacked RBMs.

- **Chapter 5 (Predicting Molecular Electronic Properties):** In this chapter, we review an application of machine learning to quantum chemistry. We discuss the multiple challenges that arise from solving the problem with neural networks, in particular, the difficulty of finding a good representation of molecules, and the special wide-range and multiscale nature of the problem. We propose several techniques including random Coulomb matrices and input binarization in order to overcome these difficulties. We finally apply these techniques to train successfully a neural network on the prediction task.

Main Contributions of the Thesis

The thesis includes four main contributions:

- **Application of RDE to Deep Networks:** The main contribution of this thesis is the application of the relevant dimensionality estimates (RDE) method of Braun et al. (2008) to analyze layer-wise representations in deep networks in terms of *noise* and *dimensionality*. We motivate the use of RBF kernels as a proxy to measure these two quantities at each layer of the network. We perform extensive experiments showing the capacity of the method to capture and quantify the layer-wise transformation of representations in deep networks.

- **Deductive Methods for Understanding Layer-Wise Representations:** In backpropagation networks, the qualitative analysis of the response function of a simple two-unit deep network allows us to deduce an *early discrimination* effect in presence of learning noise. For deep Boltzmann machines and stacked RBMs, we propose the use of binomial Boltzmann machines as a proxy to model their macroscopic complex behavior. We identify several *modes of interaction* in DBMs that we call "preserve" and "complement" and that can be quantified using the *layer interaction number*. This novel analysis allows us to deduce the emergence of invariance in DBMs, a

layer-wise oscillation effect in stacked RBMs, and to better understand the effect of learning noise on the quality of emerging representations.

- **Enhanced Training Procedures for Deep Boltzmann Machines:** Our analysis of representations in DBMs have led us to develop enhanced methods for training these models. This includes the *centering trick*, a technique that consists of reparameterizing the energy of the DBM as a function of centered states. The additional stability of the resulting learning algorithm is verified by the analysis of the Hessian of the objective, showing that centered DBMs have a much better condition number than their non-centered counterpart. The training procedure is further enhanced by *structural sampling*, a recursive sampling procedure that further reduces learning noise in top layers at reasonable computational cost.

- **Learning Molecular Electronic Properties with Neural Networks:** We investigate the use of several neural network techniques for addressing the specific challenges posed by the problem of predicting molecular electronic properties. We propose the *random Coulomb matrices* (a technique to extract more statistical information from the data), and *input binarization*, that we show to better handle some scaling issues with the problem representation. Applying these techniques leads to a significant performance improvement on our learning task.

Relation to Previously Published Work

Many results in this thesis have already been previously published in conference proceedings, books and journals. They are taken mainly from the following papers:

- Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. Kernel analysis of deep networks. *Journal of Machine Learning Research*, 10:2579–2597, September 2011

- Grégoire Montavon, Katja Hansen, Siamac Fazli, Matthias Rupp, Franziska Biegler, Andreas Ziehe, Alexandre Tkatchenko, O. Anatole von Lilienfeld, and Klaus-Robert Müller. Learning invariant representations of molecules for atomization energy prediction. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 449–457, 2012b

- Grégoire Montavon and Klaus-Robert Müller. *Deep Boltzmann Machines and the Centering Trick*, volume 7700 of *LNCS*, chapter 25, pages 621–637. Springer, 2nd edition, 2012

- Grégoire Montavon, Mikio L. Braun, Tammo Krueger, and Klaus-Robert Müller. Analyzing local structure in kernel-based learning: Explanation,

complexity, and reliability assessment. *Signal Processing Magazine, IEEE*, 30(4):62–74, 2013a

- Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. A study of representations in deep Boltzmann machines. under review, 2013b

The thesis also borrows ideas, results and figures from the following papers:

- Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. Layer-wise analysis of deep networks with Gaussian kernels. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1678–1686, 2010

- Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. Deep Boltzmann machines as feed-forward hierarchies. *Journal of Machine Learning Research - Proceedings Track*, 22:798–804, 2012a

- Grégoire Montavon and Klaus-Robert Müller. Neural networks for computational chemistry: Pitfalls and recommendations. *MRS Online Proceedings Library*, 1523, 2013

- Katja Hansen, Grégoire Montavon, Franziska Biegler, Siamac Fazli, Matthias Rupp, Matthias Scheffler, O. Anatole von Lilienfeld, Alexandre Tkatchenko, and Klaus-Robert Müller. Assessment and validation of machine learning methods for predicting molecular atomization energies. *Journal of Chemical Theory and Computation*, 9(8):3404–3419, 2013

- Grégoire Montavon, Matthias Rupp, Vivekanand Gobre, Alvaro Vazquez-Mayagoitia, Katja Hansen, Alexandre Tkatchenko, Klaus-Robert Müller, and O. Anatole von Lilienfeld. Machine learning of molecular electronic properties in chemical compound space. *New Journal of Physics*, 15(9): 095003, 2013c

2. Kernels and Deep Networks

There is a simple relation between kernels and deep networks: They both produce a nonlinear representation of data $\phi(x)$—implicit for kernels, or explicit for deep networks—in which the problem is supposed to be more easily solvable. In this chapter, we explore how the kernel framework can be used to better understand the functions learned by deep networks.

A generic way of defining a representation of data is through a kernel. The kernel maps each pair of samples (x, x') to a score $k(x, x')$, typically, some measure of similarity. By definition, the kernel must satisfy a positive semi-definiteness constraint where $\sum_{ij} \alpha_i k(x_i, x_j)\alpha_j > 0$ for all vectors α, and for all sequences of data points $x_1, \ldots, x_n$. If this condition is satisfied, there exists a nonlinear feature map $\phi(x)$ such that the Euclidean dot product in such space is equivalent to the kernel operator, that is:

$$k(x, x') = \phi(x)^\top \phi(x').$$

This correspondence between kernel and dot product in the feature space is known as the *kernel trick*. The advantage of this feature map representation is that many ideas developed in the context of linear models such as large margin classification and principal component analysis can be extended to nonlinear models. In particular, any model of the form $f(x) = \sum_i \alpha_i k(x, x_i)$ can be rewritten as $f(x) = \beta^\top \phi(x)$. Conversely, many algorithms developed for linear models can be "kernelized", that is, transformed in a way that only the kernel operator needs to be accessed. This avoids to work explicitly in the possibly high-dimensional feature space. See, for example, Müller et al. (2001) for an introduction to kernel-based learning algorithms.

In the next sections, we review the *kernel principal component analysis* (Schölkopf et al. 1998), and the *relevant dimensionality estimates* proposed by Braun et al. (2008). Then, based on these two methods, a comprehensive framework for analyzing layer-wise representations in deep networks is introduced.

2.1. Kernel Principal Component Analysis (kPCA)

Kernel principal component analysis (Schölkopf et al. 1998) finds the main directions of variance in the kernel feature space $\phi(x)$. For finite data sets, such kernel principal components can be approximated empirically. For this, we first collect a

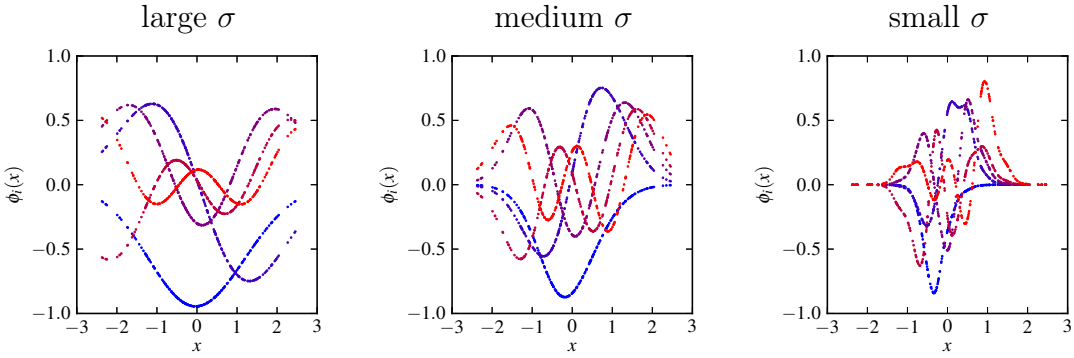


Figure 2.1.: Interpretation of kernel principal components for Gaussian kernels with different scale parameters σ. The input distribution $p(x)$ is chosen to be normal with mean zero and unit variance. Plots show the five leading kernel principal components (shown from blue to red, and scaled by their corresponding eigenvalues).

dataset of n samples,

$$\mathcal{D} = \{x_1, \ldots, x_n\},$$

drawn i.i.d. from some probability distribution $p(x)$. Then, we compute the Gram matrix K of size $n \times n$ associated to this dataset, where $K_{ij} = k(x_i, x_j)$. In general, the feature map $\phi(x)$ produced by the kernel is not necessarily centered. The strict version of PCA requires centering ($\sum_i \phi(x_i) = 0$). Centering of the feature map can be done by centralizing the Gram matrix: $K^c_{ij} = K_{ij} - \frac{1}{n}\sum_j K_{ij} - \frac{1}{n}\sum_i K_{ij} + \frac{1}{n^2}\sum_{ij} K_{ij}$. However, because of the relatively large number of kernel principal components considered here, the effect of centering is minor. In our application, centering may in fact discard useful information. In this work, the kernel will always be taken non-centered. Once the Gram matrix K is computed, we solve the linear eigenvalue problem:

$$Ku = \lambda u$$

Due to the positive semi-definiteness of the kernel, all eigenvalues $\lambda_1, \ldots, \lambda_n$ are nonnegative. The eigenvectors $u_1, \ldots, u_n$, sorted by decreasing associated eigenvalue, are the kernel principal components. Each kernel principal component is an n-dimensional vector and can be interpreted as a map from each data point in the dataset to a certain value. In the continuous case, kernel principal components become infinite-dimensional and can be interpreted as a function in the input space $u_i(x)$ weighted by the input distribution $p(x)$. Kernel principal components can be used as a feature space representation by rescaling them as $\phi_i(x) = u_i(x) \cdot \sqrt{\lambda_i}$.

An interpretation of kernel principal components is given in Figure 2.1 for a Gaussian kernel of parameter σ and an input probability distribution $\mathcal{N}(0, 1)$. The five leading kernel principal components are shown ordered from blue to red,

for scales σ corresponding to the 0.1, 0.3, and 0.6 quantiles of the distribution of pairwise distances in the dataset. We can observe that for a given kernel, the kernel principal components have an increasing level of detail (or higher frequencies).

A second observation that can be made from Figure 2.1 is that principal components vary significantly depending on the value of the kernel scale parameter σ: Kernels with large scale parameters tend to produce "risky" extrapolation on the fringe of the data distribution. On the other hand, a small scale parameter produces kernel principal components that are bounded in the input space.

2.2. Dimensionality and Noise Estimates

We present here an analysis by Braun et al. (2008) that looks at the structure of the learning problem in the space of kernel principal components. Learning predictive models is often achieved by joint minimization of two terms,

$$\min_{f \in \mathcal{F}} \quad R_{\text{emp}}(f, \mathcal{D}) + \Omega(f),$$

where $f \in \mathcal{F}$ is a function taken from the set of functions that the learning machine can implement, and $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ is a dataset of input-output pairs. The fitting term $R_{\text{emp}}(f, \mathcal{D})$ encourages functions f that predict accurately y_i from x_i. On the other hand, the regularization term $\Omega(f)$ penalizes too complicated functions.

When applied to the kernel-based models, such regularization scheme often ignores an important structural property of kernel feature spaces: Braun et al. (2008) found that for a given learning problem satisfying some smoothness properties, the label information is implicitly contained in a limited number of kernel principal components and the remaining components contain essentially noise. More precisely, the projection of the label vector y onto the leading components $Z(i) = u_i^\top y$ typically takes the form

$$Z(i) = S(i)_{i>d} + N(i) \tag{2.1}$$

where $S(i)$ is a signal component that is restricted to a small number of dimensions d, and $N(i)$ is a noise baseline that has similar statistical properties for all i. This decomposition provides an estimate of noise and dimensionality, and allows us to distinguish between four different scenarios: (1) high noise and high dimensionality, (2) high noise and low dimensionality, (3) low noise and high dimensionality and (4) low noise and low dimensionality. Scenario 1 is the worst possible. Scenario 4 is the best.

An important aspect of the noise and dimensionality estimates proposed by Braun et al. (2008) is the advantageous convergence property of the signal component $S(i)$ with respect to the infinite sample case. In essence, Braun (2006) and Braun et al. (2008) show that the signal relevant part $S(i)$ of the spectrum $Z(i)$ is close to its asymptotic counterpart (when $n \to \infty$), and that projection error for

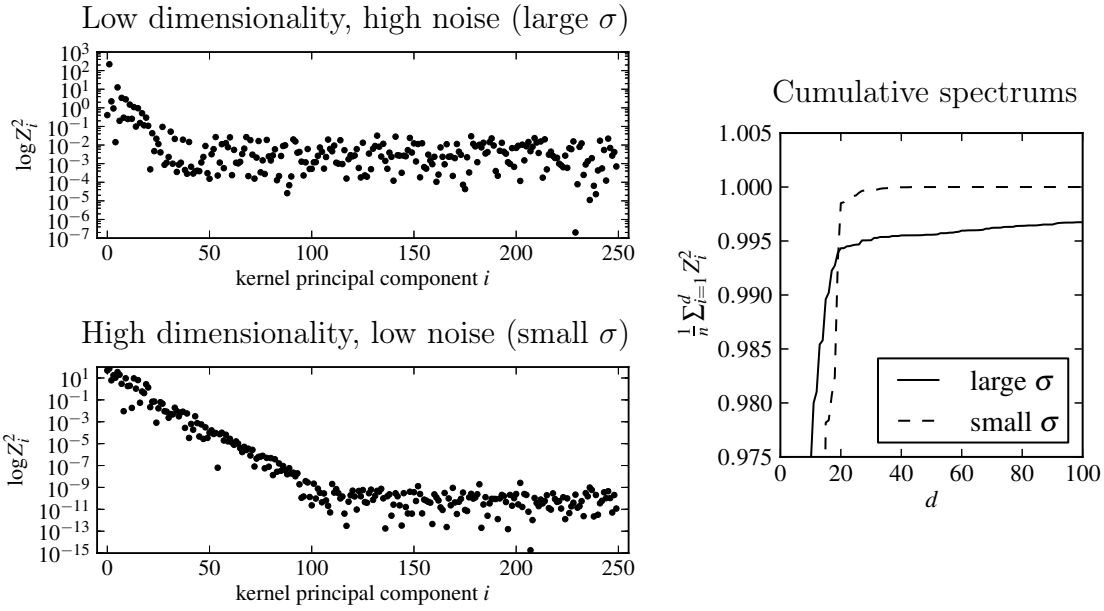


Figure 2.2.: RDE analysis for estimating noise and dimensionality on two of the kernels of Figure 2.1. We consider the toy problem of modeling the function $y = \tanh(10 \cdot x)$. On the left, the spectrum coefficients Z_i are plotted for each kernel principal component i. On the right, the same signal is plotted in a cumulative fashion. While the spectrum is instructive of the signal-noise structure of the problem, cumulative plots are generally more appropriate for comparing kernels quantitatively.

non-leading principal components is typically much smaller than for leading components. The fact that noise and dimensionality estimates are both invariant and robust to sample size implies that the analysis can be used to compare representations with very different sampling rates. In particular, Cadieu et al. (2013) show the importance of such property when comparing representations in the brain (low sampling rate) and in artificial neural networks (high sampling rate).

It is intuitive to discuss noise and dimensionality in the context of Gaussian kernels: As we have seen in the example of Figure 2.1, the eigenspace of Gaussian kernels is formed by a basis of increasingly high-frequency components. Thus, the analysis tells us that the only part of the signal that can be recovered is its low frequencies, and that the high frequencies should be filtered out, as they are overwhelmingly dominated by noise. We discuss two different scenarios:

- *Low dimensionality, high noise:* In this setting, a narrow signal component $S(i)$ is emerging from a relatively high noise bed $N(i)$. Gaussian kernels with large scale σ fall into that category: Their rigidity and global nature allow to capture a large amount of label information within their principal components. However the same properties also prevent the kernel from adapting to more complex decision functions. The noise-dimensionality profile for one

of such kernels is given in Figure 2.2 (top). For this type of kernels the performance of the learning algorithm cannot be improved by simply adding more samples since the problem is low-dimensional. Instead, a finer-grained kernel (e.g. with smaller scale) has to be provided.

- *High dimensionality, low noise:* In this setting, the signal component $S(i)$ spans a large number of dimensions, but the noise bed $N(i)$ is also lower. Gaussian kernels with small scale σ fall into that category: The small scale provides the granularity necessary to resolve complex functions. However, due to their local nature, these kernels also lack the ability to capture in a compact way the most salient factors of variation in the function. The noise-dimensionality profile of such kernel is given in Figure 2.2 (bottom). In this context, the learning problem can be solved more accurately by adding a large amount of labeled samples.

2.2.1. Practical Considerations

We have reviewed some of the main ideas behind the dimensionality and noise estimates of Braun et al. (2008), and gave an intuition for the analysis in the context of Gaussian kernels. In practice, in order to deal with a wider range of applications such as multidimensional regression, classification, comparing representations, or local analysis, the original method has to be adapted in several ways. We first recapitulate the sequence of computations that lead to the spectrum coefficients $\{Z_i^2\}$, on which the analysis is based. Let $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a dataset of n samples, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. After computing the non-centralized kernel Gram matrix K where $K_{ij} = k(x_i, x_j)$, kernel principal components $u_1, \ldots, u_n$ are obtained by solving the eigenvalue equation

$$Ku = \lambda u.$$

Let $u_1, \ldots, u_n$ be sorted by decreasing eigenvalues. The coefficient Z_i corresponding to the ith principal component is then obtained as:

$$Z_i^2 = \sum_j (u_{ij} \cdot y_j)^2.$$

So far, the label vector y was assumed to be one-dimensional. For multidimensional regression problems or classification problems, the label information is best represented instead as a multidimensional vector $y \in \mathbb{R}^c$. In the classification case, y is simply a vector of class indicators, or alternatively, class probabilities. Let Y be the label matrix of size $n \times c$. The spectrum Z is now computed as

$$Z_i^2 = \sum_k (\sum_j u_{ij} \cdot Y_{jk})^2, \tag{2.2}$$

that is, by summing out the contributions of label projections onto each dimension of the output space. In practice, it can be simpler or more intuitive to look

directly at the prediction error of a linear model based on the d leading principal components. Spectrum coefficients $\{Z_i^2\}$ can be related to the error residuals as:

$$e(d) = ||\sum_{i=1}^{d} u_i u_i^\top Y - Y||_F^2 = \sum_{i=d+1}^{n} Z_i^2. \tag{2.3}$$

A flat spectrum Z^2 (e.g. noise baseline) will be captured by a linear decay of $e(d)$. Thus, assuming a clear signal-noise decomposition of the spectrum Z^2, the error curve $e(d)$ will decrease sharply for small d and have a slower linear decrease for large d. Example of error curves are given in Figure 2.3 in the context of analyzing deep networks. A sample code for the kernel analysis is available at `http://gregoire.montavon.name/code/kanalysis.py`.

We have seen in the example of Figure 2.1 that the eigenfunctions of a Gaussian kernel are taking the form of wavelets of increasing frequencies. Such basis is appropriate for modeling smooth functions of the input space. This is typically the case for regression problems. This is also the case for noisy classification problems where we wish to approximate the posterior $p(y|x)$. However, for non-noisy classification problems, the function $p(y|x)$ takes the form of wide class plateaus separated by sharp cliffs corresponding to the class boundaries. The high-frequency content of such class indicator functions implies that, if applying this analysis straight out-of-the-box, the classification problem would always look high-dimensional with no clear cutoff between noise and dimensionality. Two solutions are proposed to overcome this problem: First, we can consider an alternative probability distribution

$$p'(x) \propto p(x) \cdot 1_{x \in \mathcal{B}}$$

where $\mathcal{B}$ is the region of the input space representing the class boundary. This ensures that kernel PCA does not waste its capacity to model the plateaus that are irrelevant for classification. Alternatively, Montavon et al. (2011) proposed an *a posteriori* method that consists of fitting a logistic model β on top of the d kernel principal components built from the original probability distribution $p(x)$. We first find the logistic model parameter $\beta^\star$ of dimensionality $d \times c$ that solves the optimization problem

$$\max_\beta \prod_{i=1}^{n} \text{softmax}([\sum_{j=1}^{d} u_j \beta_j^\top]_i)_{y_i}.$$

The classification error is in turn computed as $e(d) = \frac{1}{n}\sum_{i=1}^{n} 1_{\hat{y}_i \neq y_i}$ where $\hat{y}_i = \text{argmax}([\sum_{j=1}^{d} u_j \beta_j^\top]_i)$. Here, the eigenfunctions of increasing frequencies are defining implicit separating surfaces in the input space that can be combined to form complex classification boundaries.

2.2.2. Extension to Similarity and Local Reliability Estimates

The method can be easily extended to measure similarity between two different kernels k and k'. The measure of similarity $s(d)$ is obtained by projecting the principal components u_i of the kernel k onto the principal components u'_j of the other kernel k':

$$s(d) = \frac{1}{n} \sum_{i=1}^{d} \sum_{j=1}^{d} (u_i^\top u'_j)^2. \tag{2.4}$$

This measures the total correlation between the subspaces spanned by the d leading components of each kernel. The similarity measure has the boundary values $s(0) = 0$ and $s(n) = 1$. A similarity curve $s(d)$ that increases quickly with d indicates that the two kernels are similar. We note that such measure of similarity is symmetric, that is, the kernels can be swapped without changing the value of $s(d)$.

Finally, Montavon et al. (2013a) have developed a local extension of dimensionality and noise estimates, which consists of performing the analysis on a local kernel defined as $k_\xi(x, x') = k(\xi, x) \cdot k(x, x') \cdot k(x', \xi)$. This last extension is presented in Appendix C.

2.3. Analyzing Deep Networks with Kernels

Deep networks are characterized by being composed of multiple layers of representation. In this respect, they are able to implement some functions more efficiently than shallow architectures such as linear models or simple RBF kernels. A well-known problem where depth is necessary is the bit-parity problem. The bit-parity problem consists of determining whether a sequence of bits

0	1	1	0	0	0	1	1	0	1	...

of length d contains an even or odd number of "1"s. Restricting the class of models to all possible compositions of elementary functions "AND" and "OR", it is known for this problem that the most efficient solution in terms of number of units consists of a deep sequence of operations made of at least $O(\log(d))$ layers. See Utgoff and Stracuzzi (2002) and Bengio and LeCun (2007) for a discussion.

In addition, there is some evidence that some of the practical problems that we are trying to solve using machine learning also require depth. For example, studies by Hubel and Wiesel (1959), Serre et al. (2005), Wibisono et al. (2010) provide compelling evidence that deep processing also takes place in the brain. In particular, dimensionality reduction is achieved through a hierarchy of feature detectors (simple cells) interleaved by pooling units (complex cells). It is often argued, that if such processing were to be efficiently achieved with less layers of representation, deep models would be difficult to defend from an evolutionary

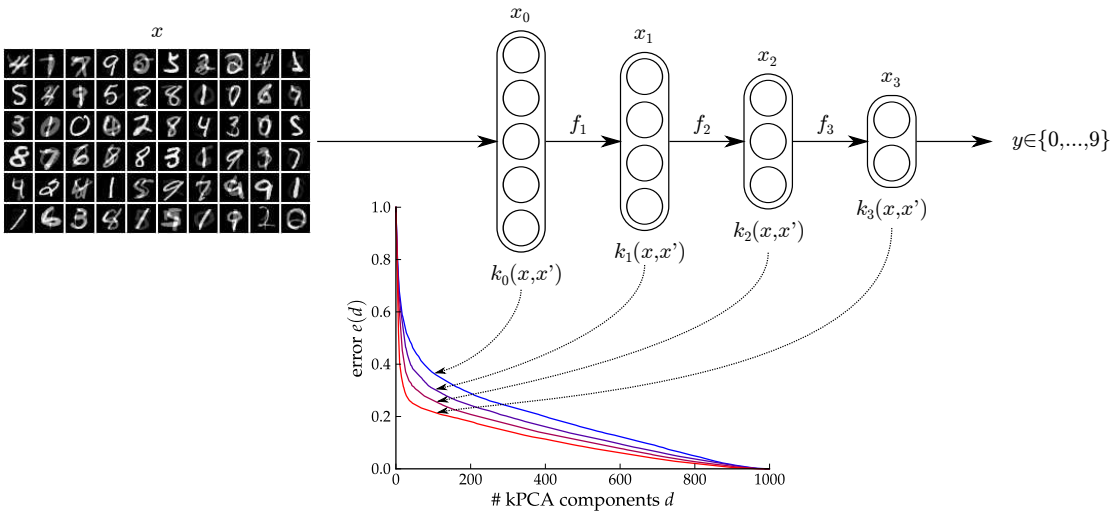


Figure 2.3.: Overview of the kernel analysis of a deep network. The input is propagated through the deep network. A kernel is built at each layer and produces the error curve $e(d)$ from which noise and dimensionality can be estimated.

standpoint, due to inherent latencies caused by processing input with multiple layers.

As a current limitation, it is sometimes difficult to take advantage of the deep architecture, even in the case where the problem is known to be deep. As we will see in the next chapters, there are many factors that influence qualitatively the solutions learned by a deep network. Being able to characterize and quantify these qualitative differences, and to relate quantitatively the parameters of the learning algorithm to the properties of emerging representation is therefore of crucial importance in order to develop more informed deep learning procedures.

2.3.1. Layer-Wise Analysis of Deep Networks

We now introduce the main analytic tool used in this thesis: Quantifying representations at each layer of a deep network in terms of noise and dimensionality using the method of Section 2.2. An overview of the analysis is given in Figure 2.3 for a four-layer deep network that has been trained to classify noisy handwritten digits. First, the deep network is abstracted as a composition of functions:

$$f(x) = f_L \circ \cdots \circ f_1(x).$$

For example, in a simple multilayer perceptron, each subfunction would consist of a linear transformation followed by an element-wise nonlinearity.

A kernel is constructed at each layer:

$$\begin{aligned} k_0(x, x') &= k_{\text{RBF}}(x, x') \\ k_1(x, x') &= k_{\text{RBF}}(f_1(x), f_1(x')) \\ &\vdots \\ k_L(x, x') &= k_{\text{RBF}}(f_L \circ \cdots \circ f_1(x), f_L \circ \cdots \circ f_1(x')). \end{aligned}$$

The choice of an RBF kernel will be motivated later in Section 2.3.3. We feed the network with some data coming i.i.d. from the input distribution $p(x)$. Using this data, the kernel at each layer can compute its own Gram matrix and apply the procedure described in Section 2.2.1.

As a result, an error curve $e(d)$ is obtained for each layer. Each error curve characterizes the structure of the learning problem at a given layer. Error curves are characterized by a noise baseline (the slope of $e(d)$ for large d) and a dimensionality (the value d after which the error $e(d)$ starts to decay linearly). In this example, the deep network achieves a layer-wise reduction of both noise and dimensionality.

2.3.2. Understanding Noise and Dimensionality Reduction

We now look at how the basic computational elements of the network (sigmoid functions and linear projections) are transforming the problem representation in terms of noise and dimensionality. Figure 2.4 (left) illustrates the concept of noise and dimensionality reduction for simple linear kernels. In this example, we consider three data points x_1, x_2, x_3 with their associated label (red = +1, blue = -1). Dimensionality reduction can be achieved by rescaling principal components such that the discriminative subspace is moved from the second principal component to the first principal component. Such rescaling can be achieved, for example, by linear transformation. If the problem is linearly unsolvable (i.e. there is no linear relation between input and label), the kernel perceives the task as noisy. In such case, only a nonlinearity such as the sigmoid function is able to distort the representation in order to reduce noise, as shown in Figure 2.4 (bottom left).

In addition to noise and dimensionality reduction, combined effects where low dimensionality is traded for low noise, or vice-versa, are also possible: A noise reduction combined with a dimensionality increase may occur if a sharp nonlinear transformation that uncovers a more complex problem, was perceived initially as noise by the kernel. Such scenario is depicted in Figure 2.4 (middle). Conversely, a too aggressive dimensionality reduction may discard label-informative components and introduce noise in the problem. Such scenario is depicted in Figure 2.4 (right).

2.3.3. RBF Kernels: A Proxy for Looking at Deep Networks

It remains to demonstrate how noise and dimensionality reduction can be understood in the context of deep networks, and why RBF kernels are appropriate for

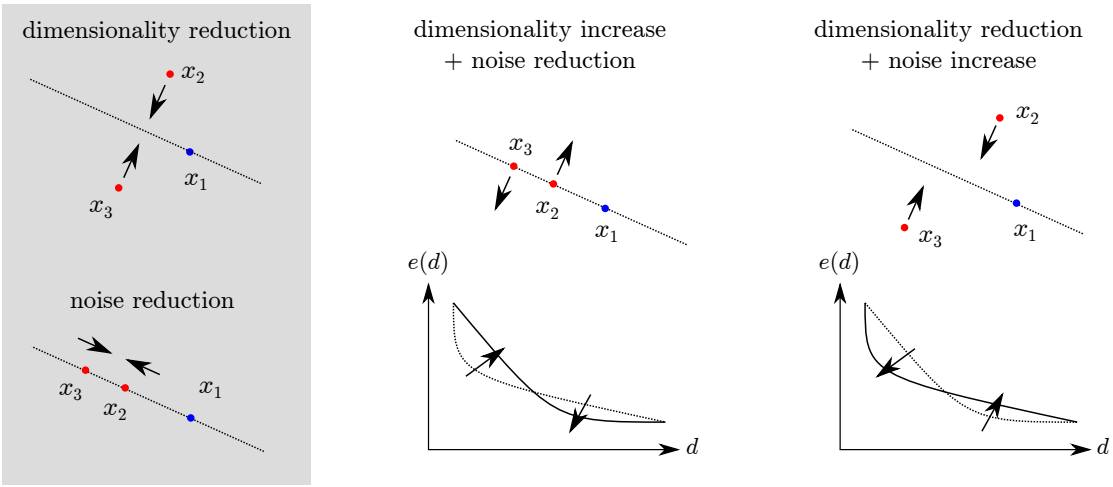


Figure 2.4.: On the left, interpretation of dimensionality and noise reduction for a linear kernel. On the right, example of transformations that combine a reduction or increase of noise and dimensionality. Both examples are shown along with a cartoon of their RDE analysis.

our layer-wise analysis: First, as we have seen in the example of Figure 2.1, principal components of an RBF kernel take the form of smooth functions of increasing frequency in the input space. Given that the set of leading principal components spans the set of low-frequency functions, the representation at each location can be understood as essentially locally linear. As a consequence, the mapping performed by deep networks can be viewed locally in the same way as in Section 2.3.2. In particular, we can see the original monolithic nonlinear function $f_l(x)$ at layer l as:

$$f_l(x) = \begin{cases} V_1 \cdot \mathrm{sigm}(W_1\, x) & \text{if } x \in \mathcal{C}_1 \\ \vdots & \\ V_N \cdot \mathrm{sigm}(W_N\, x) & \text{if } x \in \mathcal{C}_N. \end{cases}$$

where $\mathcal{C}_1, \ldots, \mathcal{C}_N$ are small disjoint regions that form together a partition of the mapped input space at layer $l-1$, and $\{V_i \cdot \mathrm{sigm}(W_i\, x)\}_{i=1}^N$ are a set of smallish nonlinear networks implementing the basic transformations shown in Figure 2.4. In each small region $\mathcal{C}_i$, the linear kernel is therefore a good proxy to measure the evolution of noise and dimensionality from one layer to another.

A second argument in favor of RBF kernels is that they produce a statistical aggregate of noise and dimensionality at different locations. The easiest way to understand this is to consider a dataset made of N compact and well separated clusters, so that an RBF kernel views the data essentially as a set of local kernels.

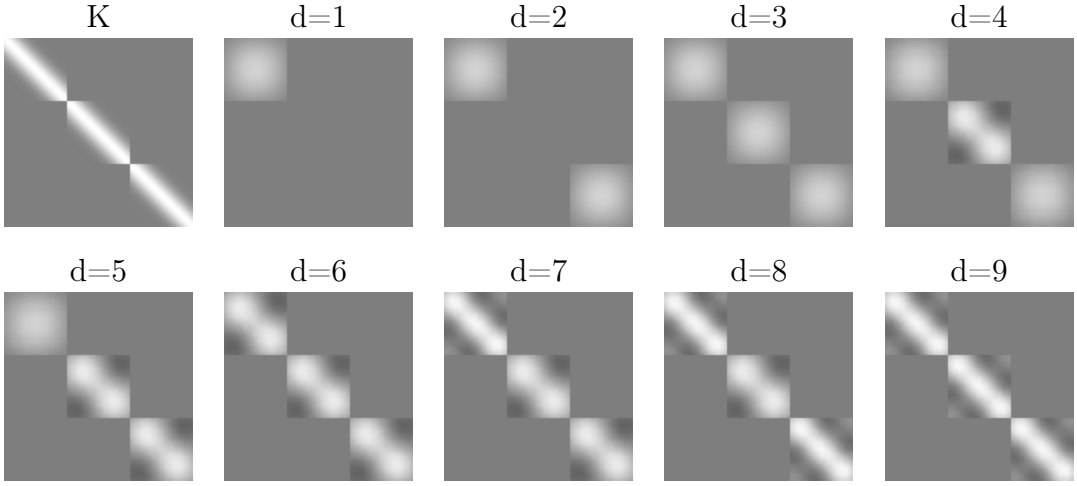


Figure 2.5.: Toy example showing how the kernel principal components relate locally and globally. On the top-left corner, we depict a block-diagonal Gram matrix representing a set of three disjoint one-dimensional data manifolds. We then display better and better approximations by adding more and more components. With every added dimension we add one more local eigenvector. Adding three dimensions to the model is equivalent to adding one dimension to each local model.

In such setting, the global Gram matrix takes the form of a block-diagonal matrix

$$K = \begin{pmatrix} K_1 & & \mathbb{O} \\ & \ddots & \\ \mathbb{O} & & K_N \end{pmatrix}$$

where $\{K_i\}_{i=1}^N$ are the local kernels. Let λ_i be the ith eigenvalue of matrix K and λ_{jk} be the jth eigenvalue of the matrix K_k. It can be shown easily, that the following identities hold:

1. The eigenvalues of the matrix K are the concatenation of the eigenvalues of matrices $K_1, \ldots, K_N$.

2. The eigenvalues of the matrix K preserve their associated eigenvector in the local matrix K_k. In other words, defining $u(\lambda)$ to be the eigenvector associated with the eigenvalue λ, we have that if $\lambda_i = \lambda_{jk}$, then $u(\lambda_i) = u(\lambda_{jk})$.

An intuition is given for such identities in Figure 2.5. We observe that considering N principal components of the global data distribution is the same as considering one component of each local data distribution. Overall relevant dimensionality is the sum of local relevant dimensionalities. Similarly global noise is the average of local noise levels. Therefore, a combination of noise or dimensionality reduction events taking place at various locations of the representation space will translate into similar reduction of noise or dimensionality for the global RBF kernel.

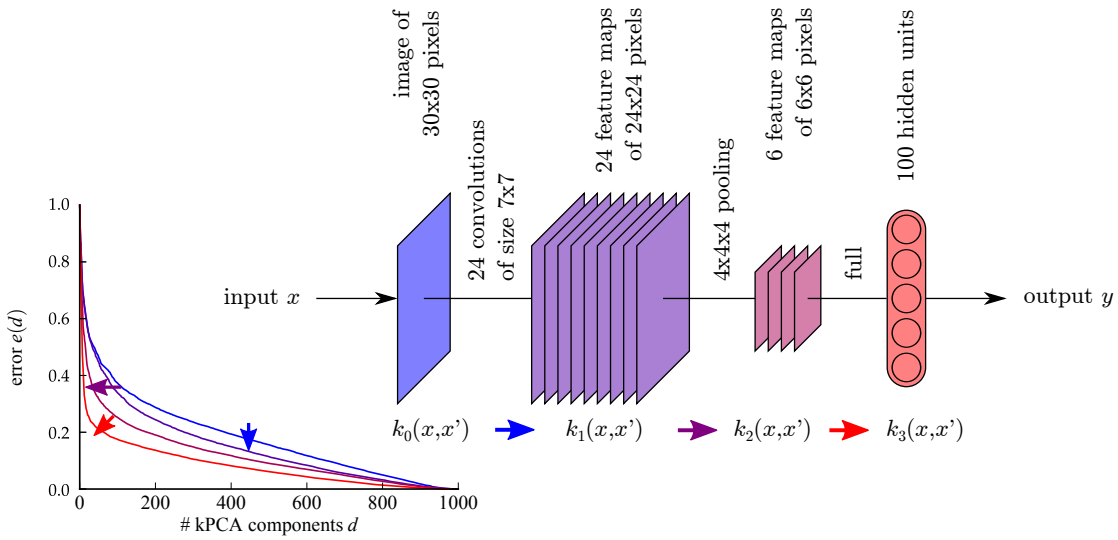


Figure 2.6.: Layer-wise analysis of a convolutional neural network. In this network, noise and dimensionality reduction are structurally separated. The convolution performs a noise reduction step, as shown by the blue arrow. The pooling mechanism performs a dimensionality reduction step, as shown by the purple arrow. The last fully connected layer performs both noise and dimensionality reduction, as shown by the red arrow.

A Statistical View

Finally, we give a statistical interpretation of noise and dimensionality reduction in deep networks from the perspective of an RBF kernel. Such interpretation is based on the fact that the empirical feature space of RBF kernels captures reliably only a few leading components (low frequencies) while perceiving higher frequencies as noise due to insufficient sampling rate or insufficiently small scale parameter σ.

We have seen in Section 2.3.2, that noise reduction essentially arises from the application of a nonlinearity to the input representation. Such nonlinearities have to be learned from many samples in order to justify statistically the induced extra complexity. Similarly, we have seen in Section 2.3.2, that dimensionality reduction is essentially a reweighting of the kernel principal components. Again, in order to amplify smaller components without introducing noise, this dimensionality reduction must be learned with high accuracy.

Figure 2.6 illustrates noise and dimensionality reduction in a convolutional neural network (LeCun et al. 1998) trained on the same noisy variant of MNIST as in Figure 2.3, where lower contrast irrelevant handwritten digits are linearly superposed in the pixel space. The network is composed of three layers: a convolution layer, a pooling layer and a fully connected layer whose output is fed to a logistic classifier.

Each layer of the network has a particular purpose: The convolution layer is

directly followed by a nonlinearity and is able to learn sharp nonlinear features that perform denoising. This is translated in our kernel analysis by a reduction of the noise baseline in $e(d)$, as shown by the blue arrow. The statistical robustness of the convolution layer, caused by features sharing across multiple locations in the pixel space, allows to learn a large number of convolutional filters. This increases the problem dimensionality but reduces the level of noise.

On the other hand, the pooling layer is linear, parameterless, and designed to aggregate features from similar locations and feature maps. Pooling is measured by our analysis as a reduction of dimensionality shown by the purple arrow. In particular, the statistical robustness of the pooling layer caused by the absence of learned parameters allows to perform a clean dimensionality reduction step, without introducing noise.

Finally, the last fully connected layer performs one more step of noise and dimensionality reduction where the representation is nonlinearly mapped onto a lower-dimensional and more discriminative subspace.

3. Backpropagation Networks

Perhaps the most basic supervised learning algorithm is the perceptron (Rosenblatt 1958). The perceptron is essentially a linear classifier coming with an efficient learning algorithm. A fundamental limitation of the perceptron is that it does not allow for representing nonlinear decision functions. Backpropagation networks (Rumelhart et al. 1986) extend the perceptron by adding several layers of nonlinear transformations that are learned by backpropagating errors. This seemingly simple modification of the original algorithm endows these multilayer networks with universal approximation capabilities (Cybenko 1989, Hornik 1991), allowing them to model, in practice, extremely complex tasks (e.g. Krizhevsky et al. 2012). An important feature of backpropagation networks is that they are able to learn their own internal representation from data. In this chapter, we aim to characterize these internal representations using the kernel analysis of Chapter 2.

A backpropagation network learns a feedforward nonlinear mapping between input and output. In a network of L layers with $N_1, \ldots, N_L$ units at each layer, the mapping can be defined by the sequence of nonlinear functions

$$\begin{aligned} x_0 &= x, \\ x_1 &= \sigma(W_1^\top (x_0 - \beta_0) + b_1), \\ &\vdots \\ x_L &= \sigma(W_L^\top (x_{L-1} - \beta_{L-1}) + b_L). \end{aligned}$$

An illustration of a backpropagation network is given in Figure 3.1. The variables $x_l \in \mathbb{R}^{N_l}$ denote the representations at each layer l. $\{W_l\}_{l=1}^L$ are the weight matrices of size $N_{l-1} \times N_l$, and $\{b_l\}_{l=1}^L$, $\{\beta_l\}_{l=0}^{L-1}$ are the bias and offset vectors of size N_l. Offset vectors are usually set to the mean activation of respective units ($\beta_l = \mathbb{E}[x_l]$) in order to improve the conditioning of the optimization problem (LeCun et al. 1998). The nonlinear function $\sigma(x)$ is applied element-wise to the representations at each layer. A common nonlinear function is the logistic sigmoid, defined as $\sigma(x) = \frac{e^x}{1+e^x}$. Another nonlinearity that is often used in practice is the rectifying function, defined as $\sigma(x) = \max(0, x)$ (Nair and Hinton 2010, Glorot et al. 2011).

We denote by $\theta = \{W_l, b_l\}_{l=1}^L$ the set of model parameters at each layer that has to be learned from data. For a given data point (x, t), we wish to minimize the error function

$$E = L(x_L, t) + \Omega(x, \theta)$$

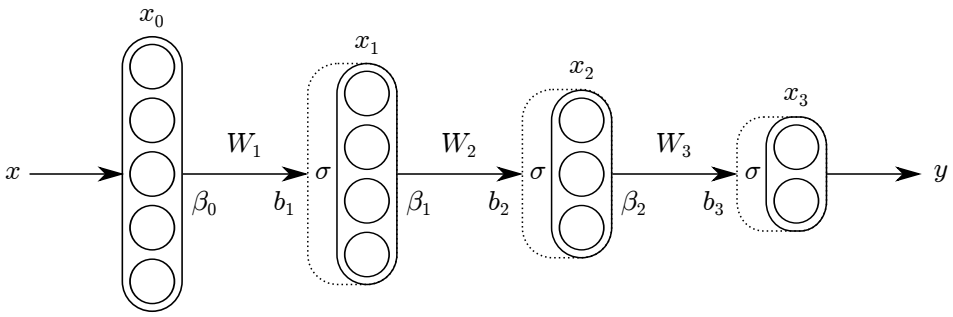


Figure 3.1.: Sketch of a backpropagation network with three hidden layers along with the weight, bias and offset parameters at each layer.

with respect to the parameter θ. The loss function L measures the mismatch between the top-layer representation x_L and the label t to be predicted, and is differentiable with respect to x_L. The regularization term $\Omega(x, \theta)$ penalizes parameters and representations that are too complex. Using gradient descent, the parameter update is computed at each iteration as

$$\theta \leftarrow \theta - \gamma \frac{\partial E}{\partial \theta},$$

where γ is the learning rate. Let $\psi(\xi)$ be the function mapping the representation at a given layer to its derivative with respect to the pre-activations at the same layer (before σ). For a sigmoid network, it is given by $\psi(\xi) = \xi \cdot (1 - \xi)$. For a rectifier network, it is given by $\psi(\xi) = 1_{\xi > 0}$. The gradient of E with respect to model parameters at each layer can be obtained by applying the chain rule (Hinton and Sejnowski 1986). It is backpropagated from one layer to another as:

$$\frac{\partial E}{\partial x_{l-1}} = \frac{\partial E}{\partial x_l} \cdot \frac{\partial x_l}{\partial x_{l-1}} = \left(\frac{\partial E}{\partial x_l} \circ \psi(x_l)^\top \right) \cdot W_l$$

where $\circ$ denotes the element-wise product. These gradients can be iteratively computed starting from $\frac{\partial E}{\partial x_L}$ down to $\frac{\partial E}{\partial x_1}$. As the error gradient is backpropagated, it is alternatively multiplied by weights matrices and by $\psi(\xi)$. Similarly, the gradient with respect to parameters at each layer can be computed easily as:

$$\frac{\partial E}{\partial W_l^\top} = \frac{\partial E}{\partial x_l} \cdot \frac{\partial x_l}{\partial W_l^\top} = x_{l-1} \cdot \left(\frac{\partial E}{\partial x_l} \circ \psi(x_l)^\top \right).$$

When training over a dataset $\mathcal{D}$, the objective becomes the average of the error for each sample in the dataset, and the gradient of the objective with respect to the model parameters simply becomes the average of all individual gradients. A major difficulty when training backpropagation networks arises from the nonconvexity of the objective E in the parameter space. Nonconvexity makes the model sensitive to the learning procedure, in particular, the initial solution θ_0, the type of optimizer, the parameterization of the network, and more generally, the amount of noise

injected in the learning procedure. In this chapter, we will analyze the effect of the learning procedure, in particular, the amount of learning noise, on the layer-wise organization of the learned solution.

3.1. Layer-Wise Simplification of Representations

We have briefly reviewed in Section 2.3 some evidence from circuit theory and neural recordings, that deep architectures are necessary in order to efficiently represent a certain class of learning problems. One might hypothesize that a backpropagation network that has been trained for long enough and with enough data will learn such deep representation of the problem. However, while such decomposition of task is usually understood on a conceptual level, based on our common understanding of how objects are constructed from basic features, it is not directly obvious *why* the backpropagation algorithm would learn such deep sequence of increasingly low-dimensional representations.

3.1.1. An Optimization Perspective

In this section, we argue that such layer-wise distribution of the representation is already implicitly "contained" in the backpropagation algorithm. In particular, the multiplicative nature of the mapping between input and output provides a mechanism of fairness in the allocation of modeling load at each layer. This is best illustrated by looking at simple prototypical deep networks.

Let us consider a simple narrow linear deep network composed of one unit at each layer.

$$x_L = w_L \cdot \ldots \cdot w_1 \cdot x = \Big(\prod_{i=1}^{L} w_i \Big) \cdot x \tag{3.1}$$

The gradient of error with respect to each parameter is given by

$$\frac{\partial E}{\partial w_l} = \frac{\partial E}{\partial x_L} \cdot w_L \cdot \ldots \cdot w_{l+1} \cdot w_{l-1} \cdot \ldots \cdot w_1 \cdot x = \frac{\partial E}{\partial x_L} \cdot \Big(\prod_{i=1}^{L} w_i \Big) \cdot \frac{x}{w_l} \tag{3.2}$$

The gradient is a simple product of weights at each layer except the weight for which the gradient is computed. This particular structure of the gradient allows for learning a set of weights that equally contribute to the function to be learned: The argument consists of supposing that a weight at a given layer l is smaller than weights at different layers. It is easy to see from Equation 3.2 that this weight has the largest corresponding gradient, because the weight gradient has its own small weight in the denominator. Thus, as small weights receive larger updates, assuming that some of the weights are initially too small, the magnitude of weights at each layer will tend to equalize over time. Figure 3.2 illustrates this effect in a simple narrow two-layer linear network. The error surface is shown as a function

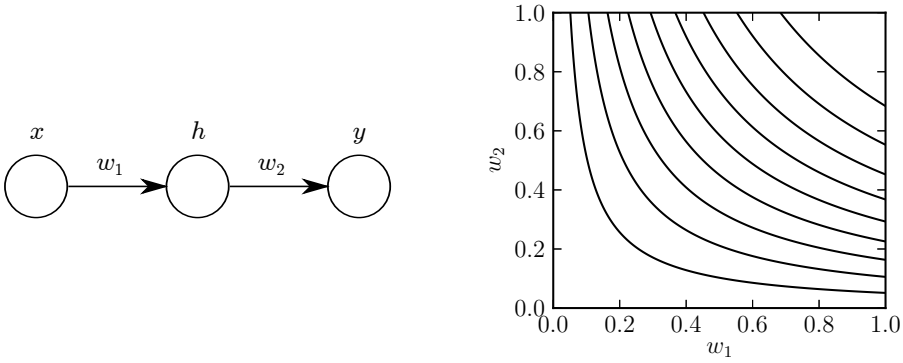


Figure 3.2.: Error surface for a two-parameter linear network $y = w_2 \cdot w_1 \cdot x$. The error in this example is the square error $E = (y - t)^2$ for $x, t = 1$. Optimal solutions are in the top right corner of the plot. We can observe that the lowest valued parameter tends to have the strongest update. This encourages a fair distribution of the modeling load at each layer.

of the first layer parameter w_1 and second layer parameter w_2. The optimum is located in the upper right corner. The error gradient always points to the middle ($w_1 = w_2$), showing the tendency to fair allocation of modeling load at each layer.

The argument also holds in the nonlinear case when a nonlinearity $\sigma(x)$ is applied at each layer. Here, for a specific input x, the nonlinear mapping can be locally approximated by a linear function. Here, an additional difficulty compared to the pure linear case discussed before is that the locally linear approximation now introduces multiplicative factors in both the forward pass and the backward pass. In the forward pass, the signal is multiplied at each layer by $\sigma(x_l)/x_l$ (the ratio between activation and pre-activation). In the backward pass, the error gradient is multiplied at each layer by $\sigma'(x_l)$ (the local derivative of the nonlinearity). Interestingly, if the activation vs. pre-activation ratio matches the slope of the nonlinearity, that is, if

$$\sigma(x_l)/x_l = \sigma'(x_l) \quad \text{for all } l,$$

multiplicative factors in the forward and backward pass cancel out. In practice, only linear functions and some piecewise linear functions satisfy this equation exactly. The rectifying function $\sigma(x) = \max(0, x)$ is one of them. The hyperbolic tangent function and other sigmoids centered at the origin also satisfy this equation up to some bounded error.

The argument can also be extended to higher-dimensional networks by defining the feed-forward pass as a chain of matrix products $x_L = W_L^\top \cdot \ldots \cdot W_1^\top \cdot x$. The gradient with respect to the weight matrix at layer l is

$$\frac{\partial E}{\partial W_l} = (W_{l+1} \cdot \ldots \cdot W_L \cdot \frac{\partial E}{\partial x_L^\top}) \cdot (x^\top \cdot W_1 \cdot \ldots \cdot W_{l-1}).$$

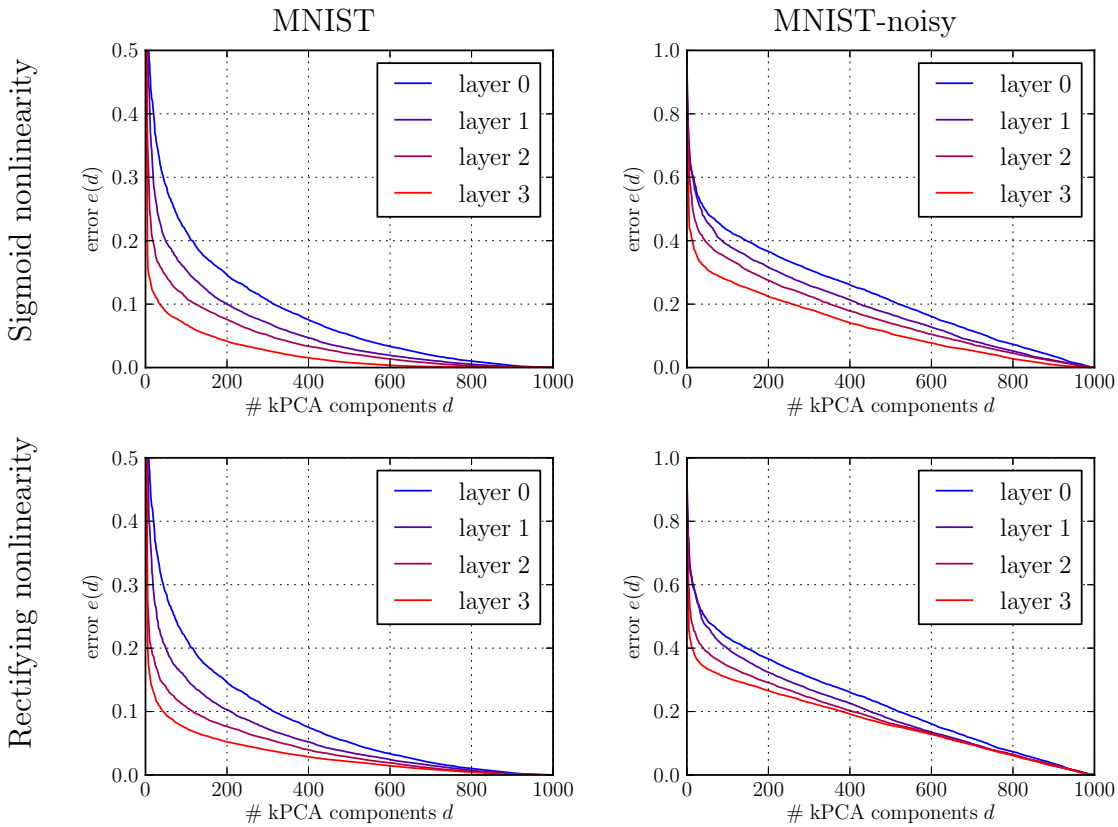


Figure 3.3.: Layer-wise evolution of the representation in a backpropagation network with three hidden layers. The network is trained on MNIST and a noisy variant of it, and with two types of nonlinearities. On the MNIST dataset, the input representation is high-dimensional but has low noise. The network implements a progressive layer-wise reduction of dimensionality. On the MNIST-noisy dataset, the input representation is low-dimensional with high-noise as we can see from the linear decay of $e(d)$. Here, in addition to dimensionality reduction, the network implements a progressive denoising of the representation.

Again, a highly contributing layer (a weight matrix with large weights) does not have itself entering its own gradient, but contributes to the gradient of all other layers. On the other hand, a weakly contributing layer receives in its gradient the contribution of all other layers, but inhibits the gradient in other layers by contributing with its small weights.

3.1.2. Experiments

In this section, we test empirically the proposed hypothesis on the fair distribution of modeling load at each layer, using the kernel analysis of Chapter 2. If such hypothesis holds, we expect to observe a progressive layer-wise reduction of noise and dimensionality at each layer. For this, we train several backpropagation

networks on the MNIST handwritten digits dataset described in Appendix A.1. Details of the training procedure are given in Appendix A.2. We apply the kernel analysis on a subset of the test set $\mathcal{B} \subset \mathcal{D}_{\text{test}}$ defined as

$$\mathcal{B} = \{x \in \mathcal{D}_{\text{test}} \text{ s.t. } ||x - c|| < \varepsilon\},$$

where c is chosen to be near the decision boundary, and ε is chosen such that $\mathcal{B}$ contains exactly 1000 samples. This preliminary step reduces the number of samples involved in the computation of kernel principal components, and lets the analysis concentrate on the region of the input space containing the discrimination boundary.

Figure 3.3 shows the layer-wise evolution of the representation for a simple backpropagation network trained on MNIST and on MNIST-noisy (a noisy version of the MNIST dataset where handwritten digits are linearly superposed in the pixel space to an unrelated handwritten digit with lower contrast). We perform the experiments both for a sigmoid network and a rectifier network. In all cases, we can observe a layer-wise reduction of the problem dimensionality. This is best seen by looking at the number of dimensions d required to achieve a certain error e, that recedes layer after layer.

While we cannot easily distinguish a noise baseline on the MNIST dataset (the problem is perceived as high-dimensional by the RBF kernel at each layer), such noise baseline is clearly visible on the MNIST-noisy dataset, where the error $e(d)$ starts to decay almost linearly after approximately 100 dimensions. On this last noisy dataset, we can observe that the noise baseline (i.e. the slope of the linear part of $e(d)$) decreases progressively as the input is propagated onto more layers of the network. This illustrates the capacity of deep networks to perform joint denoising and dimensionality reduction of the input representation.

Interestingly, on the MNIST-noisy dataset, the rectifier network does not perform much further denoising after the first layer. This may be due to the fact that the rectifying function does not saturate for positive values, and backpropagates more error signal in the network. This makes the learning process noisier than with sigmoids.

3.2. Learning Noise and Early Discrimination

In this section, we study how the noise in the learning algorithm affects the layer-wise organization of the representation. In particular, we will observe that noise causes an early discrimination effect, where most of the dimensionality reduction takes place in the first few layers. Then, we will argue that such early discrimination effect is the way by which the neural network spontaneously adapts to the high level of noise, in order to optimize for the training objective.

3.2.1. Three Sources of Noise

When learning a backpropagation network, we can distinguish at least three different sources of noise:

- *Stochastic gradient noise*: The first source of noise originates from the fact that most learning algorithms are training the network one example at a time (Bottou 1991; 2012). While such learning scheme is necessary in order to scale to large datasets, it also introduces noise in the parameter update, due to the fact that we do not descend the true gradient of the objective function, but a random approximation of it.

- *Parameter update noise*: Another source of noise arises from the parameter update rule where the local gradient (or local Hessian in the case of a second degree method) is extrapolated to a small neighborhood of the current parameter θ. Such extrapolation can become a major source of noise, in the event where the optimization problem is not well conditioned, a rather common scenario when training deep networks (Martens and Sutskever 2012).

- *Regularization noise*: A third source of noise comes from regularization. The regularization can be implicit, for example, by initializing the network with small weights and using early stopping. It can also be explicit, for example, by using weight penalties or by injecting noise into the model.

3.2.2. Understanding Early Discrimination

The Saturating Sigmoid

We consider a toy example consisting of a simple backpropagation network $y = v \cdot \tanh(w \cdot x)$, where w and v are the parameters. The dataset to learn is shown by black dots in Figure 3.4 (left) and admits an optimal solution at $w, v = 1$. The error as a function of the parameters (w, v) is shown in Figure 3.4 (right). We look at the stability of the optimal solution $(w, v = 1)$ shown in blue, and a near-optimal solution ($w = 2.7$ and $v = 0.97$) shown in red. In the first case, a slight variation of parameter w (more exactly, a decrease in value of w) can lead to a large error increase as shown by the narrowly spaced contour lines near the blue dot. On the other hand, in the second case where w is larger, the sigmoid reaches saturation and the error is not varying anymore as a function of w.

We would like to measure the effect of noise on the solution. We first note that in presence of learning noise, the optimization procedure typically converges to a stationary time-independent probability distribution $p(w, v)$. We consider the noise induced by a large learning rate, combined with some small random perturbation of the error gradient. Figure 3.5 shows the distribution $p(w, v)$ resulting from using a small or large learning rate. These distributions are respectively depicted in blue and red. Note that in each case, the learning rate for the parameter w is multiplied by 100, in order to account for the pathological curvature of the error

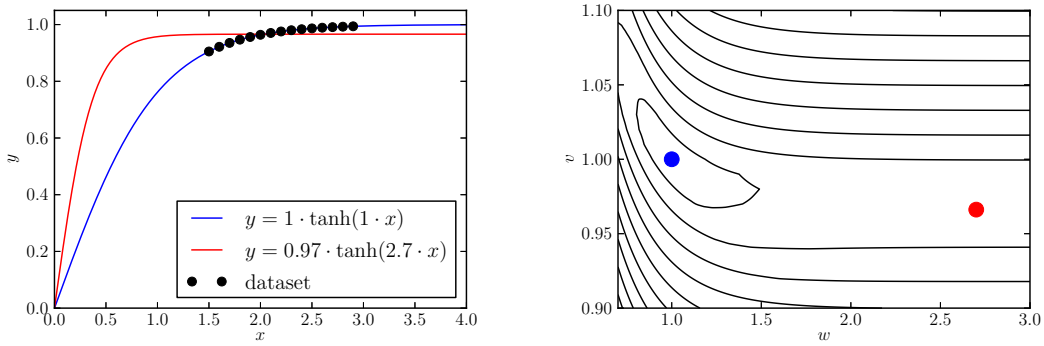


Figure 3.4.: The saturating sigmoid problem. *Left:* two different parameterized mappings $y = v \cdot \tanh(w \cdot x)$ between input and output, approximating the dataset (shown by black dots). *Right:* Error function in the parameter space (w, v).

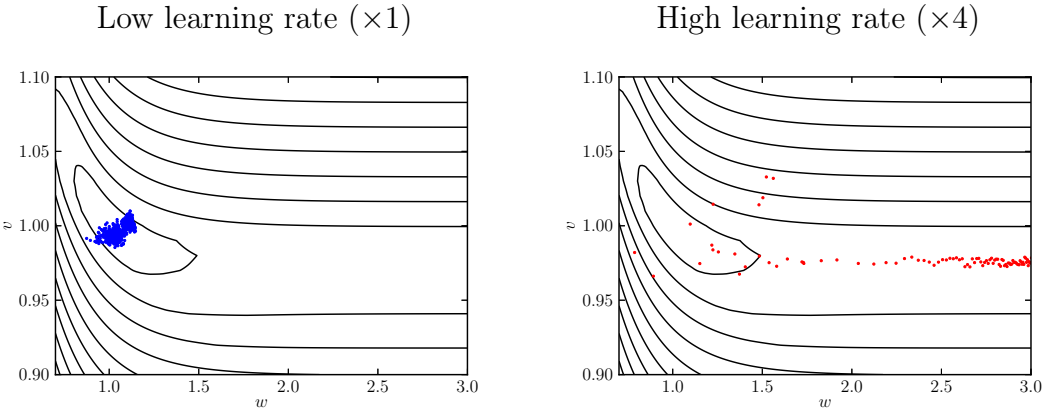


Figure 3.5.: Simulation of the gradient descent for different learning rates on the saturating sigmoid problem. Each point denotes a different iteration of the gradient descent. High learning rate tends to move the optimizer in the saturation mode of the sigmoid.

surface. For small learning rates, $p(w, v)$ has most of its mass near the optimum $w, v = 1$. For high learning rates, $p(w, v)$ spreads to the robust saturated regime of the sigmoid, while has low density near the optimum. An intuitive explanation for this effect is that the optimizer is frequently pushed away from the optimum, due to the strong neighboring gradients. Once it is in the saturated region, it takes a very long time for the parameter w to move back to the optimum due to the very small derivatives $\partial E/\partial w$ when w is large.

We now analyze the noise and dimensionality profile of each solution (blue and red) using our kernel-based analysis:

- When w is large (red solution), all data maps to $y = 0.97$. This is a consequence of the squashing effect of the sigmoid. Therefore, this point-wise distribution is contained in a single kernel principal component of $k_{\mathrm{RBF}}(y, y')$,

however, it comes at the price of raising the noise baseline.

- When w is small (blue solution), mapped data spans approximately $y \in [0.9, 1]$. Here, from the perspective of $k_{\text{RBF}}(y, y')$, the noise baseline remains zero, since we have captured the optimal first layer feature, but this comes at the price of higher dimensionality since it now requires several kernel principal components to model the range $[0.9, 1]$.

Therefore, in presence of learning noise, our analysis predicts that the first layer representation is low-dimensional, but introduces noise. From this point, it is difficult to further improve the representation in the following layers. On the other hand, in absence of learning noise, the mapped representation does not introduce noise, and a second layer of dimensionality reduction can be implemented more easily.

A Channel View

Another way of understanding the early discrimination effect in presence of learning noise is by viewing the neural network as a communication channel between input and output, and where noise is injected at each layer. Such setting is illustrated in Figure 3.6 (left). The dimensionality reduction nature of the problem we are studying implies that labels have much less entropy than the input ($H(T) \ll H(X)$). A solution to the problem of noise would be for the network to use only one layer ($y = f_1(x)$) and to use the remaining layers for redundantly encoding y (in a way that makes it robust to the noise).

The interpretation of such channel effect in the our kernel-based framework is illustrated in Figure 3.6 (middle). By learning a first layer representation consisting of only a very few number of label-related principal components, the bulk of remaining principal components can be filtered out, and the noise arising from the learning algorithm can be easily removed layer after layer using low-dimensional mappings. On the other hand, if a large number of kernel principal components are propagated in the hierarchy, these principal components will be incrementally corrupted by noise.

The early discriminating solution that we have argued to be more robust to noise, assumes that the low-dimensional problem representation can be extracted directly with only a few layers. However, when the problem requires depth in order to be correctly represented, we are facing a trade-off between approximation error E_{approx} caused by the restricted number of layers, and noise-induced error E_{noise} caused by the noise in the learning algorithm:

$$E(l) = E_{\text{approx}}(l) + E_{\text{noise}}(l).$$

The effective number of layers implementing the learned solution is denoted by l. Such trade-off is illustrated in Figure 3.6 (right). The approximation error $E_{\text{approx}}(l)$ decreases with l, because more layers increase the representational power of the network. On the other hand, the noise-induced error $E_{\text{noise}}(l)$ increases with

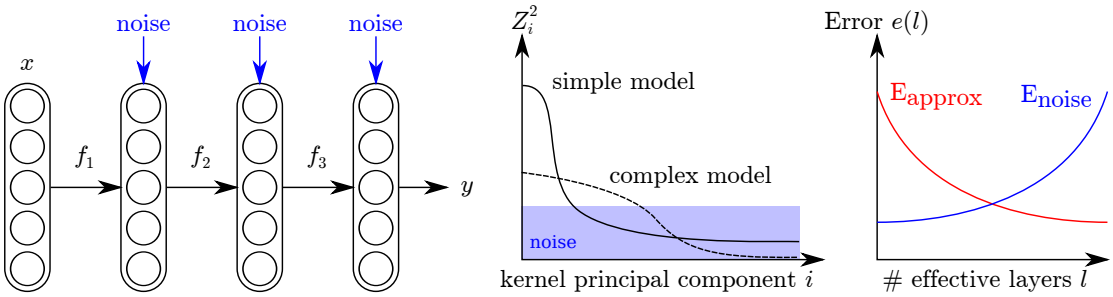


Figure 3.6.: Cartoon illustrating the view of a feed-forward network as a noisy channel. Left: Diagram of a feed-forward network with noise added at each layer. Middle: RDE interpretation of the noisy network. The simple low-dimensional representation is more robust to noise. Right: Tradeoff between approximation error and noise-induced error.

l, because the signal transmitted in the feed-forward pass is more easily corrupted by noise.

Overall, we can conjecture, that there will be an optimal number of layers that the learning algorithm will discover, in order to minimize the error. As the learning procedure becomes less noisy, the effective number of layers will increase. On the other hand, if learning remains noisy, the effective number of layers will stay low.

3.2.3. Experiments

We now train several backpropagation networks with different sources of noise. Details of the training procedure are given in Appendix A.2. As a baseline, we consider the backpropagation network of Section 3.1.1 trained on MNIST. We test the effect of the three types of noise described in Section 3.2.1, on the layer-wise evolution of the representation. The first network is trained using pure stochastic gradient descent (the error gradient is computed from only one data point). The second network has parameter update noise. Such noise is created by reparameterizing the network such that the optimization problem becomes badly conditioned. In this network, we add a constant offset "+3" to the sigmoids activations at each layer. For the third network, we inject Gaussian noise with variance 1 to the output derivatives in order simulate a regularizer.

Figure 3.7, shows the layer-wise evolution of the representation for these three noisy backpropagation networks. We can observe that under the effect of noise, the bulk of discrimination becomes concentrated in the first layer. In particular, noise causes the relevant problem dimensionality to shrink to only a few kernel principal components in each hidden layer. This observation corroborates our early discrimination hypothesis made earlier. This also suggests that these networks are essentially using the deeper layers as a robust transmission channel, without much further dimensionality reduction or denoising taking place.

There are at least two algorithms that are not subject to early discrimina-

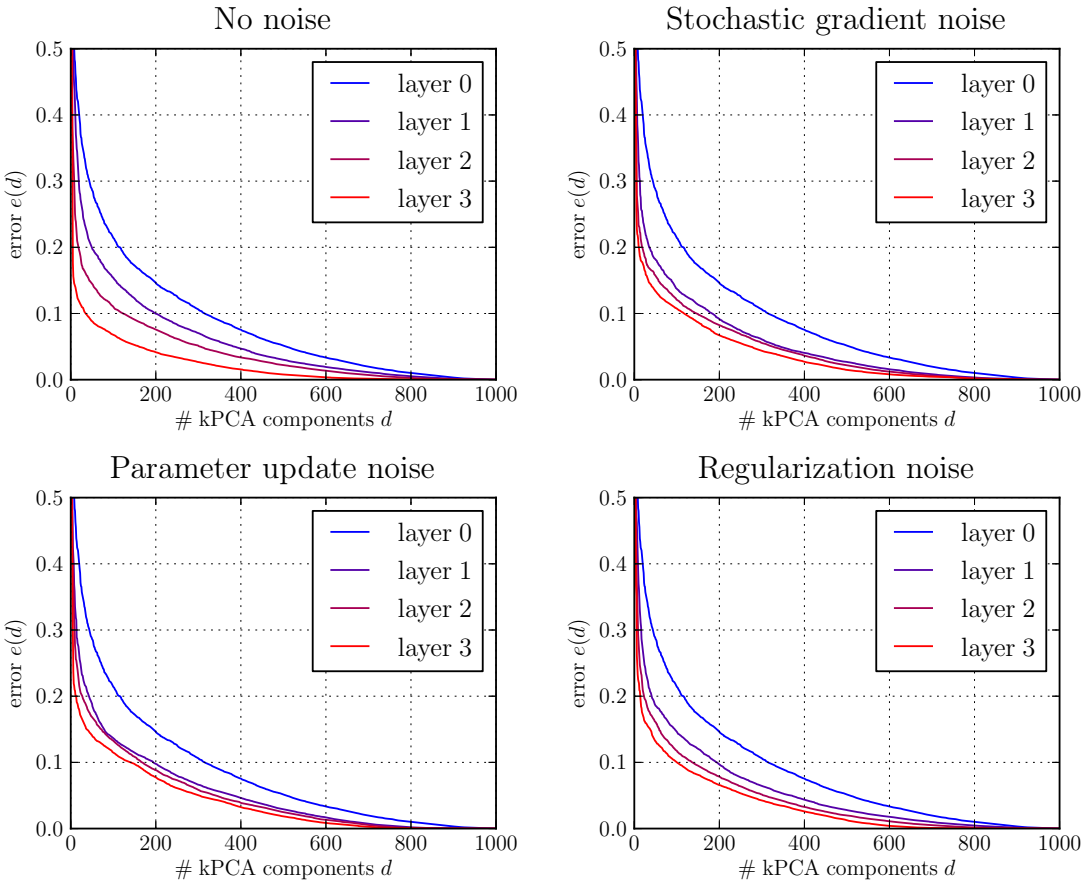


Figure 3.7.: Plots showing the effect of various sources of noise on the layer-wise evolution of the representation of a network trained on the MNIST dataset. All sources of noise consistently cause an early discrimination effect where the first layer carries most of the dimensionality and noise reduction.

tion: pretrained MLPs (PMLP, Hinton et al. 2006) and convolutional neural networks (CNN, LeCun 1989), both of them have led to significant performance improvements in practical applications: See, for example, Salakhutdinov and Hinton (2009), Rifai et al. (2011a), Dahl et al. (2011) for successful applications of PMLPs, and LeCun et al. (1998), Krizhevsky et al. (2012), Abdel-Hamid et al. (2012) for successful applications of CNNs.

Figure 3.8 (left) shows the layer-wise evolution of the representation for an MLP (simple backpropagation network), a PMLP and a CNN. The training procedure for these networks and details of the kernel analysis are described in Appendix A.2. We display the error $e(d)$ for $d = 10$ dimensions. Experiments show that the PMLP and the CNN are less subject to the early discrimination effect, as they discriminate considerably later than the MLP. For the PMLP, such absence of early discrimination can be explained as follows: The PMLP has already a well-structured feature hierarchy that only requires a small amount of supervised fine-

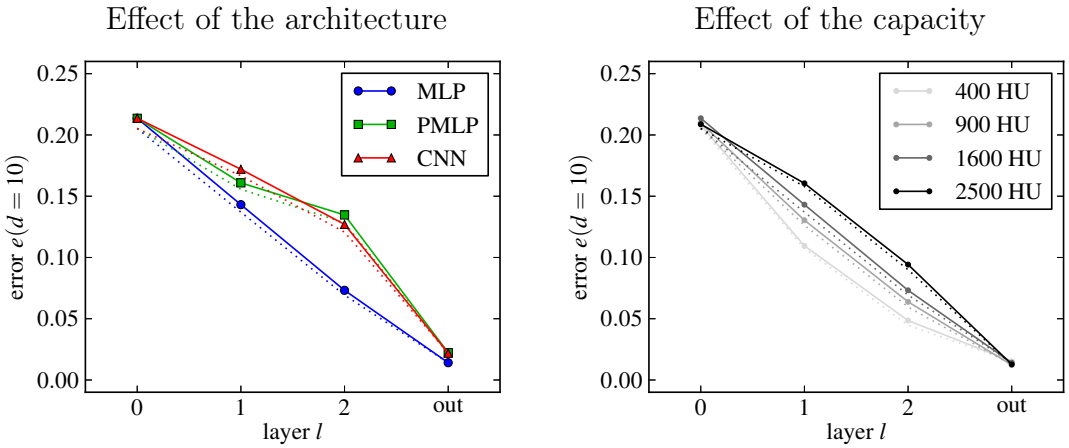


Figure 3.8.: Plots taken from the paper by Montavon et al. (2011) showing the effect of neural network architecture and capacity on the layer-wise evolution of the representation. Error is shown for a model based on the 10 leading kernel principal components.

tuning. As a consequence, (1) the learning noise that causes early discrimination in standard backpropagation will disappear very fast since the initial solution is close to the final one, and (2) the unsupervised model uses nonlinearities in a more saturated regime, and the error gradient is consequently strongly dampened as it flows backwards. On the other hand, the CNN, structurally prevents the early discrimination effect, by restricting first layers to low level convolutions and spatial pooling. Here, even a large learning noise will be unlikely to cause early discrimination since the early discriminating solutions are simply not part of the class of functions that the CNN can represent.

Figure 3.8 (right) shows the layer-wise evolution of the representation in several MLPs with different numbers of units at each layer. We can observe that more units tend to reduce the early discrimination effect. This can be explained as follows: In the asymptotic case where the number of units in the network is infinite, all possible solutions are already contained in the network and the training objective simply consists of extracting the correct solution among the many possible ones. Also, the learning noise will not force the top layers of the network to behave as a robust transmission channel, as the higher capacity better absorbs the learning noise.

3.2.4. Discussion

We have studied the early discrimination effect in backpropagation networks and demonstrated how pretrained architectures and convolutional neural networks are less affected by it. Unfortunately, pretraining and convolutional networks are not as broadly applicable as simple backpropagation networks. Convolutional neural networks are only applicable to sequential data, where the task exhibits some degree of spatial invariance. Unsupervised pretraining makes underlying

assumptions about the nature of the task to be solved (Ghahramani 2003), and can be counterproductive when these assumptions are not met.

Backpropagation networks are usually initialized randomly. This introduces a bias in favor of some randomly complex models and harms generalization. Ensemble methods deal with this random complexity by averaging model predictions at test time between multiple networks and allow for higher generalization (Naftaly et al. 1997). However the computational overhead created by having several networks is not always affordable.

Stochastic networks (Neal 1992, Bengio and Thibodeau-Laufer 2013) or stochastic learning algorithms (Bottou 1991) have sometimes been advocated as a way of reducing the initialization bias. They allow in principle to explore more exhaustively the space of models of similar complexity, by randomly jumping between these models under the effect of noise. However, our analysis suggests that noise in deep networks causes an early discrimination effect, where the multiple layers are not used effectively. In this case, the model complexity tends to spread horizontally rather than vertically.

Our kernel-based analysis provides a way to quantify such early discrimination bias. However, we leave the discovery of generic methods that explore the space of deep structures and that are immune to early discrimination as an open question.

4. Deep Boltzmann Machines

Deep Boltzmann machines (DBM, Salakhutdinov and Hinton 2009) are deep undirected generative models of data, that can be used to learn multiple layers of representation. Deep Boltzmann machines have been proposed as an alternative to other deep unsupervised learning algorithms such as deep belief networks (Hinton et al. 2006, Hinton and Salakhutdinov 2006) or stacked autoencoders (Bengio et al. 2007, Vincent et al. 2010, Rifai et al. 2011b, Bengio et al. 2013b). A particular feature of deep Boltzmann machines is that they integrate top-down feedback at learning and prediction time, allowing them in practice to produce better generalizing models (Salakhutdinov and Hinton 2009, Srivastava and Salakhutdinov 2012).

In this chapter, we aim to characterize the representations that are learned in DBMs. Here, we will narrow our analysis to centered DBMs (Montavon and Müller 2012), where each unit of the network has mean activation 0. Centered DBMs confer two important advantages compared to the original non-centered version: First, they make the underlying optimization easier, which facilitates learning of all layers at the same time. This allows us to analyze representations at each layer, without the perturbing effect of layer-wise pretraining. Second, the learned model can be interpreted in terms of *modes of interaction* that we further develop in Section 4.1.

We consider a centered DBM with three layers x, y and z. Each layer has N_x, N_y and N_z units. The first layer x represents the visible units (the data) and the other layers represent the hidden units. Units at each layer are binary, so that each configuration (x, y, z) of the DBM is in the space $\{0,1\}^{N_x+N_y+N_z}$. A three-layer DBM is depicted in Figure 4.1. The centered DBM associates to each state (x, y, z) an energy defined as

$$\begin{aligned} E(x,y,z) = &- (x-\alpha)^\top W (y-\beta) \\ &- (y-\beta)^\top V (z-\gamma) \\ &- (x-\alpha)^\top a - (y-\beta)^\top b - (z-\gamma)^\top c. \end{aligned}$$

The parameters of the model are the weight matrices W of size $N_x \times N_y$ and V of size $N_y \times N_z$, the bias vectors a, b, c and offset vectors α, β, γ of same size as their respective layer. The offset vectors are set to correspond to the expectation of respective units ($\alpha = \mathbb{E}[x]$, $\beta = \mathbb{E}[y]$, $\gamma = \mathbb{E}[z]$). A probability distribution is associated to each possible configuration of the network as

$$p(x,y,z) = \frac{1}{Z(\theta)} e^{-E(x,y,z)}.$$

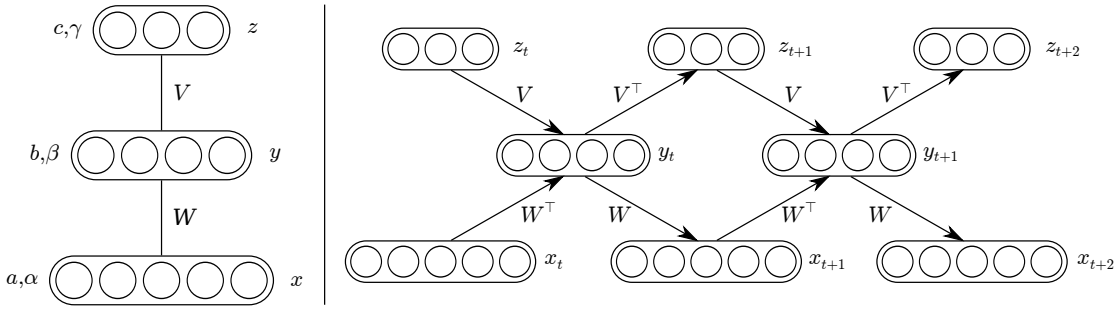


Figure 4.1.: *Left:* Sketch of a deep Boltzmann machine with two hidden layers along with its parameters. *Right:* Unfolded view of the same DBM defining a possible Gibbs sampling strategy.

The probability distribution can be marginalized over its hidden units in order to obtain an input distribution: $p(x) = \sum_{y,z} p(x, y, z)$. The constant $Z(\theta)$ is the partition function and normalizes the probability distribution to one. In practice, it is impossible to compute the partition function exactly as it would require to compute the energy of an exponentially large number of states. For this reason, sampling algorithms are often used (Hinton 2002, Tieleman 2008, Salakhutdinov and Murray 2008).

Binomial Boltzmann Machines

For analysis purposes, one can simulate the collective behavior of n binary units with a single binomial unit behaving according to a binomial distribution of parameters n and p (Teh and Hinton 2001). A binomial Boltzmann machine (BBM) has the probability distribution

$$p(x) = \frac{1}{Z(\theta)} \left[\prod_{i=1}^{N} \binom{n}{x_i} \right] e^{\frac{1}{2}(x-\beta)^\top W(x-\beta)+(x-\beta)^\top b}. \tag{4.1}$$

where N is the number of binomial units and $x \in \{0, \ldots, n\}^N$. The motivation for using binomial Boltzmann machines instead of binary ones is that it combinatorially reduces the size of the state space (from $2^{N \cdot n}$ to $(n+1)^N$). In our analysis, we use a maximum of 4 binomial units with $n = 25$. This amounts to enumerating $(25+1)^4 = 456976$ states. If the distribution were represented by a binary Boltzmann machine, the number of states would become $2^{4 \cdot 25} \approx 10^{30}$, which is clearly intractable.

4.1. Quantifying Layer-Wise Interactions

Deep Boltzmann machines are undirected probabilistic models, where the multiple layers interact in a non-benign fashion. In order to study emerging representations in these models, it is important to develop an understanding of the interaction

between the multiple layers. In this section, we propose an interpretation of these models in terms of modes of interaction. These modes of interaction originate from the careful analysis of simple binomial Boltzmann machines of two units, and will then be quantified by the *layer interaction number* (LIN), that we use in our empirical evaluations. We distinguish two modes of interaction:

Preserve Mode

The preserve mode between two binomial units is illustrated in Figure 4.2 (middle) and is characterized by a joint probability distribution that is strongly bimodal. The preserve mode is reached by setting a positive weight w between the two interacting units and by having biases of same sign. More generally, it is reached when $a \cdot b \cdot w > 0$. Here, the two biases a and b control how much of the probability mass is allocated to each cluster. The preserve mode allows for modeling multimodal, locally Gaussian distributions. Multimodality is important in order to implement dense regions of the input space representing class manifolds separated by large low density regions. For example, in handwritten digit recognition, low density regions between class manifolds arise from the impossibility of continuously transforming a handwritten digit into another digit, for example, continuously transforming the digit "0" into "1".

Complement Mode

The complement mode between two binomial units is illustrated in Figure 4.2 (right) and is characterized by a unimodal non-Gaussian joint distribution. Olshausen and Field (1996) showed that in the context of modeling natural images, Gaussianity is a very poor modeling assumption. This extends naturally to a wide range of signals exhibiting similar statistics such as handwritten characters and spectrograms. The complement mode is reached by setting a negative weight between units with similar biases. More generally, it is reached when $a \cdot b \cdot w < 0$. Here, the nonlinear elongated shape of the joint probability distribution arises from the conflicting requirement between the weight parameter (forcing the two units to take different values) and the biases (forcing the two units to take the same values), thus creating a manifold of different configurations that are all equally plausible. This can also be understood as a result of the coincidental superposition of the functions $\mathbb{E}[y|x]$ and $\mathbb{E}[x|y]$ (showed by solid and dashed lines).

Using Modes of Interaction to Model Physical Translations

In order to understand the importance of the complement and preserve modes of interaction for modeling real data, we consider the toy scenario of Figure 4.3. This simple example consists of a one-dimensional physical space with sensors x_1 and x_2 placed at different locations. Each sensor detects whether an object is in front of it.

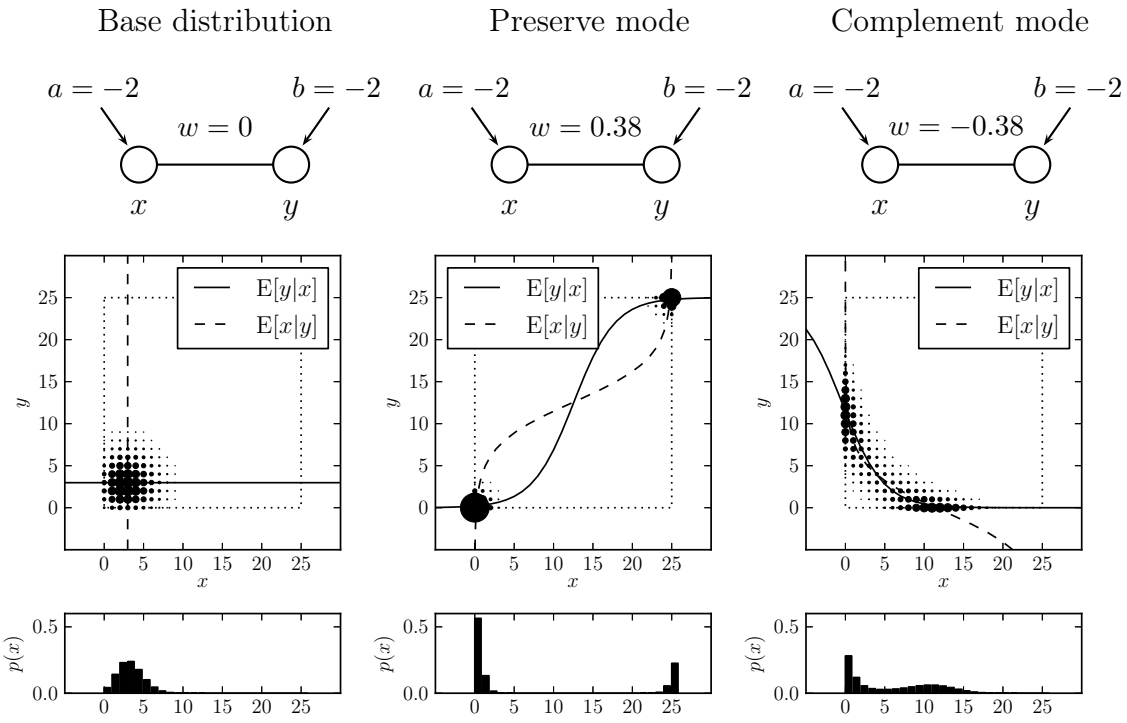


Figure 4.2.: Illustration of the modes of pairwise interaction in a binomial Boltzmann machine. *Top:* Diagram of the binomial Boltzmann machine with associated parameters. *Middle:* Joint probability distribution $p(x, y)$ as computed from Equation 4.1 and represented as a cloud of black dots. Solid and dashed lines represent a sigmoidal approximation of conditional expectations $\mathbb{E}[y|x]$ and $\mathbb{E}[x|y]$. *Bottom:* Marginal probability distribution $p(x)$.

Such setting is very common in practical applications, for example, for pixel images, where class manifolds usually follow directions of physical translation. This is more generally the case in any network of sensors that are physically limited in scope and placed at different locations.

We move a large object from left to right (or from right to left) so that sensors x_1 and x_2 are alternatively activated and deactivated. The translation of the object is linear in the physical space, but describes a nonlinear path in the sensor space (x_1, x_2). More precisely, physical translations are describing a loop in the sensor space similar to the one observed in Figure 4.3 (right).

In this example, units are dynamically set in the preserve and complement modes of interaction depending on the state of other units. In particular, when looking at the joint distribution $p(x_1, y_1)$, we can distinguish the four following cases:

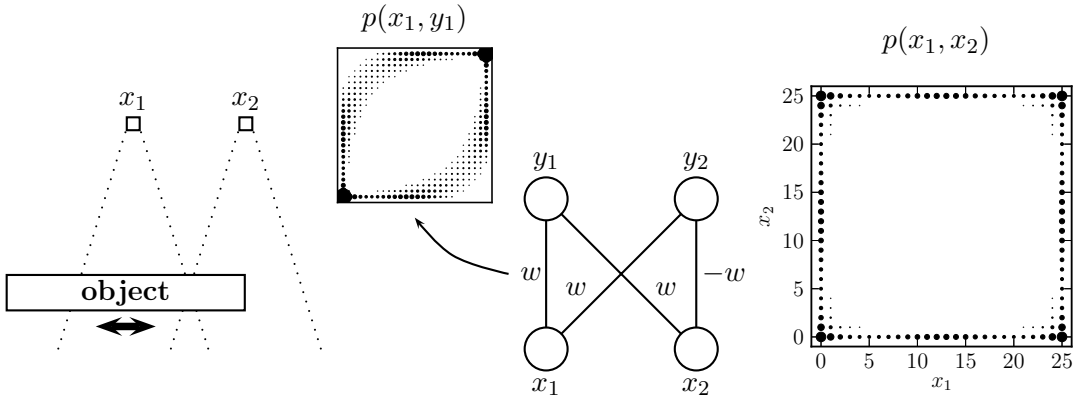


Figure 4.3.: Toy example showing how the two modes of interaction can be combined to implement physical translation in the sensor space. *Left:* Physical object being translated, and whose presence is detected by sensors x_1 and x_2. *Middle:* Binomial Boltzmann machine implementation of translation with $w = 0.35$. Pairs of connected units are set dynamically in each mode of interaction depending on the state of other units, as shown by the joint distribution $p(x_1, y_1)$. *Right:* Joint probability distribution over x_1 and x_2 modeling the physical translation (a loop in the joint space (x_1, x_2)).

x_2	y_2	interaction between x_1 and y_1	
off	off	preserve mode	bottom left of $p(x_1, y_1)$
off	on	complement mode	bottom right of $p(x_1, y_1)$
on	off	complement mode	top left of $p(x_1, y_1)$
on	on	preserve mode	top right of $p(x_1, y_1)$

4.1.1. The Layer Interaction Number (LIN)

For the purpose of our analysis, we would like to measure whether the trained DBM is globally in the preserve or complement mode of interaction. In a fully connected Boltzmann machine with energy function $E(x) = \frac{1}{2}(x-\beta)^\top W(x-\beta) + (x-\beta)^\top b$, assuming that all units are centered (i.e. $\mathbb{E}[x-\beta] = 0$), we define the interaction number (or IN) as:

$$\text{IN} = \frac{1}{2}\sum_{ij} b_i W_{ij} b_j = \frac{1}{2} b^\top W b. \tag{4.2}$$

We emphasize that the IN only applies to centered Boltzmann machines. Figure 4.4 shows the IN and the three-dimensional probability distribution, for various parameters of a three-unit fully connected binomial Boltzmann machine. A negative IN produces distributions that are unimodal and strongly non-Gaussian. On the other hand, a positive IN produces bimodal distributions made of two Gaussian-like clusters.

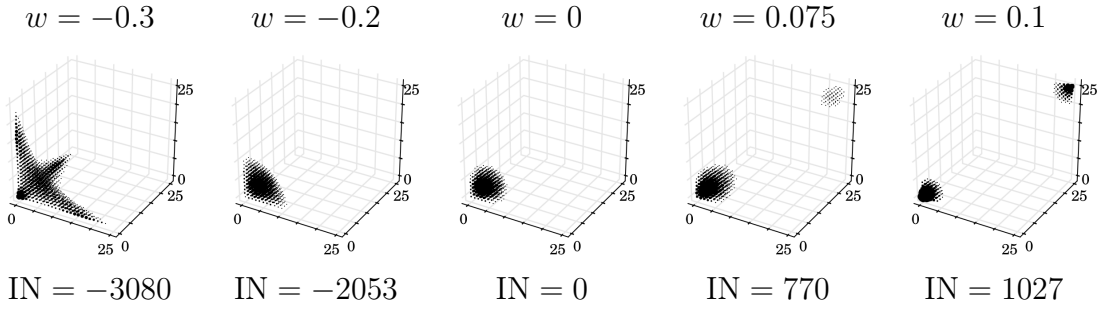


Figure 4.4.: Illustration of the type of distributions produced for different values of the interaction number IN in a fully connected Boltzmann machine composed of three binomial units. We set the same weight w between each pair of units, and the same bias $b = -2.25$ on each unit.

It is useful to discuss the limitations of the IN in the physical translation example of Figure 4.3. In this example biases were set to 0, therefore, IN $= \frac{1}{2}0^\top W 0 = 0$. Here, the IN does not represent the fact that both modes of interaction (preserve and complement) are used in the model and are determined dynamically depending on the state of other units. Instead, the IN is only a statistical measure that tells whether one of these modes is dominating over the other.

In practice, when analyzing deep Boltzmann machine or related models, it is more instructive to look at specific layers. Let x and y be two adjacent layers in a DBM with weight matrix W, bias vectors a, b and offsets vectors α, β. Assuming that units at each layer are centered ($\mathbb{E}[x - \alpha] = 0$ and $\mathbb{E}[y - \beta] = 0$), the layer interaction number (or LIN) is measured as:

$$\text{LIN} = \sum_{ij} a_i W_{ij} b_j = a^\top W b. \tag{4.3}$$

When LIN > 0, the two layers are globally in the preserve mode. When LIN < 0, the two layers are globally in the complement mode. The LIN is easy to monitor at training time, as it does not involve the enumeration of the exponentially large state space. The global IN of the DBM is further related to the LIN at each layer by

$$\text{IN} = \sum_{l=0}^{L-1} \text{LIN}(l, l+1),$$

where $\text{LIN}(l, l+1)$ measures the interaction between layers l and $l+1$.

Figure 4.5 shows the LIN at each layer of a DBM trained on various datasets. The training procedure is described in Appendix A.2. The LIN is plotted at multiple iterations of training. We observe that the LIN in the first layer tends to be negative initially, which suggests that the complement mode of interaction is the most appropriate, in order to quickly cover the whole data distribution. As training proceeds, the LIN raises to positive values, supposedly because of the need

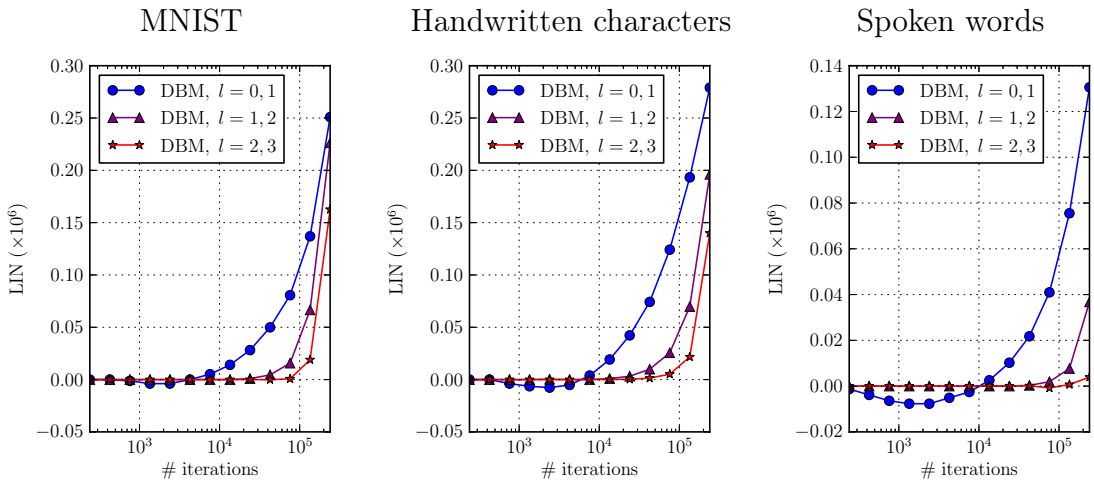


Figure 4.5.: Evolution of the layer interaction number (LIN) at each layer of a DBM trained on various datasets. A negative LIN indicates that layers are globally in the complement mode of interaction. A positive LIN indicates that layers are globally in the preserve mode.

to carve low density regions between class manifolds. Interestingly, higher layers, despite their smaller size, contribute significantly to the global IN, suggesting that they play an important role for carving these low-density regions. The role of each layer in a DBM will be further discussed in Section 4.2.

4.2. The Emergence of Invariance in DBMs

Montavon and Müller (2012) have shown on the MNIST dataset, that deep Boltzmann machines produce low-dimensional representations of task in the top layers. In this section, we explain why and under which conditions, these low-dimensional representations are obtained. We first look at a simple prototypical DBM made of a few binomial units, and relate the input distribution to the corresponding representation in top layers. Although small prototypical models cannot be expected to be fully representative of the complexity of real-world distributions, they are able to retain certain key features such as non-Gaussianity or manifold structure.

Our approach is in contrast to the more classical practice of starting with an existing data distribution and learning good model parameters. It has two main advantages: First, we do not need a learning algorithm that maps the input distribution $p(x)$ to the model parameters θ. Therefore, the bias caused by a potentially faulty learning algorithm is eliminated. Second, the number of units can be kept small, since the input distribution must only meet a loose specification: In order for a problem to be of class-manifold nature, there must be regions of low density between the multiple manifolds, and the manifolds must not be trivially related (i.e. not take the form of the same probability distribution replicated at different

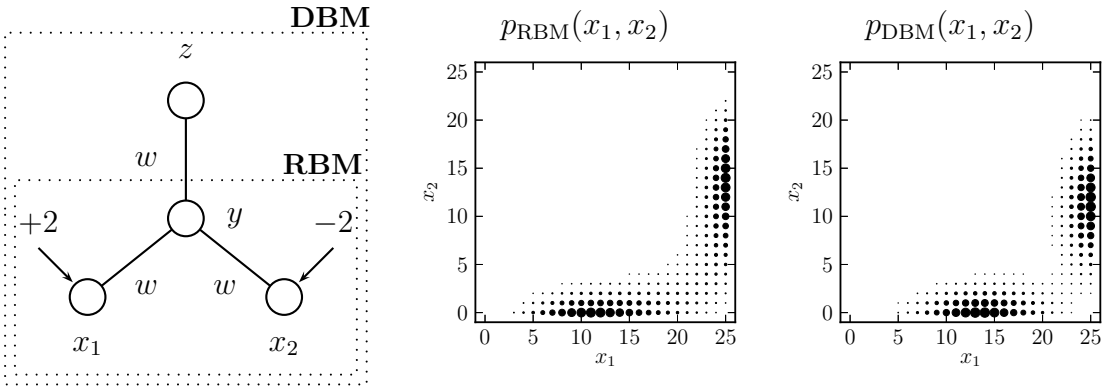


Figure 4.6.: RBM and DBM variant of a Boltzmann machine implementing two distinct manifolds in the input space. Input distributions for both models are shown on the right. Note that the weights of the RBM are scaled upwards (from $w = 0.16$ to $w = 0.20$) in order to account for the lack of top-down feedback.

locations).

Figure 4.6 shows a possible four-unit DBM that meets these requirements. As it can be seen from the joint distribution $p(x_1, x_2)$, the two manifolds are clearly separated by a low density region, while at the same time not trivially related. The rationale for choosing this DBM is the following: (1) Use a proxy variable y that dynamically determines which pair of units (x_1, y) or (x_2, y) is in the complement mode and preserve mode. This creates a banana-shaped distribution in the input space. (2) Add a top-layer unit z, that forces y to strongly associate to either x_1 or x_2, thus creating a region of low density region in the bottom-right corner of the banana-shaped distribution. This effectively creates two data manifolds with different statistical properties.

As a comparison, we show a similar input distribution implemented by a simple RBM. In the RBM, the weights between units must be slightly increased in order to overcome the lack of top-down feedback. In the RBM, we can observe that the distribution comes closer to the points $(0, 0)$ and $(25, 25)$ in the input space. Yet, it is still insufficient in order to separate the two manifolds.

4.2.1. From Better Distributions to Invariant Representations

We have shown how a DBM can produce a distribution composed of several non-trivially related manifolds. In order to obtain this special distribution, we used an additional top-layer unit z to force unit y to associate either with x_1 or x_2, and carve a region of low data density. Not only it allows to implement the desired input distribution, the top-layer unit z also becomes a very abstract quantity that behaves essentially as a manifold indicator. Figure 4.7 shows the joint distribution in the space formed by the input and the representation at each layer. The joint

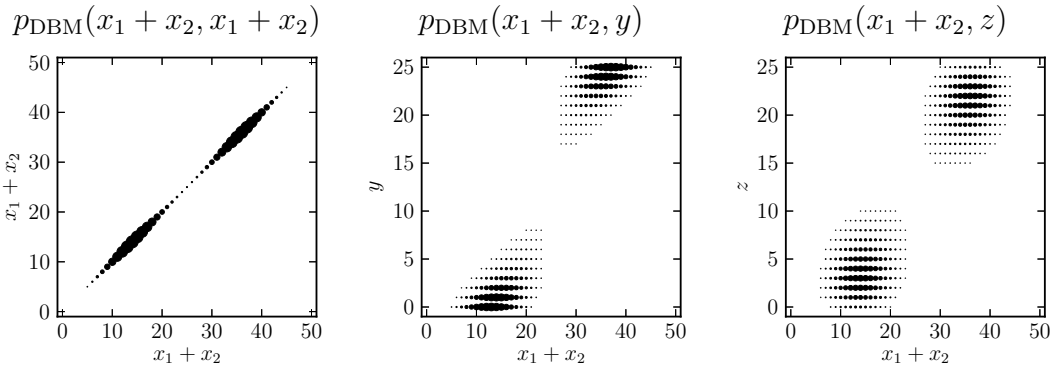


Figure 4.7.: Joint probability distribution of the input (represented as $x_1 + x_2$) with representations at each layer ($x_1 + x_2$, y and z) of the DBM of Figure 4.6.

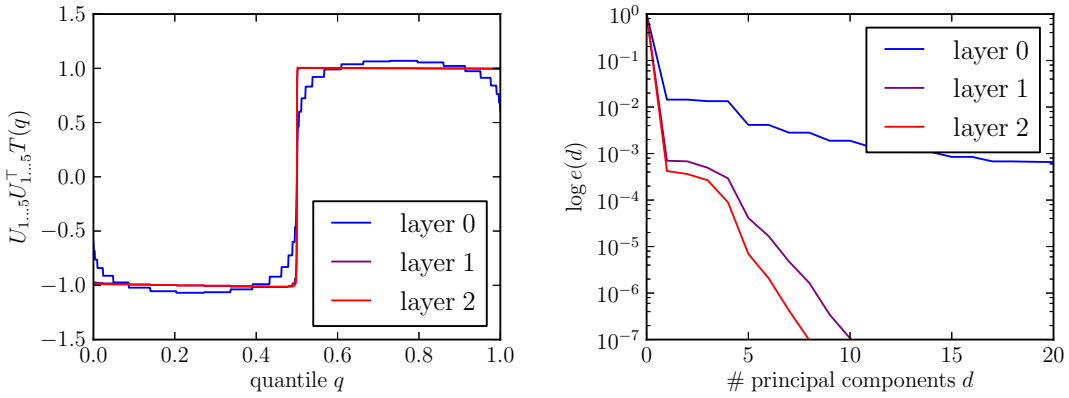


Figure 4.8.: Analysis of the DBM of Figure 4.6. Left: kernel PCA approximation of labels obtained from the five leading components at each layer. Right: kernel analysis at each layer of the DBM, showing the layer-wise reduction of dimensionality.

distribution features two modes, that are becoming increasingly Gaussian-shaped with every layer. In other words, the representation becomes layer after layer invariant to the position within each data manifold.

The emergence of invariance can be quantified by applying the kernel-based analysis of Chapter 2 to the representations formed at each layer. We apply the analysis to the toy DBM of Figure 4.6. We draw 1000 samples from the input distribution $p(x_1, x_2)$ and build the representation at each layer using a mean-field procedure. We assign to each sample the label "$t = -1$" if it belongs to the first manifold ($x_1 + x_2 < 25$) and "$t = 1$" if it belongs to the second manifold ($x_1 + x_2 > 25$). The analysis is shown in Figure 4.8: On the left, we plot the projection of the class manifold indicator T onto the five leading kernel principal components of the representation at each layer. We can observe that the transition

near the 0.5 quantile[1] of the data distribution is very sharp in the top layer. On the right, the residual error (see Equation 2.3) is plotted as a function of the number of kernel principal components. We can observe that the noise and dimensionality are gradually reduced. This demonstrates the tendency of deep Boltzmann machines to produce layer after layer simple representations of class-manifold problems.

4.2.2. Experiments

In order to test empirically the hypotheses made above about the emergence of invariance in DBMs, we train large deep Boltzmann machines of size 1600–800–400–200 and stacked RBMs of size 1600–800–400–200–100 on various datasets. We consider the following models: a DBM with top-down feedback, a DBM traversed with a feed-forward pass, a stack of RBMs trained with persistent contrastive divergence (PCD, Tieleman 2008), and a stack of RBMs trained with simple contrastive divergence (CD-1, Hinton 2002). Details of the training procedure are given in Appendix A.2. Then, we analyze the representations learned at each layer with the kernel analysis of Chapter 2. The kernel analysis uses 1000 samples drawn randomly from the test set and is repeated 5 times for different subsets of data, in order to produce error bars.

Figure 4.9 shows the result of the kernel analysis. Representations at each layer are collected by propagating the input in a feed-forward pass, or by using top-down feedback. The analysis reveals that, as we move from the input to the top layers, an increasingly small number of kernel principal components is necessary in order to model the problem with a certain accuracy. This validates our hypothesis stated in Section 4.2, that DBMs are creating low-dimensional top-layer representations of the problem. Remarkably, the DBM is doing so while keeping the noise level low. In stacked RBMs, the layer-wise evolution of the representation is typically slower in terms of number of layers than in a DBM. In Section 4.3, we will argue that stacked RBMs are plagued by a layer-wise oscillation effect that prevents the quick emergence of abstract top-layer representations.

Figure 4.10 (top) compares side-by-side the layer-wise evolution of the representation for each model and for various datasets. We observe that the DBMs are building in most cases problem representations that are simpler than the stacks of RBMs. On the spoken words dataset, the feed-forward pass produces a top-layer representation that is similar to the one obtained with top-down feedback. However, on the more complex MNIST and handwritten character datasets, top-down feedback still allows for more abstract top-layer representations.

Figure 4.10 (bottom) repeats the analysis for a transfer task consisting of representing classes that have not been observed at training time. On the MNIST dataset, the transfer task consists of representing handwritten digits that have been flipped horizontally. On the handwritten characters dataset, the transfer task consists of representing handwritten digits (0–9) in a model that has only been trained on alphabetic characters (A–Z). On the spoken words dataset, the

[1] Quantiles are computed from the one-dimensional distribution $x_1 + x_2$.

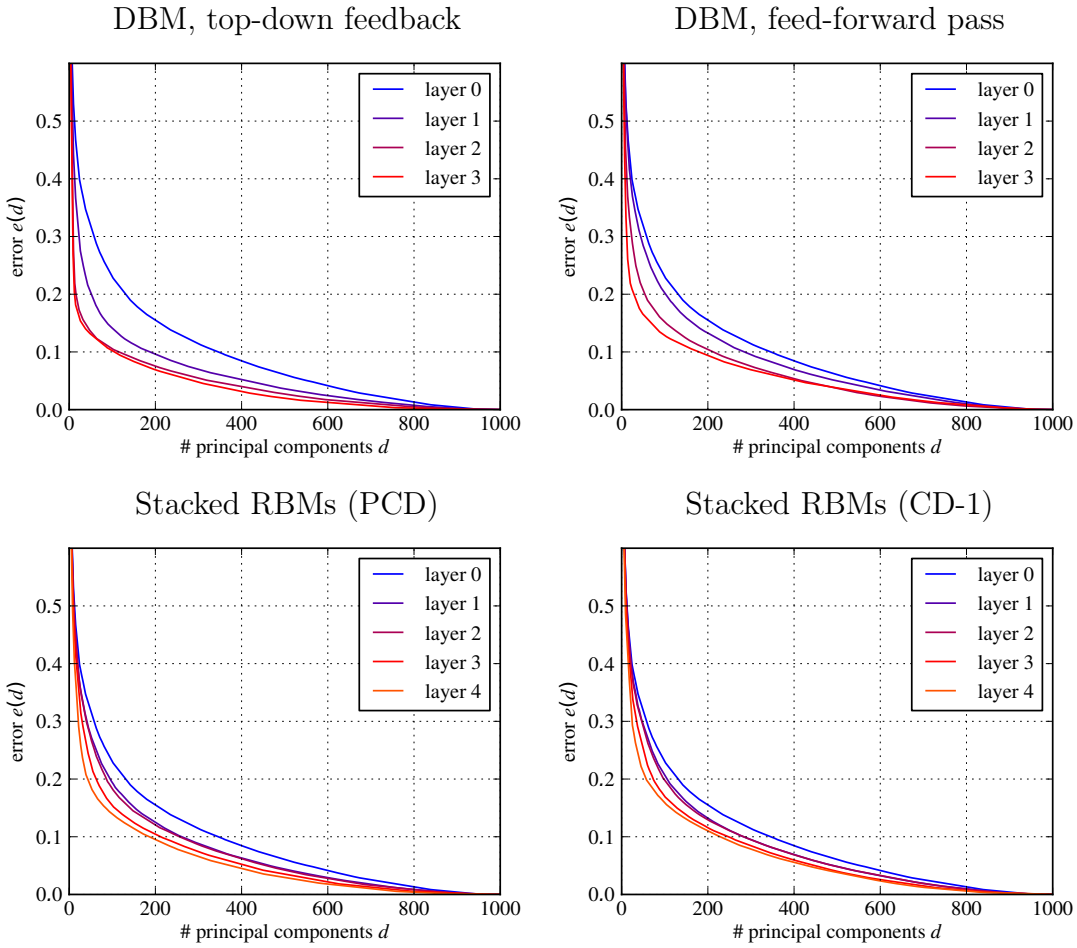


Figure 4.9.: Plots showing the kernel analysis of several models trained on the MNIST dataset. Each model produces increasingly low-dimensional and low-noise representations of the task. Among these models, the DBM with top-down feedback achieves the largest dimensionality reduction.

task consists of generalizing to new words that have not been observed at training time. Interestingly, all models are able to transfer features to a certain extent to the related task. While DBMs with top-down feedback seem to overfit on the original task and to produce a poor top-layer representation of the transfer task, the feed-forward DBM seems to benefit from the lower amount of processing in order to retain a certain level of reusability.

Figure 4.11 displays the filters learned by the DBM at each layer by linearly back-projecting units onto the input space. We observe that filters are increasingly smooth, starting with sharp edge detectors in the first layer and finishing with global class-related features in top layers.

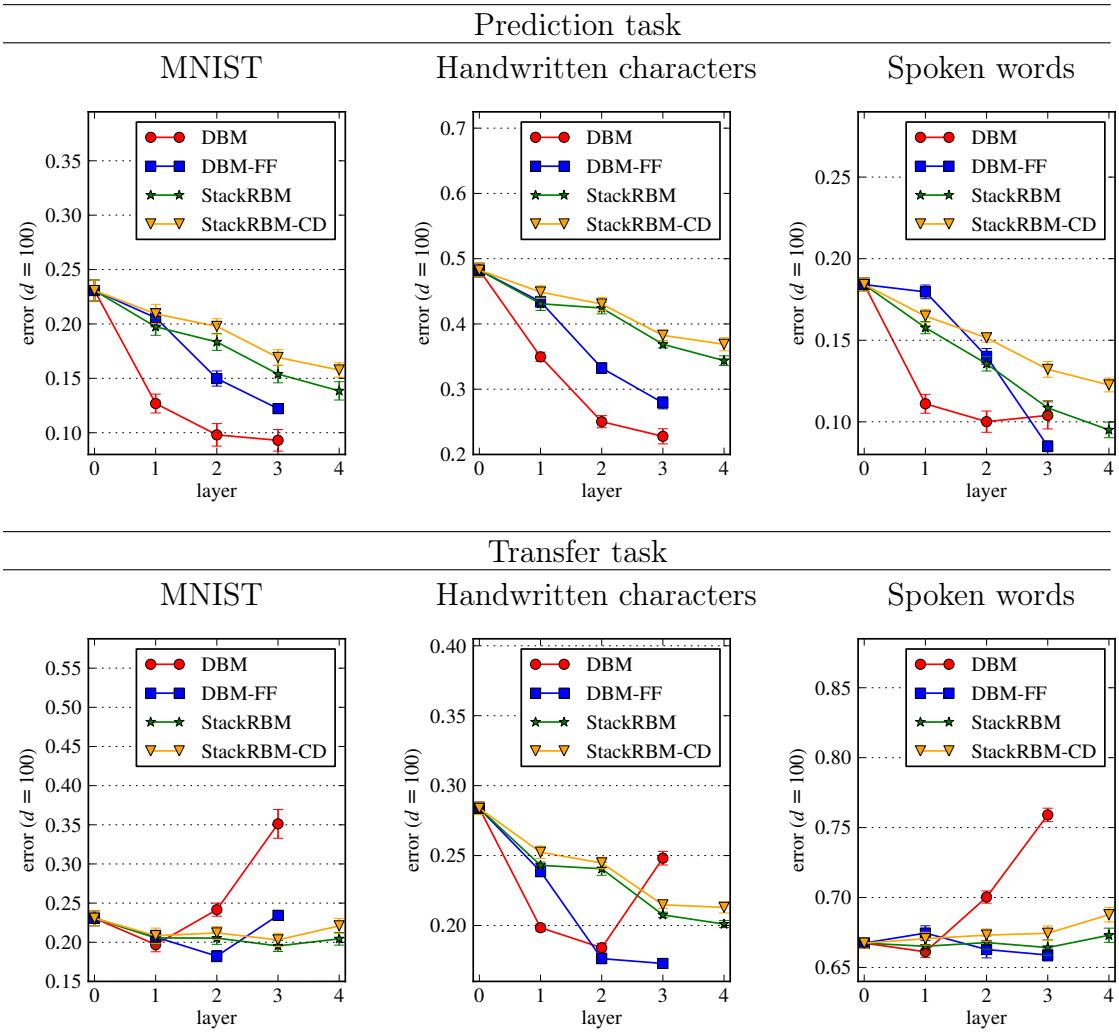


Figure 4.10.: Kernel analysis of the representation at each layer for various datasets and models. We summarize the curve $e(d)$ by only plotting its value at $d = 100$. *Top*: Analysis of the prediction task. The DBM features the most drastic reduction of error. *Bottom:* Analysis of the transfer task. The DBM also produces low error intermediate representations, but the top layer seems to be specific to classes in the training set.

4.3. Layer-Wise Oscillations in Stacked RBMs

Stacked restricted Boltzmann machines (or stacked RBMs, Hinton et al. 2006) are a widely used method to quickly and robustly pretrain backpropagation networks. This pretraining procedure has been shown to initialize the network in a better region of the parameter space, that is otherwise never reached if starting backpropagation from a random initialization (Erhan et al. 2010). In many cases, it accelerates backpropagation training and leads to better generalization. The pro-

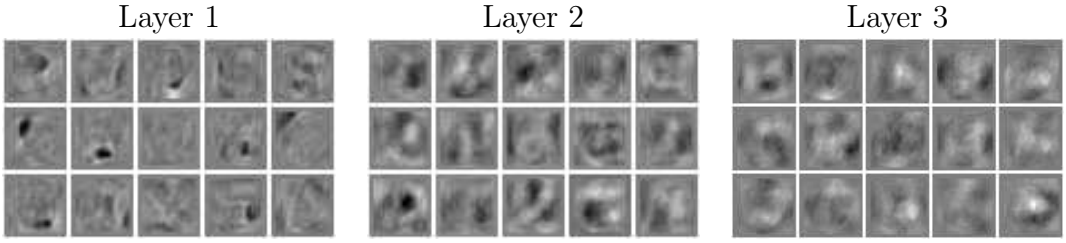


Figure 4.11.: Features learned at each layer of a DBM trained on the handwritten character recognition task. Features in higher layers are visualized by linear backprojection of the corresponding units onto the input space. We observe that as we move from the input to the top layers, filters are becoming increasingly global and invariant to small pixel-wise variations of the input.

Algorithm 1 Stacked RBMs training

- Train an RBM with two layers x and y and associated probability distribution $p_1(x, y)$ to match the input distribution $p_{\text{data}}(x)$.
- Map the input distribution onto the layer y of the RBM. This can be done in one step of Gibbs sampling starting from randomly drawn data points:

$$x \sim \text{data}$$
$$y \sim p_1(y|x).$$

- Train a new RBM with layers y and z and associated probability distribution $p_2(y, z)$ to match the mapped distribution $p_{\text{data}}(y)$.
- Map the input distribution onto the new layer z of the second RBM. This can be done with two steps of Gibbs sampling starting from randomly drawn data points:

$$x \sim \text{data}$$
$$y \sim p_1(y|x)$$
$$z \sim p_2(z|y).$$

- Repeat the procedure several times to produce a stack of arbitrarily many layers.

cedure for training and stacking RBMs is explained in details by Hinton (2012) and Bengio (2009; Section 5 and 6). It is summarized in Algorithm 1. However, the limitations of this method, in particular, the absence of top-down feedback, and its consequences on emerging representations at each layer, have been recently investigated by Arnold and Ollivier (2012) and Montavon et al. (2012a).

In this section, we predict that stacked RBMs are characterized by a layer-wise oscillation effect (Montavon et al. 2012a) where every two layers of the hierarchy are mutually similar. This is in contrast with deep Boltzmann machines where

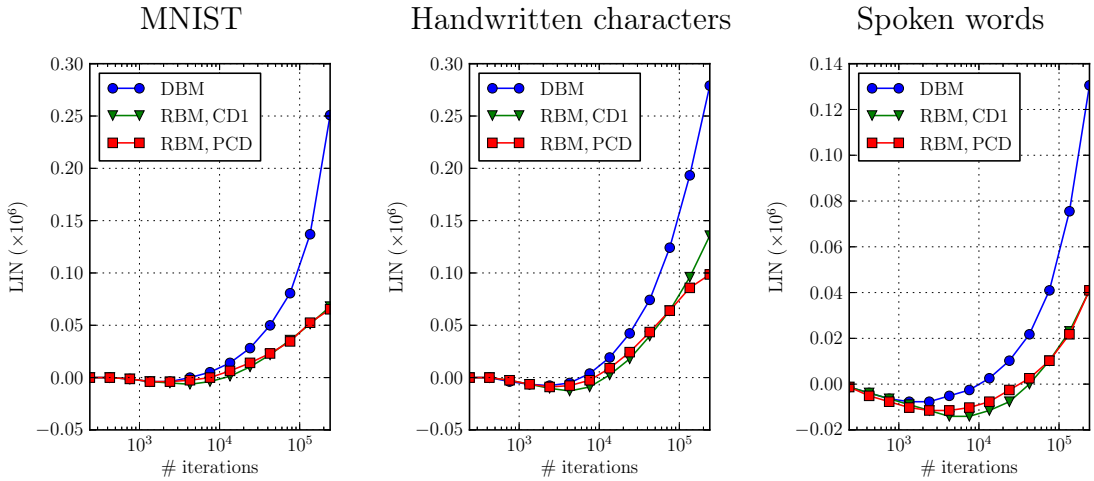


Figure 4.12.: Evolution of the layer interaction number (LIN) in the first layer of a DBM, a stack of RBMs trained with PCD, and a stack of RBMs trained with CD-1. Both RBMs are making more use of the complement mode of interaction than the DBM.

representations are expected to be increasingly dissimilar with every added layer. We will argue that such layer-wise oscillation effect is a consequence of the complement mode of interaction between units, that tends to alternatively expand and compress specific regions of the input space. Figure 4.12 shows that the complement mode of interaction (as measured by the LIN) is more prevalent in RBMs than in DBMs, presumably due to the added difficulty of mixing in shallow models (Bengio et al. 2013a).

The concept of layer-wise oscillations is illustrated in Figure 4.13, where we show a stack of RBMs (each of them composed of two units), where each pair of connected layers is set in the complement mode. This is achieved by using the same bias on each layer and negative connections between each layer so that the layer interaction number (LIN) is negative:

$$\text{LIN} = a \cdot w \cdot b = (-2.5) \cdot (-0.4) \cdot (-2.5) < 0.$$

In the input space, the small dense area is represented by a dotted line and the large sparse area is represented by a solid line. As we map these two distributions (solid and dotted) onto increasingly many layers through repeated sampling, each distribution is repeatedly switching from *narrow and dense* to *wide and sparse*. The reason for such change in data density is the nonlinearity of the sigmoid function that is able to compress some regions of the input space and expand other regions. The oscillation effect is slowly dampened as the distributions are progressively reaching their stationary mode. Similar behavior is expected for deterministic mapping at the only difference that the distribution will become point-wise after many layers.

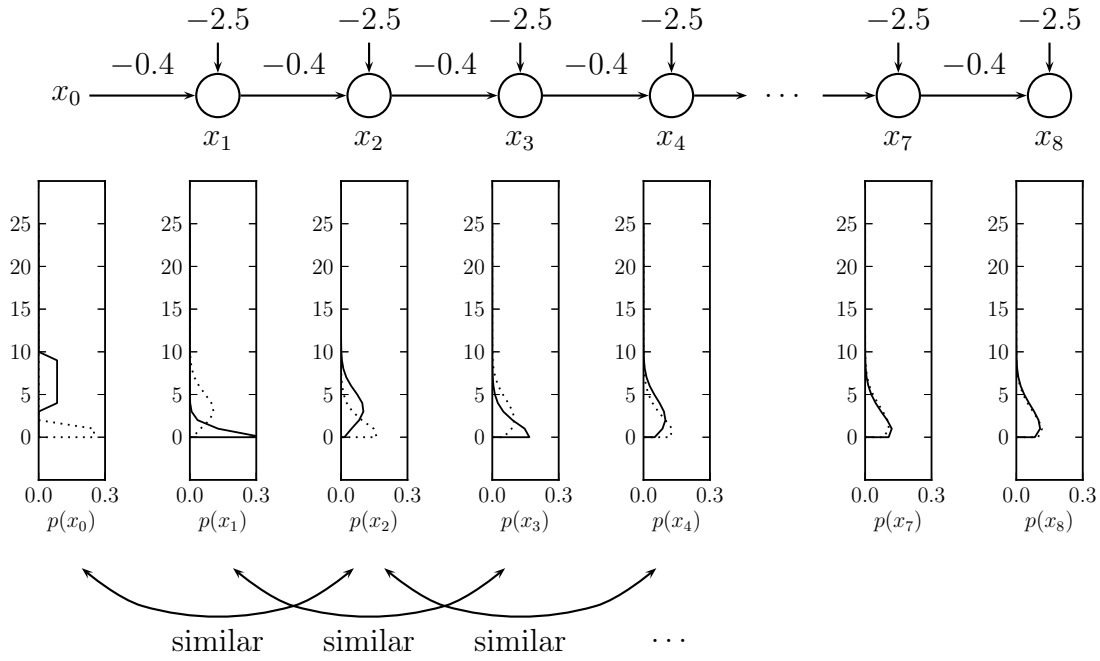


Figure 4.13.: Illustration of the layer-wise oscillation effect in a stacked model with one unit at each layer. Each layer is set in complement mode by setting the same biases at each layer and using negative connections. We can observe that each input density (solid and dotted) alternatively expands and contracts layer after layer. As a result, odd layers are mutually similar and even layers are mutually similar.

4.3.1. Measuring Layer-Wise Oscillations with Kernels

In order to measure the layer-wise oscillation effect in larger networks, we need a way to quantify these oscillations, for example, using the kernel analysis of Chapter 2. For this, we need to state the oscillation effect, that we initially described as local change in data density, in terms of kernel principal components.

Interestingly, there is a close relation between data density and its expression in the principal components of an RBF kernel: Dense regions are strongly represented in the kernel feature space while sparse regions are not. This is best understood by considering a dataset composed of a high density region and a low-density region. In this context, the kernel matrix can be roughly expressed in block-diagonal form as

$$K = \begin{pmatrix} \mathbf{1} & 0 \\ 0 & I \end{pmatrix},$$

where the first block represents the high-density region (in which all samples are mutually similar) and where the second block represents the low-density region (where samples are only similar to themselves). Here, assuming that blocks have

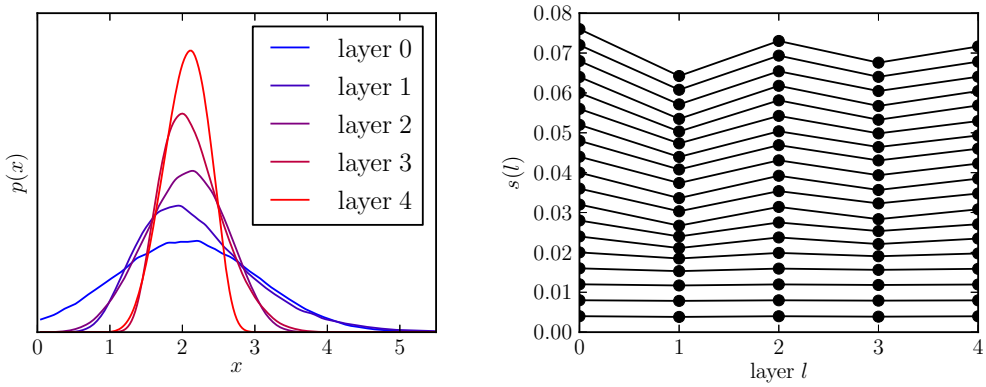


Figure 4.14.: Kernel analysis of the toy stacked RBM of Figure 4.13. On the left, mapped distributions at each layer of the network (we draw 250 input samples as $x \sim \mathcal{N}(2, 1)$). On the right, similarity $s(l)$ between the input distribution and the mapped representation at each layer l and for various dimensionalities d. Each dimensionality (ranging from 0 to 20) is plotted as a different line, from the bottom to the top. The analysis is able to capture the layer-wise oscillation effect, as shown by even layers having higher similarity.

same size, the first (scaled) eigenvector of such kernel matrix can be found to be

$$\phi_1 = u_1 \sqrt{\lambda_1} = \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix}.$$

Clearly, ϕ_1 models the high density area[2]. A mapping that densifies a certain region of the input space will see the corresponding kernel subspace emerge as principal component. On the other hand, regions that are being expanded will have their associated subspace leave the principal components.

We can quantify layer-wise changes in density, by looking at the correlation between the kernel principal components at different layers. As proposed in Chapter 2, assuming two different layers with kernels k and k' and respective eigenvectors $(u_1, \ldots, u_n)$ and $(u'_1, \ldots, u'_n)$, a measure of similarity is given by:

$$s(d) = \frac{1}{n} \sum_{i=1}^{d} \sum_{j=1}^{d} (u_i^\top u'_j)^2.$$

As an example, we apply this measure of similarity to the toy stacked RBM of Figure 4.13. The analysis is shown in Figure 4.14. We draw random samples from the distribution $x \sim \mathcal{N}(2, 1)$. These samples are given as input to the stacked RBM and are mapped deterministically onto each layer. As the input is propagated in the network, it is subject to nonlinear distortions caused by the sigmoids. In

[2]Note that this would not be the case if centralizing the kernel matrix. The variation of densities would be lost.

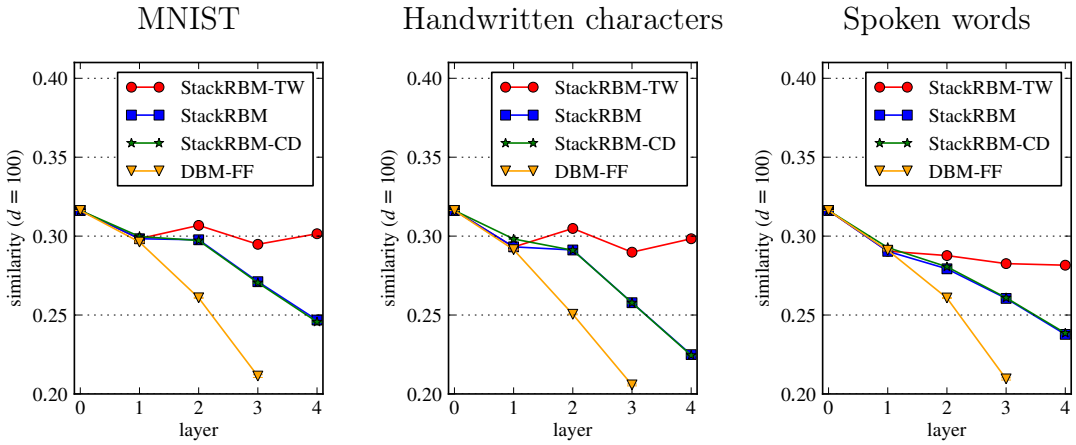


Figure 4.15.: Layer-wise evolution of the similarity between the representation and the input. In stacked RBMs, even-layered representations are more similar to the input than odd-layered representations. The similarity measure is shown for the 100 first kernel principal components. The oscillation effect is particularly strong for the stacks of RBMs with tied weights (StackRBM-TW).

particular, if looking carefully at the distributions at each layer, we can observe that the skew of the distribution tends to be reversed with every layer. Our kernel analysis is able to capture this oscillation effect: The similarity measure $s(l)$ that compares the representation at each layer with the input is shown in Figure 4.14 (right) and correctly measures that even layers are more similar to the input than odd layers.

4.3.2. Experiments

We now use the measure of similarity between representations in order to capture the oscillation effect in the same stacked RBMs as in Section 4.2. We compare the representations at each layer to the input representation. We consider three types of stacked RBMs. The first one is a stack of RBMs with tied weights (where weights at each layer as set to $W, W^\top, W, W^\top, \dots$). Given that the first RBM in the stack has 1600 visible units and 800 hidden units, the number of units at each layer is therefore 1600, 800, 1600, 800 and 1600. The second stack of RBMs learns its own weights at each layer and explicitly reduces the dimensionality of the input distribution at each layer from 1600 dimensions initially, to 800, 400, 200 and 100 dimensions with every new layer. The last stack of RBMs is trained using CD-1 instead of PCD.

Measured similarities for these three stacks of RBMs are shown in Figure 4.15 for $d = 100$ dimensions. The same dimensionality was also chosen for the previous experiments in Section 4.2. The oscillation effect is clearly visible for the stacked RBMs with tied weights (stackRBM-TW). For the standard stack of RBMs trained

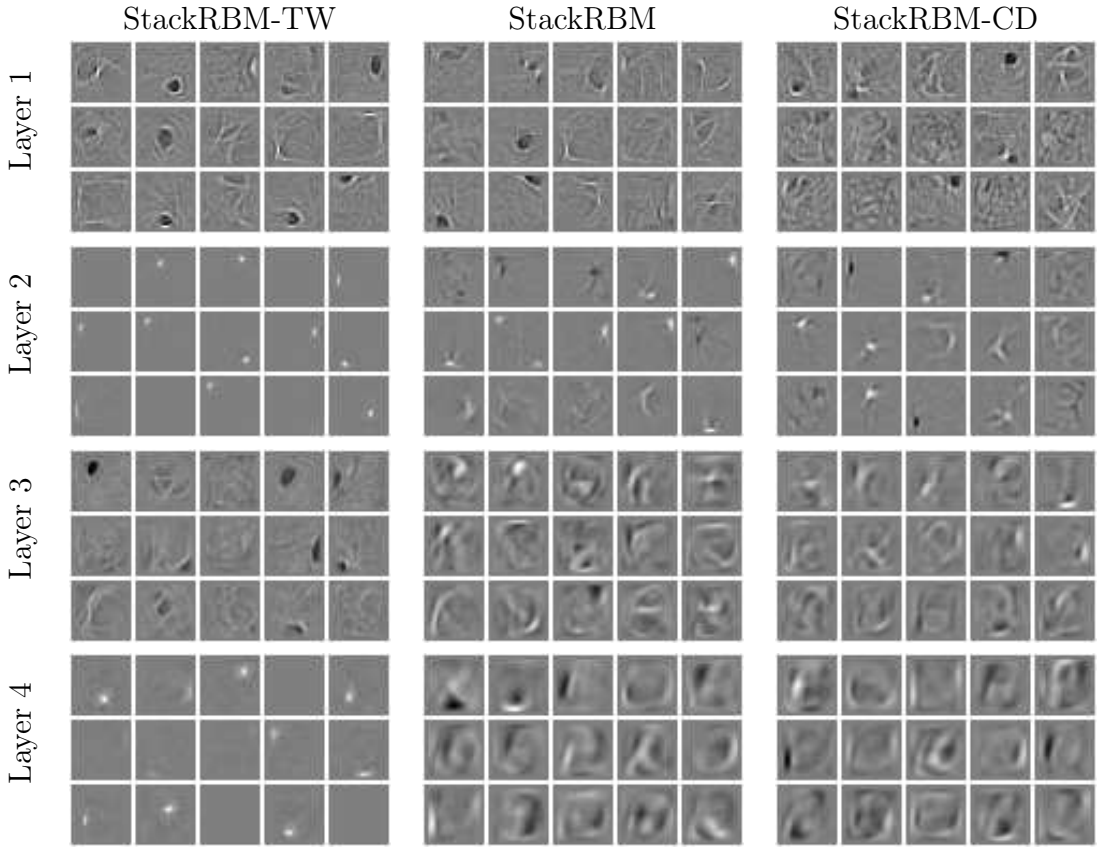


Figure 4.16.: Filters obtained at each layer of various stacks of RBM when trained on the handwritten character recognition dataset. Filters are visualized by linear backprojection onto the input. The filters at layer 2 and 4 are strongly localized and therefore similar to the input representation.

with PCD or CD-1, oscillations are less pronounced but still distinguishable in the first layers. The layer-wise evolution of the similarity $s(d)$ is very different in DBMs, where the representation at each layer is increasingly dissimilar to the input.

The layer-wise oscillation effect can also be observed by looking at the filters at each layer. This is shown in Figure 4.16 for several stacked RBMs trained on the handwritten character dataset. While the first and third hidden layers exhibit complex features covering a large range of pixels, the second layer is composed of spatially localized features that are reminiscent of the pixel-wise activation of the input layer. The spatially localized features at layer 2 and 4 are clearly distinguishable for the stacked RBM with tied weights. In the two other stacked RBMs, spatially localized filters are no longer visible at layer 4 but can still be observed at layer 2.

In a setting where we would like to learn a hierarchy of increasingly invariant representations, layer-wise oscillations appear as undesirable. Yet, the oscillation

effect arises from the ability to create complementary representation in hidden layers, that are necessary in order to model the data distribution well. We could mitigate the layer-wise oscillations by regularizing the model in order to prevent the emergence of the complementarity mode of interaction, however, doing so would prevent us from fully fitting the data distribution if many samples are available. This shows the contradiction between the global goal of learning invariant top-layer representations, and the local objective of maximizing some unsupervised criterion.

However, a main strength of layer-wise methods is that they are often easier to optimize. Therefore, we see as an important requirement to develop unsupervised algorithms that preserve the robustness of the layer-wise method, while at the same time being able to identify the high-level abstractions in top layers necessary to correctly represent the learning problem. This includes possible refinements to joint training of deep Boltzmann machines and related algorithms (see, for example, Goodfellow et al. 2013), or other forms of top down feedback, as proposed, for example, by Arnold and Ollivier (2012) and Bengio et al. (2013a).

4.4. Effect of Learning Noise on Representations

In Section 4.2, DBMs have been shown to reduce the dimensionality of the task in top layers. In practice, obtaining such top-level representations is not trivial and requires carefully designed training procedures. In this section, we will show that emerging representations at each layer are sensitive to the multiple hyperparameters of the learning algorithm, for example, the parameterization of the energy function (centered versus non-centered), the learning rate or the connectivity of the network.

Like in Chapter 3, some of these hyperparameters can be understood in terms of amount of noise that they inject into the optimization procedure. For example, a large learning rate exposes the stochastic gradient noise. Similarly, a bad parameterization of the network causes overshooting in the optimization procedure, and makes difficult for each layer in the hierarchy to exchange meaningful information. As for backpropagation networks, a predefined architecture with restricted connectivity is able to reduce the effect of learning noise, by forcing the network into the right set of solutions.

4.4.1. Effect of the Parameterization

Figure 4.17 (left) shows the kernel analysis performed on the top layer of several DBMs with different parameterizations, some of which are centered and some of which are not. We consider all parameterizations of hidden units with bias $b \in \{-2, 0, 2\}$ and offsets $\beta \in \{\text{sigm}(-2), 0, \text{sigm}(2)\}$. The DBMs are trained on the MNIST dataset and are composed of 784 visible units and two hidden layers of 200 and 25 hidden units. "Centered+" means that the energy function is continuously recentered at training time. We observe that non-centered DBMs

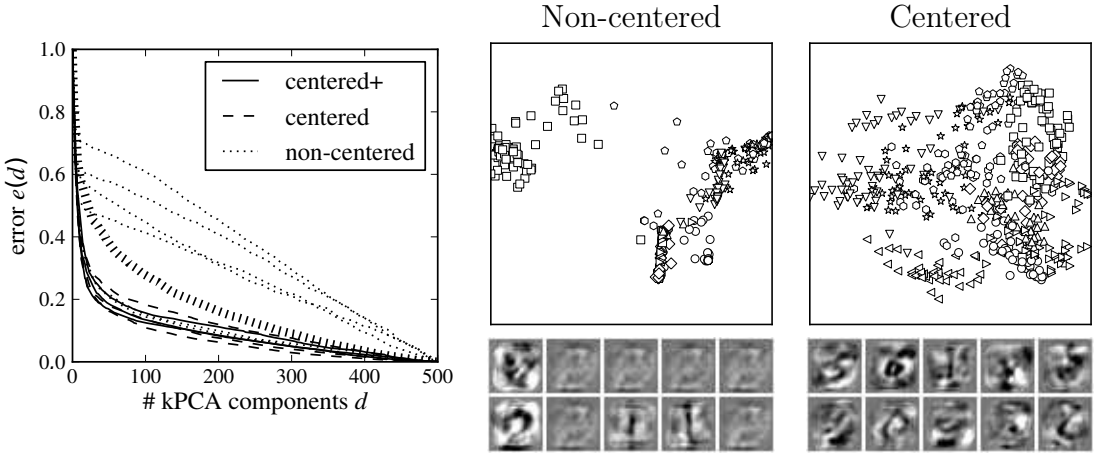


Figure 4.17.: Plots taken from the paper by Montavon and Müller (2012) showing the effect of centering on emerging representations. *Left:* Kernel analysis performed on the input (thick dashed line) and the top layer of several centered and non-centered DBMs trained on MNIST. *Right:* Two-dimensional PCA performed on the top layer of a non-centered and a centered DBM trained on MNIST, along with second layer filters. Different markers indicate different classes. Filters are visualized by linear backprojection of second layer units onto the input space.

have a few highly discriminative kernel principal components, but that the next ones are dominated by a large noise bed, as shown by the linearly decaying error $e(d)$. In contrast, centered DBMs also produce the desired highly discriminative kernel principal components, but keep the noise level low.

Figure 4.17 (right) shows the same effect of centering from the perspective of a two-dimensional linear PCA produced on the activity of top-layer units and from the perspective of second layer filters (linearly backprojected onto the input space). Two-dimensional PCA for non-centered DBMs shows two dense clusters separated by a large margin, suggesting that the DBM is able to identify two groups of handwritten digits with different classes, but at the same time, loses discriminative information between different classes within these groups. Also, in a non-centered DBM, filters are often redundant, suggesting that the top-layer representation spans a very limited subspace. In both cases, results show that the top-layer representation of a centered DBM is richer and has more discriminative features than a non-centered one.

4.4.2. Effect of the Architecture

Another approach to tackle the problem of learning noise is to incorporate built-in structure into the DBM, for example, by restricting connectivity in the first layers and only allowing global connectivity in the top layers. Such architecture decouples

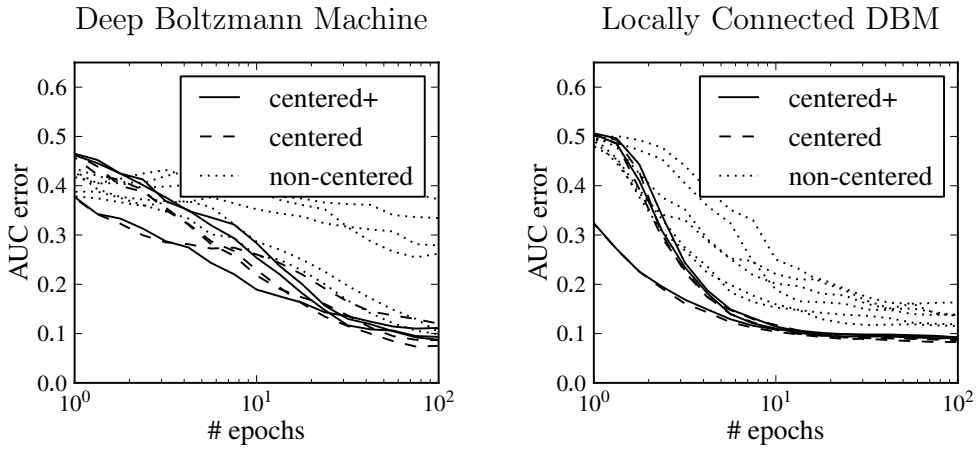


Figure 4.18.: Plots by Montavon and Müller (2012) showing the evolution of the quality of the top-layer representation as a function of the number of training epochs for a plain DBM and for a locally connected DBMs. Representations improve at different speeds throughout training. Centering and local connections help to quickly build the desired top-layer representation.

the modeling of local variations (small pixel-wise transformation of the input) from more global variations, for example, jumping between different modes of the distribution. Such decomposition restricts the space of solutions, and guides the learning algorithm towards a smaller set of plausible solutions. As a consequence, we can expect the training procedure to be faster. The accuracy of the method ultimately depends on the validity of the local connectivity assumption, that is, whether classes are truly invariant to localized variation in the pixel space.

In order to test the effect of adding structure to a DBM, we train a locally connected DBM constructed by allocating to each hidden unit in the first layer, a random patch of 6×6 pixels in the input space. We use 400 and 100 hidden units in each layer. The quality of the representation measured in AUC (area under the curve $e(d)$) is shown in Figure 4.18 for plain and locally connected DBMs. In general, locally connected DBMs produce the desired top-layer representation quickly (after a few epochs) while it takes one order of magnitude more epochs for plain DBMs. Interestingly, locally connected DBMs are also more resilient to the quality of the parameterization of the model, that is, the performance gap between centered and non-centered DBMs becomes smaller.

Finally, we measure in Figure 4.19 the generative error for the same fully connected and locally connected DBMs. We observe that the effect of bad parameterization on generative performance is inverse to the effect on the quality of the top layer: In a simple DBM, the top layer is simply discarded from the model and the generative performance is not affected. However, in the locally connected DBM, the local connections—that were supposed to favor the emergence of a good top level representation—significantly reduce the generative performance when the

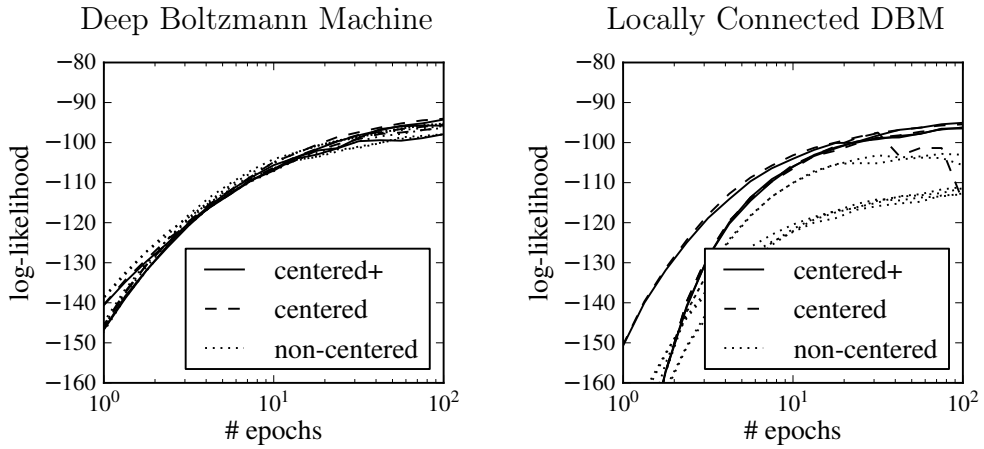


Figure 4.19.: Plots by Montavon and Müller (2012) showing the evolution of the generative performance of the model as a function of the number of training epochs for a plain DBM and for a locally connected DBM. Unlike the AUC shown in Figure 4.18, the non-centered DBM suffers from its bad parameterization only in locally connected DBMs (i.e. when depth is required to properly model the input distribution).

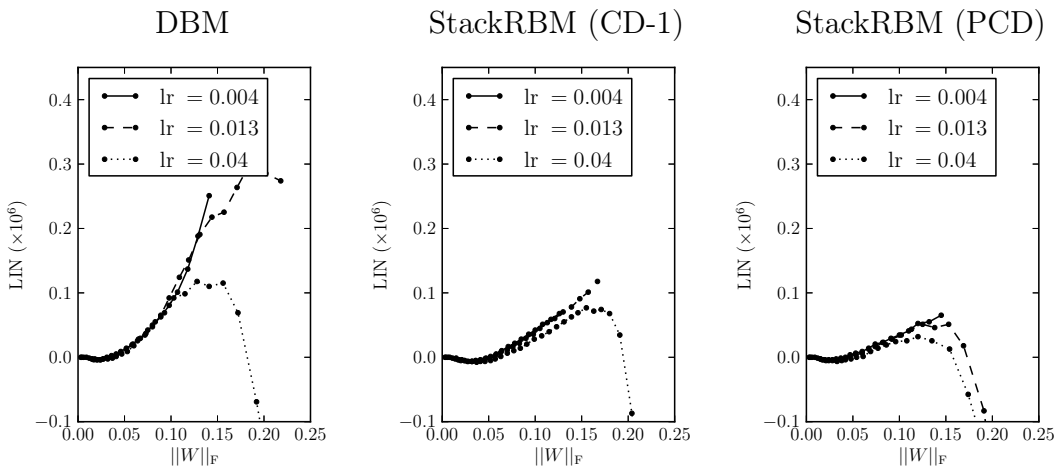


Figure 4.20.: Layer interaction number (LIN) in the first layer of each model trained on the MNIST dataset. The LIN is plotted as a function of the model complexity (measured as $||W||_{\mathrm{F}}$, where W is the weight matrix in the first layer). Higher learning rates lead to more complementarity between layers as shown by a smaller LIN.

network is non-centered. This is because the locally connected DBM is not able anymore to discard the top layer from the generative model, since it is structurally constrained to use it. As a consequence, the model is heavily exposed to the top-layer parameterization-induced noise.

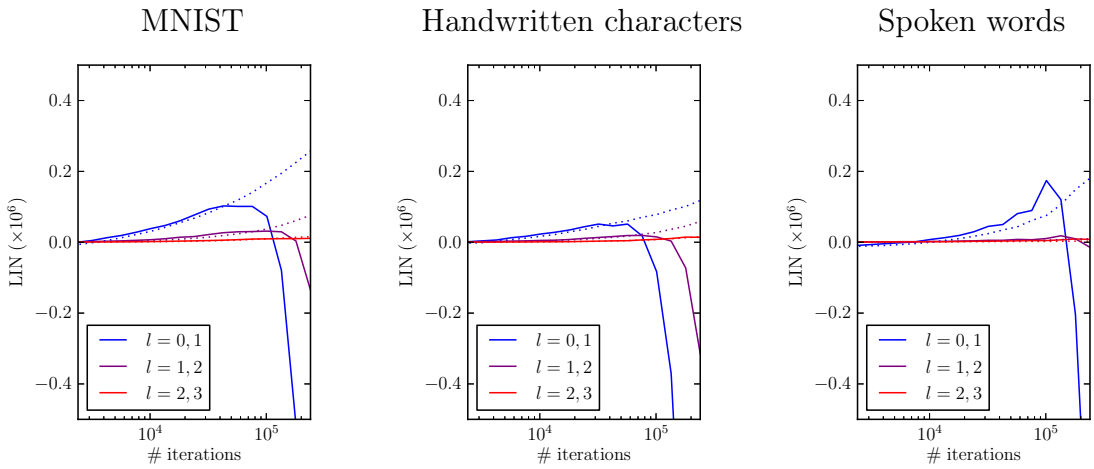


Figure 4.21.: Comparison of the layer interaction number (LIN) at each layer of stacks of RBMs trained with PCD (solid line) or CD-1 (dotted line), and with a high learning rate of 0.013. The LIN sharply drops to negative values in the late stage of PCD training, due to the increased difficulty of mixing, while it rises steadily for CD-1.

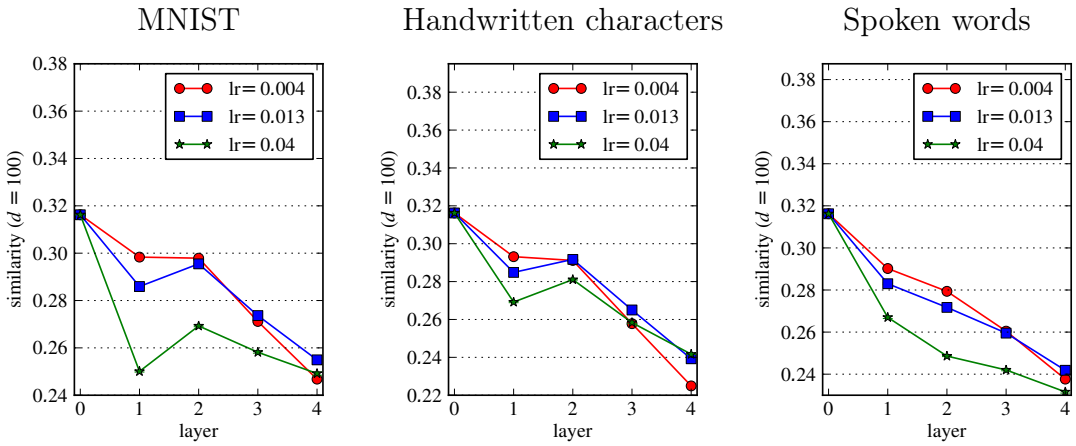


Figure 4.22.: Effect of the learning rate on layer-wise oscillations in a stack of RBMs trained with PCD. In general, a higher learning rate translates into stronger oscillations.

4.4.3. Effect of the Learning Rate

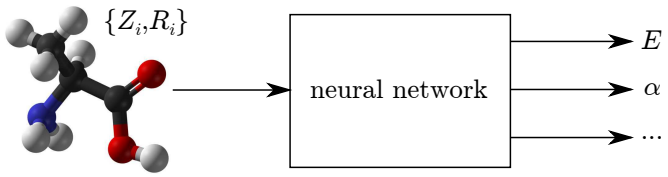
We finally look at the effect of learning rate on the DBMs and stacked RBMs of Section 4.2 and 4.3. Figure 4.20 shows the layer interaction number (LIN) for the first layer of a DBM and stacks of RBMs trained with different learning rates. We observe that an increased learning rate tends to favor solutions that make more use of the complement mode of interaction, that is, solution for which the LIN becomes smaller. An explanation for this effect is the following: As the learning

rate increases, there is less time for the Gibbs chains approximating the model statistics to mix properly. Thus, for the objective to be minimized, the different modes of the data distribution should be connected by regions of non-zero data density (otherwise, the chains used for the negative phase of the algorithm would not mix fast enough). Creating these large non-zero data density regions is best achieved by the complement mode of interaction, as we have seen in Section 4.1. This is illustrated in Figure 4.21 where the stack of RBMs trained with PCD experiences a phase transition characterized by a sharp drop-off of the LIN in the late stage of training.

Finally, we look at the layer-wise oscillation effect as a function of the learning rate. We have seen in the previous example, that a high learning rate tends to drive pairs of adjacent layers in the complement mode of interaction. Based on the argument of Section 4.3, this increased level of complementarity should translate into stronger layer-wise oscillations. Figure 4.22 empirically validates this hypothesis, showing that increased learning rates amplify the layer-wise oscillations in stacked RBMs.

5. Learning Molecular Electronic Properties with Neural Networks

In this chapter, we present an application of deep networks to quantum chemistry: predicting molecular electronic properties from molecular geometries. An illustration of the problem is shown below:



Here, $\{Z_i, R_i\}$ are the charge and Cartesian coordinates of all atoms in the molecule, and $E, \alpha, \ldots$ are some of the properties to predict. Quantum chemical computations are often carried out by deductive models derived from laws of physics such as those embodied in the Schrödinger's Equation $H\Psi = E\Psi$. Successful physics-based methods exploit redundancies in the original equation, in order to accelerate computations. For example, the DFT method (Hohenberg and Kohn 1964) bypasses the extremely costly computation of the wave function Ψ and compute ground state energies E as a minimization problem over much lower-dimensional quantities. Over the past decades, rise in computational power, and advances in chemical theory (Zerner et al. 1980, Perdew et al. 1996), enabled the simulation of relatively large compounds. However, none of the current methods is able to systematically and accurately compute molecular properties for billions of molecules.

A radical departure from physics-based methods is to learn an inductive model of the underlying quantum physical computation from a small set of labeled examples (Rupp et al. 2012). An advantage of the inductive paradigm is that it can deal with risk minimization and dimensionality reduction in a more direct manner: Constraining the inductive model to use only a few steps of dimensionality reduction and nonlinear processing leads to predictive models that are much faster than the original quantum chemical computation. Also, physics-based methods often produce a systematic bias on the prediction, that can be in some cases difficult to correct. This is a lesser problem in inductive models, because the bias is automatically reduced as more i.i.d. data becomes available. For this, we should however provide unbiased labels to the learning machine.

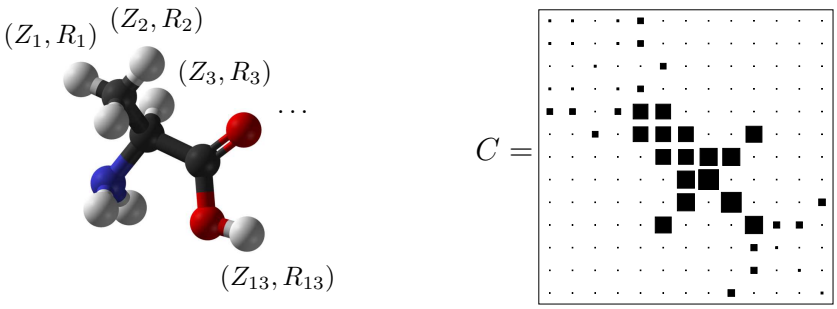


Figure 5.1.: Alanine molecule ($C_3H_7NO_2$) along with its Coulomb matrix representation.

The general idea of applying machine learning to chemistry has existed for a long time. Models that are often referred as quantitative structure-activity relationships, or QSARs (Hansch et al. 1995), have been applied successfully to both conventional chemistry (e.g. drug design, Burbidge et al. 2002, Schroeter et al. 2007), and quantum chemistry (Balabin and Lomakina 2009). QSAR methods usually rely on human-engineered molecular descriptors, that are often application-specific. Machine learning was also used to predict the molecular dynamics of single molecules (Lorenz et al. 2004, Behler 2011). Machine learning was more recently applied to modeling density functionals in small particle systems (Snyder et al. 2012; 2013), with potential application to molecular relaxation and modeling of quantum chemical reactions.

5.1. Representing Molecules

Molecules are highly structured graph-like objects that are not trivial to represent in a consistent way. In particular, the variety of possible molecular sizes and configurations makes it difficult to build a simple and invariant vector space representation. In this section, we discuss how to build a representation of molecules that is at the same time compact, complete, and invariant to translation, rotation or atoms ordering. Within Born-Oppenheimer approximation, a relaxed molecule can be fully described as the charge Z_i and the three-dimensional Cartesian coordinates R_i of all atoms in the molecule:

$$M = \{Z_i, R_i\}_{i=1}^{d} \tag{5.1}$$

Such representation is complete as it contains in principle all necessary information to compute the molecular properties. However, it does not handle molecules with different numbers of atoms. This can be addressed by adding "dummy atoms" with charge $Z = 0$, that do not influence the measured physical system. A more serious issue is the lack of invariance with respect to translation, rotation and atoms ordering.

In order to incorporate translation and rotation invariance, Rupp et al. (2012) proposed to represent molecules as a *Coulomb matrix* of size $d \times d$ defined as:

$$C_{ij} = \begin{cases} \frac{1}{2} Z_i^{2.4} & i = j, \\ \frac{Z_i Z_j}{||R_i - R_j||_2} & i \neq j. \end{cases} \tag{5.2}$$

The diagonal terms are a polynomial fit of charges to total energies of the free atoms. The off-diagonal terms are the repulsion energy between pairs of atom nuclei. An example of molecule with its associated Coulomb matrix is given in Figure 5.1. It is easy to see that the Coulomb matrix is invariant to translation and rotation, as positions R_i are now expressed only in terms of relative distances. However, this representation is not invariant to atoms ordering.

5.1.1. Random Coulomb Matrices

A simple technique to gain a certain level of invariance with respect to atoms ordering is to associate to each molecule several Coulomb matrices resulting from different atoms orderings. More precisely we can define for each molecule M a probability distribution $p_M(C)$ over Coulomb matrices, from which we can draw samples. A method proposed by Montavon et al. (2012b) to randomly draw Coulomb matrices is given in Algorithm 2.

Algorithm 2 Sampling Random Coulomb Matrices

function SAMPLE(M)
  Generate a valid Coulomb matrix C of molecule M
  Find the permutation P that sorts $\left[||C_i|| + \mathcal{N}(0, \sigma^2)\right]_{i=1}^{d}$
  $C \leftarrow \text{SORTROWS}_P(\text{SORTCOLUMNS}_P(C))$
  return C
end function

The parameter σ controls the diversity of the probability distribution from which Coulomb matrices are drawn. Figure 5.2 shows the effect of this parameter on the input distribution. Each color designates a different molecule and each point of same color represents a different Coulomb matrix of the same molecule. If the parameter σ is set to zero, we have a sorted Coulomb matrix (initially proposed by Rupp et al. (2012)). As we consider larger values for σ, the input space becomes more densely populated. If the parameter σ goes to infinity, we have a uniform probability distribution over all possible atoms orderings. Generally, we face the following tradeoff: For small values of σ, the dataset is easy to fit, but the model easily overfits. On the other hand, if σ is large, the model is less likely to overfit, but it takes more time to train.

The idea of extending the dataset by artificially generating more samples is a well-known technique in machine learning. For example, Vincent et al. (2010)

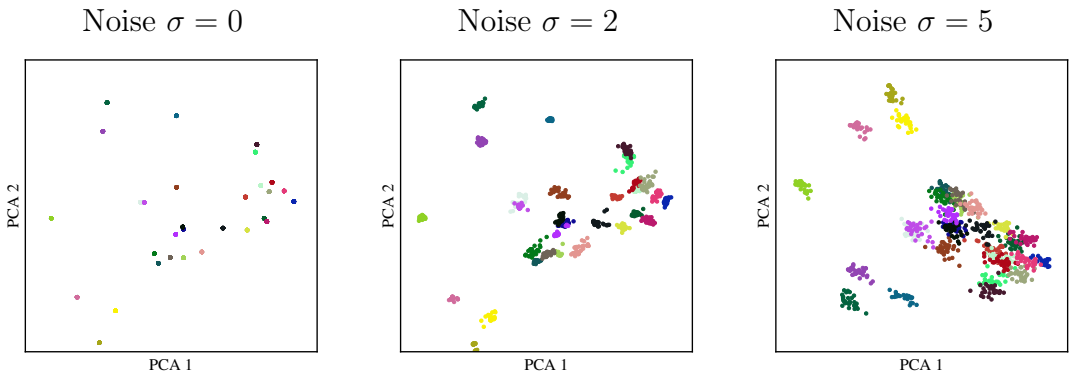


Figure 5.2.: PCA in the input space (with a Manhattan distance) showing the effect of adding permuted Coulomb matrices. Several permutation of the same molecules are depicted by the same color. Adding multiple permutations of the same molecule produces a cloud of data points. The variety of permutations is controlled by the noise parameter σ.

add noise to the input of autoencoders in order to learn more robust representations. Simard et al. (1998) or Ciresan et al. (2010) add human-engineered elastic distortions of handwritten digits to the dataset in order to learn more invariant representations. When generating artificial samples, care must be taken in order to make sure that the generated distortions do not add label noise. A subtle difference in our case is that we know in advance that the generated distortions (i.e. atoms orderings) leave the label invariant. Indeed, the molecular property to predict is by construction invariant to the choice of atoms ordering. Therefore, the optimal level of noise σ will be essentially determined on a computational basis.

5.2. Predicting Molecular Electronic Properties

The problem of mapping Coulomb matrices to molecular energies gives rise to several challenges for deep networks. Section 5.2.1 discusses the difficulty of learning accurately nonlinear functions when input and output variables have high information content. Section 5.2.2 describes a pathological scaling issue arising in the Coulomb matrix representation. Finally, Section 5.2.3 proposes to solve both problems using a binarized expansion of Coulomb matrices as input to the deep network.

5.2.1. The Problem of Wide Range Dependencies

Deep networks have been overwhelmingly studied on problems where input and output are composed of multiple variables of low information content. For example, the handwritten recognition problems considered in Chapters 3 and 4 assume binary inputs describing whether pixels are activated or not, and a set of binary

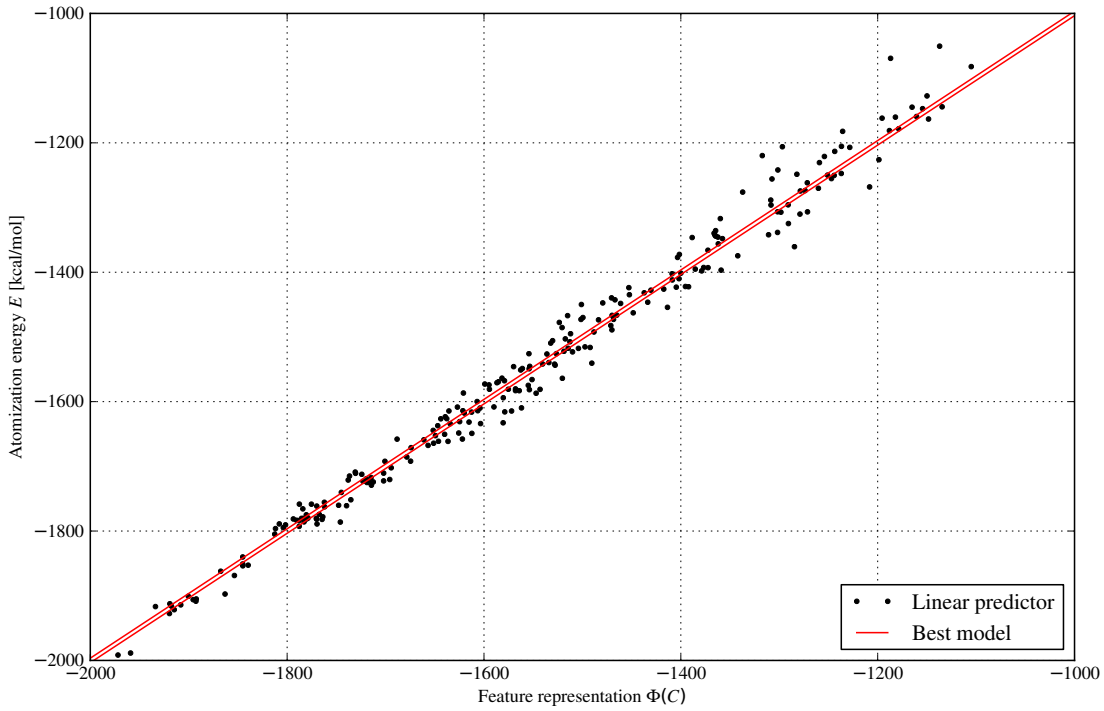


Figure 5.3.: Illustration of the problem of wide range dependencies. The label is plotted against a simple linear model (here fitted on all data). The two red lines represent the error bars that are produced by our best model (based on neural networks or Laplacian KRR). Error bars are strikingly small compared to the range of energies that has to be predicted.

outputs that are class indicators. The output is generally invariant to small variations of the input (e.g. changing the intensity of a pixel).

Instead, in our quantum chemistry problem, input and output dimensions have a high information content. First, because of the regression nature of the problem. Second, because large range of energies have to be predicted with high accuracy. There is approximately a factor 100× between the standard deviation of the labels and the standard deviation of the prediction error obtained by our best models. Similarly large variations also occur in the Coulomb matrix between the size of the input domain and the allowed measurement error. These differences of scale are illustrated in Figure 5.3, where the error bars obtained by our best model are much smaller than the output domain.

5.2.2. The Problem of Pathological Scales

Another difficulty arises from the inability of the Coulomb matrix representation to properly handle compositionality. This gives rise to a pathological scaling problem in the space of Coulomb matrices. To understand this, let us consider four

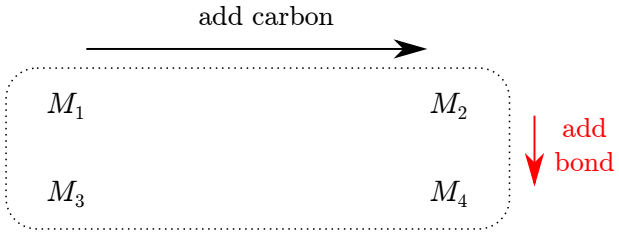


Figure 5.4.: Cartoon illustrating the problem of pathological scales: Four molecules are represented in a subspace of the Coulomb matrices that spans the addition/removal of a carbon atom, and the addition/removal of a hydrogen bond. The variation that is relevant for predicting the atomization energy (adding one bond) lies on the small component of variance.

molecules along with their Coulomb matrices:

$$M_1 = \text{H} \mid \text{H} \qquad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{pmatrix}$$

$$M_2 = \text{C} \mid \text{H} \mid \text{H} \qquad \begin{pmatrix} 36.9 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{pmatrix}$$

$$M_3 = \text{H—H} \qquad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0.7 \\ 0 & 0.7 & 0.5 \end{pmatrix}$$

$$M_4 = \text{C} \mid \text{H—H} \qquad \begin{pmatrix} 36.9 & 0 & 0 \\ 0 & 0.5 & 0.7 \\ 0 & 0.7 & 0.5 \end{pmatrix}$$

The letters H and C denote a hydrogen and carbon atom, respectively. The symbol "—" indicates a bond between two atoms and the symbol " | " indicates that the systems on both sides are separated by an infinite distance. We first note that the space of Coulomb matrices spanned by these four molecules is two-dimensional and only characterized by the presence or absence of a carbon atom and the presence or absence of a hydrogen bond. The first dimension has high variance and the second one has low variance. A depiction of this two-dimensional space of molecules is given in Figure 5.4. Unfortunately, in the context of predicting molecular atomization energies, the discriminative component—here, the presence or absence of a hydrogen bond—corresponds to the lowest component of variance. The problem can be made arbitrarily bad, by considering heavier atoms than carbon or more isolated atoms. While this example suggests that we should find better representations of molecules, we can also show that not all kernels are as badly affected by such degenerate case.

The Laplacian kernel[1] was shown to work well on empirical data (Hansen et al. 2013) and can be shown in this example to be more robust to this scaling issue: Let us suppose that, having observed M_1, M_2 and M_3, we would like to generalize to M_4. Clearly, we would like the similarity between M_3 and M_4 to be high since both molecules have the same atomization energy. The Laplacian kernel helps with this respect in two ways: First, the Manhattan distance used by the Laplacian kernel favors data points that are aligned to M_4 in the coordinate system (from

[1]The Laplacian kernel is defined as $k(x, x') = e^{-\frac{||x-x'||_1}{\sigma}}$ where $||x - x'||_1$ is the L^1 (or Manhattan) distance.

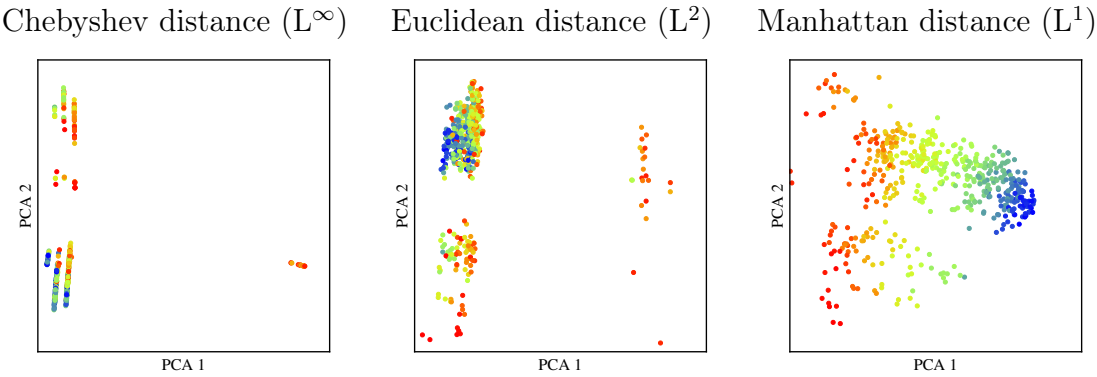


Figure 5.5.: Principal component analysis of the QM7 dataset for different distances. Data points are colored according to atomization energies. Red points are molecules with high atomization energies (unstable) and blue points are molecules with low atomization energies (stable). The Manhattan distance produces a better representation.

that perspective, $d(M_3, M_4)$ is substantially smaller than $d(M_1, M_4)$). Second, the Laplacian kernel has heavier tails than a Gaussian kernel, making $k(M_2, M_4)$ and $k(M_3, M_4)$ more equal. For these two reasons, the Laplacian kernel will therefore contribute to make M_3 and M_4 more similar.

Figure 5.5 shows a two-dimensional PCA of the QM7 dataset (see Appendix A.1 for details) color-coded by atomization energies. With a Chebyshev (or L^∞) distance, the input space is strongly clustered, with a lot of label variability in each cluster. As we move to the Euclidean (or L^2) distance, and finally, the Manhattan (or L^1) distance, the clusters are merged into a single giant component where atomization energies are much better resolved. This gives empirical support to the argument made above, about the suitability of Laplacian kernels (using a Manhattan distance) for comparing Coulomb matrices.

5.2.3. Binarizing Representations

We propose in the context of neural networks, to address both problems of wide range dependencies and pathological scales, by a simple method, also known as thermometer coding (Jeon and Choi 1999), that consists of expanding each input variable C_{ij} into a set of many binary variables:

$$\Phi(C_{ij}) = \left[\ \ldots \ , \ \text{step}(C_{ij} - \theta) \ , \ \text{step}(C_{ij}) \ , \ \text{step}(C_{ij} + \theta) \ , \ \ldots \ \right], \tag{5.3}$$

where $\text{step}(x) = 1_{x>0}$, and where $\theta > 0$ is the granularity parameter. (In our experiments, the mapping is made smoother, by replacing the step function by a sigmoid nonlinearity.) This binarization of data can be made arbitrarily fine-grained by lowering θ. If applying this transformation to an entire Coulomb matrix of dimension $d \times d$, we obtain a tensor of dimension $d \times d \times \infty$, most dimensions of which, are constant across the dataset, and can therefore be pruned.

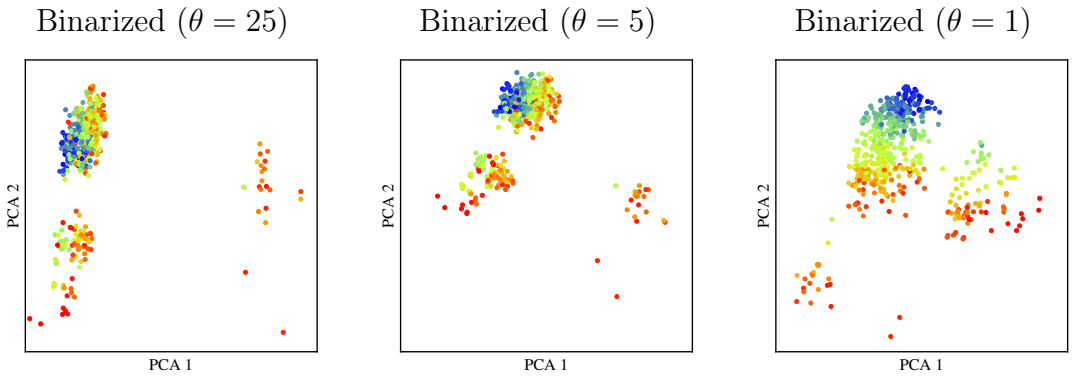


Figure 5.6.: Linear PCA in the binarized representation space for various granularities θ. As we lower θ, the principal components become increasingly similar to those obtained with a Manhattan distance. The color code is the same as in Figure 5.5.

The binarized representation solves the problem of wide range dependencies: Any local variation between input and output can be easily modeled by considering only the dimensions of the three-dimensional tensor that correlate to this local variation.

The binarized representation also solves the problem of pathological scales: It can be shown that for infinitely small θ, any L^p distance in such binarized space is equivalent to a Manhattan distance in the original space. Let x and y be two vectors and $\Phi(x), \Phi(y)$ be two matrices corresponding to their binarized embeddings. We show that $||\Phi(x) - \Phi(y)||_p^p = ||x - y||_1^1$:

$$
\begin{aligned}
||\Phi(x) - \Phi(y)||_p^p &= \sum_i \int_{k=-\infty}^{\infty} (\Phi_k(x_i) - \Phi_k(y_i))^p \cdot \mathrm{d}k \\
&= \sum_i \int_{k=-\infty}^{\infty} (1_{x_i > k} - 1_{y_i > k})^p \cdot \mathrm{d}k \\
&= \sum_i \int_{k=-\infty}^{\min(x_i,y_i)} 0^p \cdot \mathrm{d}k + \int_{k=\min(x_i,y_i)}^{\max(x_i,y_i)} 1^p \cdot \mathrm{d}k + \int_{k=\max(x_i,y_i)}^{\infty} 0^p \cdot \mathrm{d}k \\
&= \sum_i |x_i - y_i| = ||x - y||_1^1.
\end{aligned}
$$

Figure 5.6 shows the effect of binarization on the corresponding linear PCA representation. Like in Figure 5.5, data points are color-coded according to their atomization energy. As θ gets small, the PCA space becomes less clustered, and the principal components better resolve the quantity to predict. With very small θ, we obtain similar principal components to those obtained with a Manhattan distance.

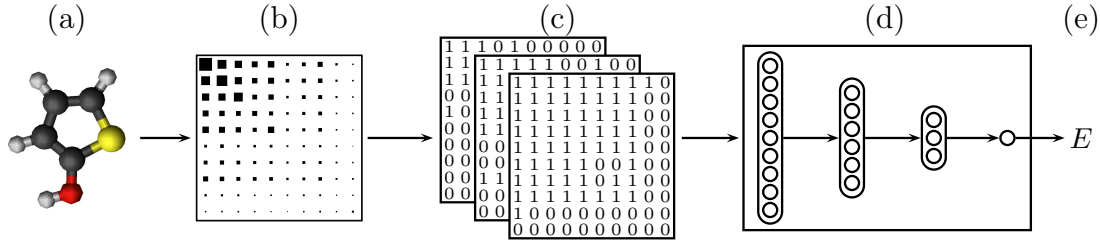


Figure 5.7.: Diagram taken from the paper by Montavon et al. (2012b) showing the data flow from the raw molecular geometry to the predicted atomization energy. A random Coulomb matrix is built for the input molecule (a) using Algorithm 2. The resulting Coulomb matrix (b) is binarized using Equation 5.3, producing a binary Coulomb tensor (c). The Coulomb tensor is pruned from its non-varying dimensions, flattened, and fed to a deep neural network (d). The neural network produces at its output a prediction of the molecular atomization energy (e) or other molecular electronic properties.

5.3. Results and Discussion

In this section, we report the results obtained with our neural network model on the QM7 dataset. This dataset is described in Appendix A.1. Then, we study the effect of the dataset size, the permutation noise σ (described in Section 5.1.1) and the binarization parameter θ (described in Section 5.2.3) on the speed of learning and the generalization performance.

Figure 5.7 shows the overall prediction pipeline used by our deep network. The input molecule is represented as a random Coulomb matrix. The random Coulomb matrix is binarized and fed to a deep network that predicts the molecular property. When predicting multiple molecular electronic properties, we use instead a multitask network (Caruana 1997), that predicts all properties simultaneously, and shares low-level features between different properties. A description of the neural network training procedure is given in Appendix A.2.

Table 5.1 shows 5-fold cross-validated results obtained on the QM7 dataset. 80% of data is used for training and the remaining 20% for testing. In order to reduce variance of the error estimator, the dataset is stratified such that each fold contains a representative range of molecular atomization energies. The backpropagation network is compared to linear regression and to kernel ridge regression with Gaussian and Laplacian kernels. Due to the difficulty to scale kernel-based methods to many samples, only 8 permutations per Coulomb matrix are retained. On the other hand, neural networks are fed with an infinite stream of random Coulomb matrices generated on the fly. We observe that there are large differences of performance between studied models. Prediction errors range from 20 kcal/mol for a simple linear predictor to 3 kcal/mol for the best models. While Laplacian kernel ridge regression and deep networks have similar performance when trained with random Coulomb matrices, it is interesting to note that backpropagation networks

Learning algorithm	Representation	MAE	RMSE
Mean predictor	None	179.02 ± 0.08	223.92 ± 0.32
Linear regression	Sorted Coulomb	20.72 ± 0.32	27.22 ± 0.84
Gaussian kernel ridge regression	Sorted Coulomb	8.57 ± 0.40	12.26 ± 0.78
	Random Coulomb	6.76 ± 0.21	10.09 ± 0.76
Laplacian kernel ridge regression	Sorted Coulomb	4.28 ± 0.11	6.47 ± 0.51
	Random Coulomb	3.07 ± 0.07	4.84 ± 0.40
Deep network	Sorted Coulomb	11.82 ± 0.45	16.01 ± 0.81
	Random Coulomb	3.51 ± 0.13	5.96 ± 0.48

Table 5.1.: Table aggregated from the papers by Montavon et al. (2012b) and Hansen et al. (2013) showing the prediction accuracy in kcal/mol of various models on the QM7 dataset. MAE denotes the mean absolute error and RMSE denotes the root mean square error.

benefit to a much greater extent from using these random Coulomb matrices compared to simple sorted Coulomb matrices. This illustrates how additional data can overcome the intrinsic regularization issues of neural networks.

Table 5.2 shows prediction error for other molecular properties. Here, energies are given in eV (1 eV $\approx$ 23.06 kcal/mol). We use 5000 molecules for training and the remaining 2211 molecules for testing. We can observe that the same deep network can predict a wide range of molecular properties to a similar level of accuracy, at the exception of E^*_{max}. This shows the generic nature of the machine learning approach when compared to physics-based methods, that would require special algorithms and approximations for each property.

Figure 5.8 (top left) shows the effect of the granularity parameter θ used for binarization on the learning speed. The smaller θ, the faster learning. However, for very small values θ, computation time starts to increase as $1/\theta$. Therefore, although the binarization technique is effective at implementing the Manhattan distance in a deep network, it is not computationally efficient beyond a certain level of granularity. Ultimately, we would like to find more direct techniques to encode Manhattan distances in deep networks.

Figure 5.8 (top right) shows the effect of the dataset size on the training speed. We consider the full QM7 dataset (7165 molecules) and two subsets restricted only to the molecules of type $C_5N_1O_1$ (1020 molecules) and $C_5N_1O_1H_9$ (319 molecules). In the last experiments, we use 50% of data for training and 50% of data for testing. As we decrease the dataset size, learning becomes much faster. This is in part, because there is less information to model in a small dataset, but also because the smaller dataset gives rise to a better conditioned optimization problem, that suffers less from the issue of wide-range dependencies described in Section 5.2.1.

Figure 5.8 (bottom) shows the effect of the permutation noise parameter σ on the test error, for increasingly large subsets of data. We display learning curves

Property	MAE	RMSE	Property	MAE	RMSE
E (PBE0)	0.16	0.36	LUMO (PBE0)	0.12	0.20
α (PBE0)	0.11	0.18	LUMO (ZINDO)	0.11	0.18
α (SCS)	0.08	0.12	IP (ZINDO)	0.17	0.26
HOMO (GW)	0.16	0.22	EA (ZINDO)	0.11	0.18
HOMO (PBE0)	0.15	0.21	E^*_{1st} (ZINDO)	0.16	0.36
HOMO (ZINDO)	0.15	0.22	E^*_{max} (ZINDO)	1.06	1.76
LUMO (GW)	0.13	0.21	I^*_{max} (ZINDO)	0.07	0.12

Table 5.2.: Prediction error taken from the paper by Montavon et al. (2013c) for the simultaneous prediction of multiple molecular properties by a deep network on the QM7 dataset. Errors for energies, polarizabilities and intensity are given in eV, Å^3, and arbitrary units respectively. MAE denotes the mean absolute error and RMSE denotes the root mean square error.

for $\sigma = 1$ (the parameter that is used in the rest of the experiments) and $\sigma = 10$ (where Coulomb matrices are drawn from a much larger distribution). A high noise parameter tends to make learning harder, but ultimately leads to better models due to the extra inbuilt regularization.

So far, we have considered training with small molecules and a large fraction of samples in the training set. Our experiments have outlined the multiple tradeoffs in designing a neural network learning algorithm that is both computationally and statistically efficient. In particular, Figure 5.8 showed that using the whole dataset in a single model comes with a high computational overhead, in part, due to the larger dataset, but also, due to raised levels of learning noise that we discussed in the first part of this thesis. Practical applications will require models that handle much larger portions of the chemical compound space, possibly, billions of molecules. In these big models, we can expect the learning noise to be even higher. Therefore, we see as an important requirement to design methods that better deal with the increased level of learning noise, in order to maintain learnability at larger scales.

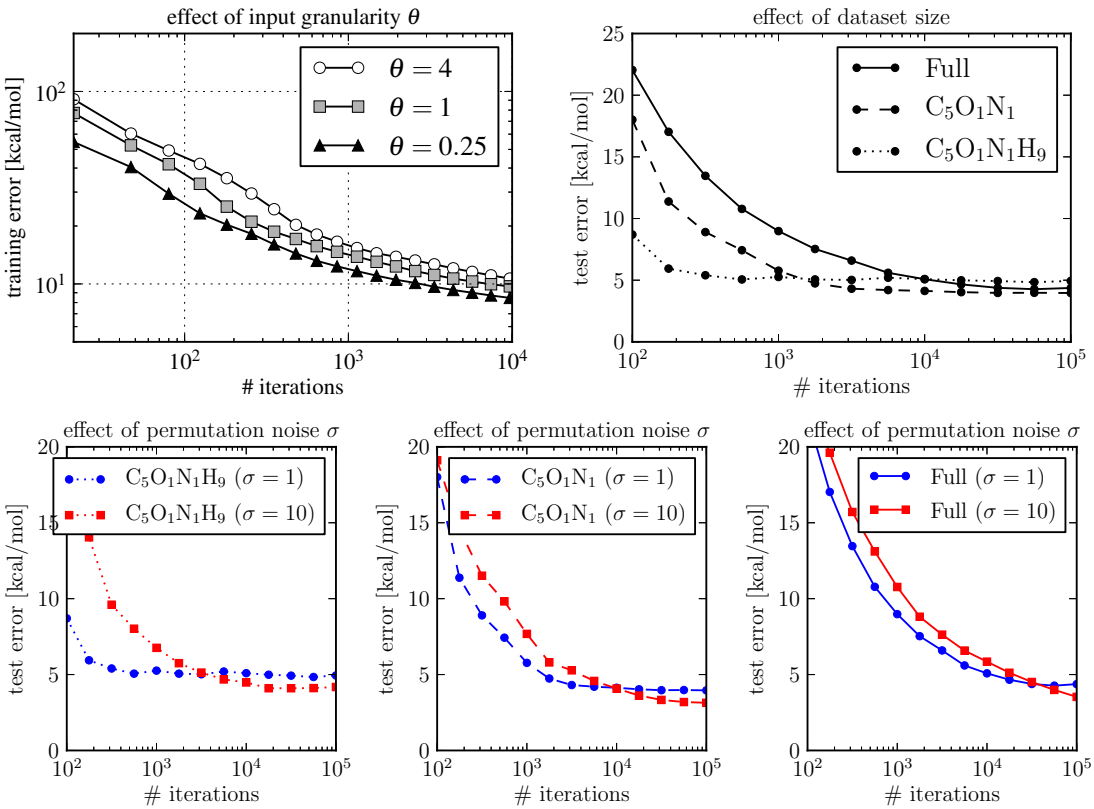


Figure 5.8.: *Top Left:* Plot by Montavon and Müller (2013) showing the effect of the binarization parameter θ on the speed of training. *Top Right:* Evolution of test error when trained on various subsets of data. *Bottom:* Effect of the noise parameter σ on the speed of learning and generalization error.

6. Conclusion and Discussion

The goal of this thesis was to understand what type of representations are built in deep networks. In Chapter 3 and 4, we formulated several hypotheses on the layer-wise evolution of representations in backpropagation networks and deep Boltzmann machines. These hypotheses were discovered from the careful observation of small systems of few interacting units. Then, we showed how such hypotheses can be understood in terms of noise of dimensionality within the kernel RDE framework of Braun et al. (2008). Here, the purpose of the kernel analysis was to serve as a proxy in order to test whether the hypotheses formulated in the context of small networks are also valid in larger networks composed of thousands of neurons and trained on real data.

Using this kernel-based analysis, we have demonstrated that backpropagation networks tend to build a hierarchy of increasingly task-relevant representations, and that similar hierarchy is expected to emerge in deep Boltzmann machines when data has some class-manifold structure. Then, we have studied the effect of learning noise on the representation at each layer. In backpropagation networks, we found that learning noise causes an early discrimination effect where most of the modeling becomes concentrated in the first layers of the network. In deep Boltzmann machines, we found that learning noise causes the complement mode of interaction between layers to be used disproportionately, and effectively excludes deeper layers from the training objective. This prevents the emergence of task-relevant representations in top layers.

Overall, the experiments highlight the capacity of deep networks to dynamically adapt their layer-wise structure to the different sources of noise in the learning algorithm. Such dynamic adaptation allows to minimize the training objective under various noise levels. However, such adaptability comes with the downside, that the original objective (i.e. learning the true function) is no longer optimized. Instead, the deep network optimizes the *proxy objective* of learning while being robust to noise. The additional difficulty with DBMs is that the proxy objective—learning a good generative model under noise—is now becoming very distant from the real objective.

Unfortunately, levels of noise only increase with more data. For example, for large datasets, it becomes impossible to wait to see all examples before updating the model, and stochastic gradient noise becomes therefore unavoidable. Similarly, parameterization noise also becomes unavoidable: While centering and other tricks might be sufficient for small datasets, for bigger problems, most regions of the input space will remain pathologically parameterized until the deep structure is discovered.

In order to learn functions $f \in \mathcal{F}_{\text{deep}}$, we see as a main requirement to control and reduce the multiple sources of learning noise. In our view, a lot progress has already been made in this direction: For example, the gradual introduction of more samples (curriculum learning, Bengio et al. 2009) is a possible approach to keep sustainable levels of noise throughout training. Structurally forcing the network to share features, for example, using a convolutional architecture (LeCun et al. 1998) or pretraining (Hinton et al. 2006), also goes into the direction of reducing learning noise: Gradients for different samples become correlated as feature sharing takes place, and stochastic gradient descent is in this case no longer noisy.

All these methods still require a human-engineered curriculum, some predefined structure or some generative assumption. Therefore, we see as a future work for deep learning to discover truly self-structuring mechanisms that are able to lower the learning noise in a generic manner. Moreover, these mechanisms should assume as little knowledge as possible about the function $f \in \mathcal{F}_{\text{deep}}$ to be learned.

A. Description of Datasets and Training Procedures

A.1. Description of Datasets

A.1.1. Datasets of Chapter 3 and 4

MNIST The MNIST handwritten character recognition dataset (LeCun et al. 1998) can be downloaded at `http://yann.lecun.com/exdb/mnist/` and consists of 60000 training and 10000 test handwritten digits. Each handwritten digit is represented as a grayscale image of 28×28 pixels and provided along with its label (a number between 0 and 9). The number of digits of each class is balanced. Some samples are shown in Figure A.1 (left).

Handwritten Characters The handwritten characters dataset (van der Maaten and Hinton 2008) can be downloaded at `http://homepage.tudelft.nl/19j49` and is composed of 40133 images of size 56×56 representing handwritten characters (A–Z uppercase and 1–9) along with their label. In order to reduce computational requirements, we rescale the images to 28×28. Handwritten characters are binarized and separated in two groups (Group 1: alphabetic characters, Group 2: digits). Alphabetic characters are split into 31616 training samples, 1000 validation samples and 5000 test samples. The remaining 2517 samples are handwritten digits and are used as a transfer task. Note that the number of samples of each class is not balanced. Some samples are shown in Figure A.1 (middle).

Spoken Words The spoken word recognition dataset is built from the TIMIT corpus (Garofolo et al. 1993) that can be obtained from `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1`. We use the word segmentation provided in the corpus to generate small spectrograms of size 28×28 for each word in the corpus. The spectrograms are obtained by evaluating the spectral power at 28 mel-frequencies and at 28 equidistant time steps. The mel-frequency coefficients are obtained by applying a 20 milliseconds Hamming window at each location, and mapping the resulting frequency power spectrum onto the mel scale. The data is whitened by applying a two-dimensional high-pass convolutional filter on each spectrogram. As the binary units of a DBM are stochastic and have maximum entropy at 0.5, we apply some dithering to the whitened spectrograms (same for all images) in order to equalize entropy for pixels of various intensities.

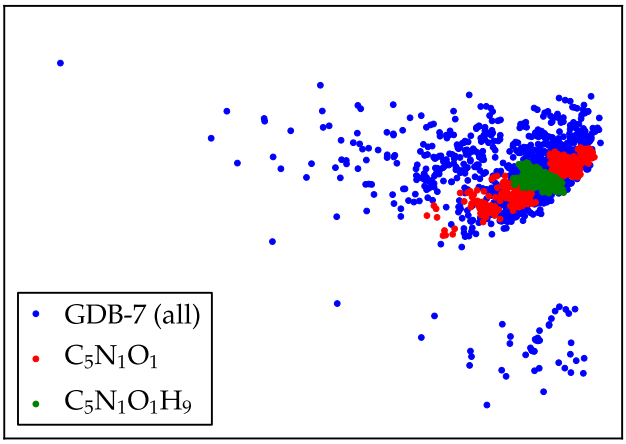
Figure A.1.: Samples from the datasets used in Chapter 3 and 4. The first dataset is composed of grayscale images of size 28×28 representing handwritten digits. The second dataset is composed of grayscale images of size 28×28 representing handwritten characters. The third dataset is composed of grayscale spectrograms of size 28×28 where 28 mel-frequency spectrum coefficients are measured at 28 evenly distributed time steps.

The TIMIT corpus is composed of several subsets (SA, SI and SX). The SA subset constitutes our core dataset and is composed of 21 distinct words spelled by 630 different speakers. The resulting 13230 samples are split into 7230 training samples, 1000 validation samples and 5000 test samples. We also build a transfer task of 5319 samples that consists of predicting 66 new words taken from the SI and SX subsets, each of them spelled by 50–150 different speakers. Some samples are shown in Figure A.1 (right).

For each of these datasets, the input distribution is believed to be highly non-Gaussian, in part, due to the special way translation and rotation are represented in pixel space. The object recognition nature of each of these problems also suggests that the input distribution is multimodal with each mode (or data manifold) representing a different class. Also, these datasets are believed to be well-modeled by a hierarchy: Handwritten characters or digits are composed of edges that can be combined. Similarly, spoken words are composed of syllables, that are themselves composed of phonemes and formants.

A.1.2. Dataset of Chapter 5

QM7 The QM7 dataset (Rupp et al. 2012) can be downloaded from `http://www.quantum-machine.org` and is based on the GDB-13 database (Blum and Reymond 2009). It consists of all 7165 molecules of up to 7 heavy atoms, along with their atomization energy E computed with hybrid density functional theory (PBE0) (Perdew et al. 1996). Molecules are represented as Coulomb matrices using Equation 5.2. A PCA visualization of the QM7 dataset is shown in Figure



class	# molecules
$C_5N_1O_1$	1020
$C_5N_1O_1H_3$	(20)
$C_5N_1O_1H_5$	(93)
$C_5N_1O_1H_7$	(239)
$C_5N_1O_1H_9$	(319)
$C_5N_1O_1H_{11}$	(287)
$C_5N_1O_1H_{13}$	(62)
$C_6N_0O_1$	837
$C_4N_2O_1$	837
$C_5N_2O_0$	756
$C_6N_1O_0$	733
$C_5N_0O_2$	651

Figure A.2.: *Left:* Two-dimensional PCA of the QM7 dataset used in Chapter 5 with constitutional isomers $C_5N_1O_1$ and $C_5N_1O_1H_9$ depicted in red and green. *Right*: Number of distinct molecules for various classes of isomers in the QM7 dataset.

A.2, with two particular groups of constitutional isomers highlighted in red and green.

Montavon et al. (2013c) extend the dataset by adding other molecular electronic properties computed with different methods. These additional properties are: averaged molecular polarizabilities α, HOMO/LUMO eigenvalues, ionization potential (IP), electron affinity (EA), first excitation energy E^*_{1st}, excitation frequency of maximal absorption E^*_{max}, and corresponding maximal absorption intensity I_{max}. Methods for computing these properties include hybrid density functional theory (PBE0) (Perdew et al. 1996), Zerner's intermediate neglect of differential overlap (ZINDO) (Zerner et al. 1980), self-consistent screening (SCS) (Tkatchenko et al. 2012), and Hedin's GW approximation (Hedin 1965).

A.2. Description of Training Procedures

A.2.1. Backpropagation Networks of Chapter 3

The backpropagation networks used in the experiments of Figure 3.3 and 3.7 are trained on the full MNIST dataset for 100000 iterations. The number of units at each layer is 784–100–50–25–10. Each hidden layer uses a centered sigmoid activation function and the output layer is a softmax layer (Bishop 1996). Output of sigmoids are continuously recentered by reparameterizing the network throughout training. Weights at each layer are initialized according to a normal distribution of scale $1/\sqrt{m}$ where m is the number of incoming connections. Weight updates are rescaled by the same factor and error derivatives are multiplied from one layer to another by $\sqrt{m/n}$, where m is the number of input nodes and n is the number

of output nodes at each layer. This ensures initially, that derivatives have similar scale at each layer. Each network is trained with stochastic gradient descent, with a minibatch of size 25, a learning rate $\eta = 0.1$ (with minibatch averaging), no momentum, and no weight averaging.

The backpropagation networks used in the experiments of Figure 3.8 are trained on 10000 samples of the MNIST dataset until a training error of 2.5% is reached. Each network is composed of one input layer, two hidden layers of 1600 units and one softmax output layer. The hyperbolic tangent nonlinearity is applied to each hidden layer. Weights at each layer are initialized according to a normal distribution of scale $1/\sqrt{m}$ where m is the number of incoming connections. The pretrained MLP (PMLP) is using the same architecture and training parameters as the MLP, but is initialized by a stack of binary RBMs trained in a greedy layer-wise fashion. The convolutional neural network (CNN) is made of two hidden layers of convolution/nonlinearity/subsampling. Convolution kernels have size 5×5. We use sigmoids nonlinearities. The pooling operation linearly resizes the feature maps by a factor 2. The number of feature maps at each layer is 100. All networks are trained with stochastic gradient descent with minibatch of size 20, a learning rate $\eta = 1.0$ (with minibatch averaging), no momentum, and no weight averaging.

A.2.2. DBMs and Stacked RBMs of Chapter 4

For the experiments of Section 4.2 and Section 4.3, we train DBMs of three hidden layers with 1600, 800, 400, 200 units at each layer and stacks of RBMs of four hidden layers with 1600, 800, 400, 200, 100 units at each layer. Units at each layer are centered (see Appendix B.1 for a detailed explanation). The initial bias in each hidden layer is set to $b = -1.3$ (where the sigmoid has the highest second-order derivative). The initial bias for visible units is set to $b = \text{sigm}^{-1}(\langle x \rangle_{\text{data}})$ in order to satisfy the centering condition. Weights are initialized to zero. For input of size N_x, we emulate a 1600-dimensional input layer by rescaling the first layer weight in the feed-forward pass by a factor $1600/N_x$ and keeping the backward pass unchanged. In the DBM, such asymmetry is also introduced in the upper layers. This reduces the amount of top-down feedback while allowing us to retain a relatively large number of units in the top layers.

All layers of the DBM are trained jointly without layer-wise pretraining. The stack of RBMs is also trained jointly by letting top layer dynamically adapt to the evolving representations of the previous layers. (We note that for stacked RBMs, there is no conceptual difference between joint training and greedy layer-wise training, as first layer training is in any case independent from training of upper layers. In practice, with appropriate learning rates at each layer, both methods produce similar results.) Unless stated otherwise, DBMs and stacked RBMs are trained with persistent contrastive divergence (Tieleman 2008). The size of the minibatch and the number of free particles are both set to 25.

We use stochastic updates of the alternate Gibbs sampler both for the estimation

of data and model statistics. In order to reduce top-layer noise, sufficient statistics are collected with structural sampling using a branching factor $b = 4$ for the DBM and $b = 2$ for the stack of RBMs (see Appendix B.2). Unless stated otherwise, learning rate is set to $\eta = 0.004$ (with minibatch averaging). Each model is trained for 240000 iterations.

A.2.3. Backpropagation Networks of Chapter 5

The neural network trained on the QM7 dataset is composed of 1800 input units, two hidden layers of 400 and 100 units respectively, and one output unit. Input to the neural network (random Coulomb matrices) is binarized with granularity $\theta = 1.0$. The network trained on the multitask variant of the dataset is composed of 2000 input units, two hidden layers of 800 units each, and 1000 output units. In this last network, the 14 outputs to be predicted are binarized with granularity $\theta = 0.25$.

Weights at each layer are initialized according to a normal distribution of mean 0 and standard deviation $1/\sqrt{m}$ where m is the number of input units to each layer. The error derivatives backpropagated from layer l to layer $l - 1$ are rescaled by a factor $\sqrt{m/n}$ where m is the number of input units and n is the number of output units at each layer. These scaling heuristics ensure that the units preactivations fall in the correct regime of the nonlinearity and that weights at each layer evolve at the correct speed. Input and output are normalized to have zero mean and unit variance. The learning rate is set to $\eta = 0.25$ (with minibatch averaging).

We train the networks for 250000 iterations of stochastic gradient descent where 25 samples are presented at each iteration. We use exponential moving average on the network parameters with averaging coefficients set such that approximately 10% of the training history is remembered. We use 90% of the training set for training and the remaining 10% for early stopping. On sorted Coulomb matrices, training the network takes less than one hour. On random Coulomb matrices, training the network takes up to one day. The prediction for out of sample molecules is obtained by averaging the prediction for 10 realizations of the random Coulomb matrix. The prediction of all 14 properties for a given molecule takes approximately 100 milliseconds.

B. Enhanced Training of DBMs

In this chapter, we describe the basic training procedure for deep Boltzmann machines and two enhancements that facilitate training in large networks. These enhancements are particularly useful when training DBMs without layer-wise pretraining. The first one is the *centering trick* (Montavon and Müller 2012) and is the analogue to centering units in backpropagation networks (LeCun et al. 1998, Schraudolph 1998). The second one is *structural sampling* and consists of sampling top layers more often than lower layers. Both techniques help to reduce learning noise and give rise to top-layer representations with better discriminative properties.

Let us consider a DBM with three layers x, y and z. Each layer has N_x, N_y and N_z units. In its canonical form, the energy of the DBM is defined as

$$E(x, y, z) = -x^\top W y - y^\top V z - x^\top a - y^\top b - z^\top c,$$

where $\theta = \{W, V, a, b, c\}$ are the parameters of the model. W and V are the weight matrices of size $N_x \times N_y$ and $N_y \times N_z$ respectively. Biases a, b and c have the same size as their associated layers x, y and z. Each state of the network is associated with a probability

$$p(x, y, z) = \frac{1}{Z(\theta)} e^{-E(x,y,z)},$$

that can be marginalized over y and z in order to obtain the input distribution $p(x)$. Learning a DBM from data, consists of finding model parameters θ that produce a distribution $p(x)$ that matches the data. The most common training objective is the data log-likelihood

$$J = \langle \log p(x) \rangle_{\text{data}},$$

where $\langle \cdot \rangle_{\text{data}}$ denotes the expectation operator with respect to the input data distribution. Maximizing this objective is usually achieved by stochastic gradient descent (Bottou 1991). The derivative of the objective J with respect to the model parameters has the simple form:

$$\begin{aligned} \frac{\partial J}{\partial W} &= \langle x y^\top \rangle_{\text{data}} - \langle x y^\top \rangle_{\text{model}}, \\ \frac{\partial J}{\partial V} &= \langle y z^\top \rangle_{\text{data}} - \langle y z^\top \rangle_{\text{model}}. \end{aligned} \tag{B.1}$$

The terms $\langle . \rangle_P$ appearing in the derivatives are called sufficient statistics and can be computed by collecting many samples from this distribution P. In particular,

data-dependent statistics $\langle\cdot\rangle_{\text{data}}$ are collected for the factored probability distribution $p(y, z|x) \cdot p_{\text{data}}(x)$, and model-dependent statistics $\langle\cdot\rangle_{\text{model}}$ are collected for the joint probability distribution $p(x, y, z)$.

Unbiased samples from these distributions can be obtained using a Gibbs sampler (Metropolis et al. 1953, Hastings 1970, Gelfand and Smith 1990). Gibbs sampling is a method for drawing samples from a multivariate probability distribution without having to compute the probability density function explicitly. It consists of sampling each variable of the joint distribution alternatively, conditioned on the current state of the previously sampled variables. An important property of Gibbs sampling is that it converges in distribution to the true probability distribution after repeated sampling. In the context of deep Boltzmann machines, we can exploit the special layered structure of the network to derive an efficient block-wise alternate Gibbs sampler (Salakhutdinov and Hinton 2009):

$$
\begin{aligned}
x &\sim \mathcal{B}(\text{sigm}(Wy + a)), \\
y &\sim \mathcal{B}(\text{sigm}(W^\top x + Vz + b)), \\
z &\sim \mathcal{B}(\text{sigm}(V^\top y + c)).
\end{aligned}
$$

Here, $x \sim \mathcal{B}(p)$ denotes the element-wise drawing of Bernoulli samples from a vector of probabilities p. In practice, convergence can be prohibitively slow for practical learning algorithms. Many techniques and approximations have been proposed in order to accelerate sampling. For example, Tieleman (2008) proposed to use persistent chains that sample from the model in background of the learning procedure. These chains can be shown to mix fast under the effect of learning. Desjardins et al. (2010) and Salakhutdinov (2010) proposed to use tempered transitions, where simpler high-temperature distributions are used to accelerate mixing. For data-dependent statistics, other techniques such as learning a feed-forward model of inference in parallel to training have been proposed (Salakhutdinov and Larochelle 2010).

B.1. The Centering Trick

Deep Boltzmann machines can be optimized more easily by centering the units at each layer. The energy of the Boltzmann machine is rewritten as

$$
\begin{aligned}
E(x, y, z) = &- (x - \alpha)^\top W(y - \beta) \\
&- (y - \beta)^\top V(z - \gamma) \\
&- (x - \alpha)^\top a - (y - \beta)^\top b - (z - \gamma)^\top c
\end{aligned}
$$

where α, β and γ are offset vectors that are used to center the activity of each unit in the network. The idea of centering Boltzmann machines was proposed in various contexts by Cho et al. (2011), Arnold et al. (2011), Tang and Sutskever (2011), and Montavon and Müller (2012). Based on this new energy function, the

gradient of the objective with respect to the model parameters becomes:

$$\frac{\partial J}{\partial W} = \langle (x-\alpha)(y-\beta)^\top \rangle_{\text{data}} - \langle (x-\alpha)(y-\beta)^\top \rangle_{\text{model}}, \tag{B.2}$$

$$\frac{\partial J}{\partial V} = \langle (y-\beta)(z-\gamma)^\top \rangle_{\text{data}} - \langle (y-\beta)(z-\gamma)^\top \rangle_{\text{model}}. \tag{B.3}$$

Similarly, the alternate Gibbs sampler is rewritten as:

$$\begin{aligned} x &\sim \mathcal{B}(\text{sigm}(W(y-\beta)+a)), \\ y &\sim \mathcal{B}(\text{sigm}(W^\top(x-\alpha)+V(z-\gamma)+b)), \\ z &\sim \mathcal{B}(\text{sigm}(V^\top(y-\beta)+c)). \end{aligned}$$

Throughout training, mean activations of x, y and z may diverge from zero. Therefore, it is important to regularly adapt the offset parameters. This should be done in conjunction with other parameters, in a way that leaves the energy function E constant (up to some constant factor), so that the probability distribution remains unchanged. Montavon and Müller (2012) propose the following reparameterization:

$$\begin{aligned} \alpha' &= \langle x \rangle & a' &= a + W(\langle y \rangle - \beta) \\ \beta' &= \langle y \rangle & b' &= b + W^\top(\langle x \rangle - \alpha) + V(\langle z \rangle - \gamma) \\ \gamma' &= \langle z \rangle & c' &= c + V^\top(\langle y \rangle - \beta) \end{aligned}$$

A simplified training procedure for a three layers centered deep Boltzmann machine is shown in Algorithm 3. The algorithm is taken from the paper by Montavon and Müller (2012) and uses persistent contrastive divergence with stochastic gradient descent. A sample code of the centered DBM is available at `http://gregoire.montavon.name/code/dbm.py`.

In order to better understand the effect of centering on the resulting optimization problem, we can analyze the properties of the Hessian of the learning objective with respect to the model parameters. For a fully connected Boltzmann machine, the Hessian $\boldsymbol{H}$ projected on some direction Δ in the parameter space can be expressed as

$$\begin{aligned} \boldsymbol{H}\Delta &= \frac{\partial}{\partial \Delta}\left(\frac{\partial}{\partial W}\langle \log p(x;W)\rangle_{\text{data}}\right) \\ &= \lim_{h\to 0}\frac{1}{h}\left(\frac{\partial}{\partial W}\langle \log p(x;W+h\Delta)\rangle_{\text{data}} - \frac{\partial}{\partial W}\langle \log p(x;W)\rangle_{\text{data}}\right) \\ &= \lim_{h\to 0}\frac{1}{h}\left((\langle \xi\xi^\top\rangle_{W+h\Delta,\text{data}} - \langle \xi\xi^\top\rangle_{W+h\Delta}) - (\langle \xi\xi^\top\rangle_{W,\text{data}} - \langle \xi\xi^\top\rangle_{W})\right) \\ &= \lim_{h\to 0}\frac{1}{h}\left(\langle \xi\xi^\top\rangle_{W+h\Delta,\text{data}} - \langle \xi\xi^\top\rangle_{W,\text{data}}\right) - \lim_{h\to 0}\frac{1}{h}\left(\langle \xi\xi^\top\rangle_{W+h\Delta} - \langle \xi\xi^\top\rangle_{W}\right), \end{aligned}$$

where ξ denotes the centered activity of each unit. Interestingly, we can observe from the last line that the projected Hessian can be decomposed into a data-dependent term and a data-independent term. If all units are visible, the data-dependent term is zero, showing that Hessian properties are intrinsic to the model

Algorithm 3 Centered deep Boltzmann machine

$W, V = 0, 0$
$a, b, c = \text{sigm}^{-1}(\langle x \rangle_{\text{data}}), b_0, c_0$
$\alpha, \beta, \gamma = \text{sigm}(a), \text{sigm}(b), \text{sigm}(c)$
initialize free particle $(x_m, y_m, z_m) = (\alpha, \beta, \gamma)$
loop
  initialize data particle $(x_d, y_d, z_d) = (\text{pick(data)}, \beta, \gamma)$
  loop
    $y_d \sim \mathcal{B}(\text{sigm}(W^\top(x_d - \alpha) + V(z_d - \gamma) + b))$
    $z_d \sim \mathcal{B}(\text{sigm}(V^\top(y_d - \beta) + c))$
  end loop
  $y_m \sim \mathcal{B}(\text{sigm}(W^\top(x_m - \alpha) + V(z_m - \gamma) + b))$
  $x_m \sim \mathcal{B}(\text{sigm}(W(y_m - \beta) + a))$
  $z_m \sim \mathcal{B}(\text{sigm}(V^\top(y_m - \beta) + c))$
  $W = W + \eta \cdot [(x_d - \alpha)(y_d - \beta)^\top - (x_m - \alpha)(y_m - \beta)^\top]$
  $V = V + \eta \cdot [(y_d - \beta)(z_d - \gamma)^\top - (y_m - \beta)(z_m - \gamma)^\top]$
  $a = a + \eta \cdot (x_d - x_m) + \nu \cdot W(y_d - \beta)$
  $b = b + \eta \cdot (y_d - y_m) + \nu \cdot W^\top(x_d - \alpha) + \nu \cdot V(z_d - \gamma)$
  $c = c + \eta \cdot (z_d - z_m) + \nu \cdot V^\top(y_d - \beta)$
  $\alpha = (1 - \nu) \cdot \alpha + \nu \cdot x_d$
  $\beta = (1 - \nu) \cdot \beta + \nu \cdot y_d$
  $\gamma = (1 - \nu) \cdot \gamma + \nu \cdot z_d$
end loop

rather than the data. Pearlmutter (1994) shows that the projected Hessian can be further reduced to

$$\boldsymbol{H}\Delta = \langle \xi\xi^\top \rangle_W \cdot \langle D \rangle_W - \langle \xi\xi^\top D \rangle_W \tag{B.4}$$

where $D = \frac{1}{2}\xi^\top \Delta\xi$. This last expression provides a numerically more stable estimate of the Hessian. Drawing a few directions Δ at random, we can build an approximation of $\boldsymbol{H}$ from which the condition number (ratio between highest and lowest eigenvalue) can be computed.

Montavon and Müller (2012) estimate the condition number for a simple fully connected Boltzmann machine of 50 units with zero weights and various bias and offset parameters. Random directions $\Delta_1, \ldots, \Delta_{100}$ are drawn in the 2500-dimensional parameter space in order to compute a low rank approximation of the Hessian:

$$\hat{\boldsymbol{H}} = \boldsymbol{H}(\Delta_1, \ldots, \Delta_{100}) = (\boldsymbol{H}\Delta_1, \ldots, \boldsymbol{H}\Delta_{100}).$$

Sufficient statistics involved in Equation B.4 are estimated from 1000 samples. Let $\lambda_1, \ldots, \lambda_{100}$ be the eigenvalues of the projected Hessian. The condition number λ_1/λ_{100} is shown below for various b and β. Centered DBMs are marked in bold.

λ_1/λ_{100}	$b=2$	$b=0$	$b=-2$
$\beta=\text{sigm}(2)$	**1.98**	12.65	51.42
$\beta=\text{sigm}(0)$	22.97	**1.82**	22.20
$\beta=\text{sigm}(-2)$	52.72	13.40	**1.94**

We can observe that the condition number in a centered DBM is better by an order of magnitude than in a non-centered one.

B.2. Structural Sampling

In deep Boltzmann machines, we would like to reduce the bias and variance of the estimated sufficient statistics entering in the computation of the parameter update. Low bias is particularly important for modeling data-dependent statistics: Let ξ denote the state of the hidden units, driven by a bias parameter b that aggregates local bias and contribution from other units. Suppose that the data statistics $\langle\xi\rangle_{\text{data}}$ are incorrectly sampled as

$$\begin{aligned}\widehat{\langle\xi\rangle}_{b,\text{data}} &= \langle\xi\rangle_{b,\text{data}} + \delta_{\text{data}} + \varepsilon_{\text{data}},\\ \widehat{\langle\xi\rangle}_{b,\text{model}} &= \langle\xi\rangle_{b,\text{model}} + \delta_{\text{model}} + \varepsilon_{\text{model}},\end{aligned}$$

where we decompose the estimation as the true expectation plus the bias δ and variance ε of the estimator. The sampling bias may arise, for example, from a not fully converged Gibbs sampler. The parameter update takes the form

$$\begin{aligned}\widehat{\Delta b} &= \eta\cdot(\langle\xi\rangle_{b,\text{data}} + \delta_{\text{data}} + \varepsilon_{\text{data}} - \langle\xi\rangle_{b,\text{model}} - \delta_{\text{model}} - \varepsilon_{\text{model}})\\ &= \Delta b + \eta\cdot(\delta_{\text{data}} + \varepsilon_{\text{data}} - \delta_{\text{model}} - \varepsilon_{\text{model}}).\end{aligned}$$

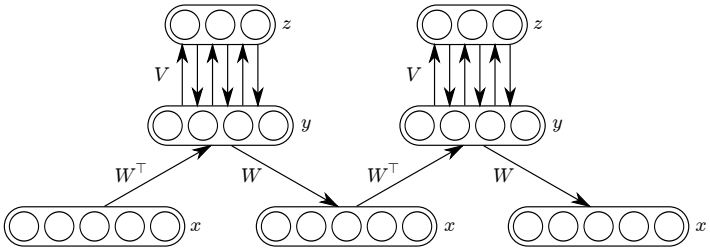
The data-driven bias is particularly problematic: Until the model picks up with the data, there will be a systematic bias in the parameter update, that will cumulate over time. In practice, bias can be removed by using a sufficient number of iterations of the Gibbs sampler. The importance of producing data-dependent statistics from a converged Gibbs sampler, or an accurate approximation of it, was already demonstrated by Salakhutdinov and Larochelle (2010).

Similarly, low variance is also important for modeling both data-dependent and data-independent statistics. The harmful effect of variance is that it might construct random pairwise interactions that affect the layer-wise organization of the model. The model hardly recovers from these random interactions, as they are immediately perceived as "ground truth" when computing data-dependent statistics in the next iterations.

Generally, we would like to reduce the variance of the sufficient statistics $\langle\cdot\rangle_{\text{data}}$ and $\langle\cdot\rangle_{\text{model}}$ where it is the most harmful and most susceptible to occur. Such variance is particularly susceptible to occur in top layers, because this is where the posterior distribution has the highest entropy H. In a three layer DBM, we

Algorithm 4 Structural Sampling

function STRUCT(l)
  SAMPLE(l)
  if $l < L$ **then**
    for $i = 1 \dots b$ **do**
      STRUCT($l + 1$)
      SAMPLE(l)
    end for
  end if
end function



typically have

$$\frac{1}{N_z} H(z|x) > \frac{1}{N_y} H(y|x) > \frac{1}{N_x} H(x|x)$$

An intuition for this is that deep Boltzmann machines can be seen as a noisy channel where each layer adds some noise to the signal x through sampling. For reasonable distributions, we expect that the same also holds true for the sufficient statistics at each layer,

$$\frac{1}{N_y N_z} H(yz^\top|x) > \frac{1}{N_x N_y} H(xy^\top|x),$$

and that top-layer statistics are therefore particularly susceptible to exhibit sampling noise.

We propose an algorithm that oversamples top layers by using a layer-wise recursive sampling procedure that we call *structural sampling*. Such sampling procedure both aims to reduce sampling noise and sampling bias, and can be shown to have a constant computational overhead compared to the standard sampling procedure. The recursive procedure is shown in Algorithm 4. We require that the number of units at each layer of the DBM decreases with layer depth as $N_x c^{-l}$ where N_x is the number of input dimensions and $c^2 > b$. For example, a two-fold reduction of the number of units at each layer combined with a branching factor $b = 3$ in the recursive procedure satisfies this inequality. The overall amount of computation for one iteration of the recursive sampling procedure in an infinitely deep network is:

$$\begin{aligned} \text{Cost} &= \sum_{l=0}^{\infty} A \cdot (N_x c^{-l}) \cdot (N_x c^{-(l+1)}) \cdot b^l \\ &= \frac{A \cdot N_x^2}{c} \sum_{l=0}^{\infty} \left(\frac{b}{c^2} \right)^l = O(N_x^2). \end{aligned}$$

The variable A encompasses the cost per parameter of the various matrix multiplications used for forward sampling, backward sampling, and collection of sufficient

statistics. On simple sequential computers, such cost per parameter is constant. Overall, the computational cost scales quadratically with the number of input dimensions.

C. Building Local Reliability Estimates with Kernels

It is an important requirement in many applications to integrate predictive uncertainty into the model. Such reliability estimates may help to implement risk averse strategies, that are often needed in applications such as automated medical diagnosis, quantitative trading or computer security. Models of predictive uncertainty may also form the basis for more refined active learning strategies.

In this section, we first review some simple and well-established models of predictive uncertainty: Gaussian processes and Parzen variance estimators. Then, we introduce a localized extension of the relevant dimensionality estimates of Braun et al. (2008). The localized extension is proposed by Montavon et al. (2013a) and takes into account both noise and dimensionality in order to produce error bars. Finally we show in which circumstances these enhanced error bars are more appropriate, and compare the performance of each method on several classification and regression tasks.

C.1. Gaussian Processes, Parzen Windows, and Local RDE

A popular algorithm for producing localized error bars is the Gaussian process (Williams and Rasmussen 1996, Rasmussen 2004). The predictive model $y(x)$ is given a prior distribution (Gaussian process) with covariance

$$\mathrm{Cov}(y(x), y(x')) = \begin{cases} k(x,x') + \lambda^2 & x = x' \\ k(x,x') & \text{else} \end{cases}$$

where λ models the intrinsic noise in data and can also be seen as a regularization parameter. Given n input samples represented as a matrix X of size $n \times d$, and its associated kernel Gram matrix K of size $n \times n$, we can compute the local mean and variance of the posterior (also a Gaussian process) as

$$\begin{aligned} \mathrm{E}^{\mathrm{GP}}[\hat{y}(x)] &= k(x,X) \cdot (K + \lambda^2 I)^{-1} Y \\ \mathrm{Var}^{\mathrm{GP}}[\hat{y}(x)] &= k(x,x) - k(x,X) \cdot (K + \lambda^2 I)^{-1} \cdot k(X,x). \end{aligned}$$

where Y is a vector of labels of size n. It is instructive to discuss the behavior of error bars close and far away from the data: Near the data, assuming $\lambda = 0$, the second term of the variance estimate collapses to one, thus, leading to a variance

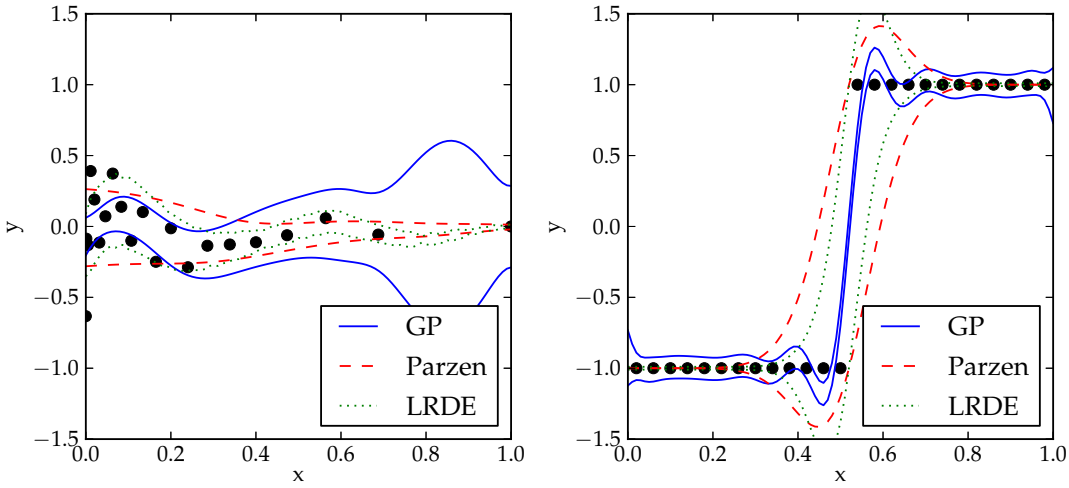


Figure C.1.: Local error bars obtained with a Gaussian process model, a Parzen window estimator of variance and an estimator based on local RDE.

of zero. Far away from the data, the terms $k(x, X)$ and $k(X, x)$ are zero and the variance estimate becomes 1. Therefore, the Gaussian process tends to produce error bars that are large outside the data and small near the data. It is important to note, that the label is not involved in the computation of error bars. In Figure C.1, we can see on the left, that when the label noise is correlated to the data density, the Gaussian process produces error bars that are opposite to what it should do. On the right, we observe that the Gaussian process is clearly not able to raise the level of uncertainty near the cliff.

Estimating local variance using Parzen windows (Rosenblatt 1956, Parzen 1962) is a different approach to produce error bars: Unlike Gaussian processes, Parzen windows are incorporating local label variance in the error estimate. This is particularly important for applications where data is heteroskedastic (i.e. noise varies as a function of the position in the input space). Densely populated regions of the input space with high level of label noise should be modeled with large error bars. Conversely, sparsely populated area with low label noise or low variations of $y(x)$ are easy to predict. Assuming a dataset $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, we can compute the mean and variance estimator at position x as

$$\mathrm{E}^{\mathrm{PA}}[\hat{y}(x)] = \frac{\sum_{i=1}^{n} y(x_i) g(x, x_i)}{\sum_{i=1}^{n} g(x, x_i)},$$
$$\mathrm{Var}^{\mathrm{PA}}[\hat{y}(x)] = \frac{\sum_{i=1}^{n} (y(x_i) - \mu(x))^2 g(x, x_i)}{\sum_{i=1}^{n} g(x, x_i)},$$

where $\mu(x) = \mathrm{E}^{\mathrm{PA}}[\hat{y}(x)]$ and $g(x, \cdot)$ is a Parzen window. Here, the noise estimate is clearly dependent on the local variability of y. Error bars produced by Parzen windows are shown in Figure C.1. The Parzen window estimator is better adapting

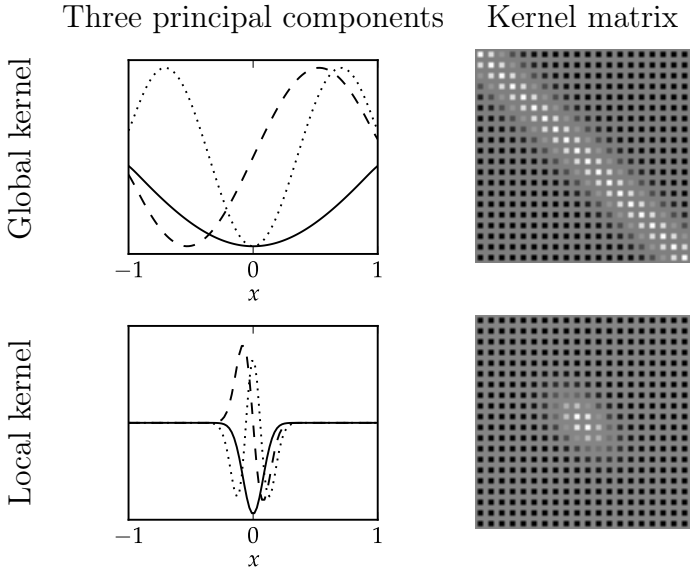


Figure C.2.: Image taken from the paper by Montavon et al. (2013a) showing a local and global RBF kernel along with its three leading kernel principal components. The solid, dashed and dotted lines represent the first, second and third components and have increasing frequency in the input space.

locally to the label noise. However, on the cliff problem, it confuses the change of sign signal as noise and produces a too smooth signal with pessimistic estimates of noise. Indeed, this model does not differentiate high noise from highly varying signal.

A better solution is to build a local model of noise and dimensionality. This can be done within the RDE framework of Braun et al. (2008) described in Chapter 2. For this, Montavon et al. (2013a) define the localized kernel k_ξ as:

$$k_\xi(x, x') = k(\xi, x) \cdot k(x, x') \cdot k(x', \xi)$$

where $k(\xi, x)$ and $k(x', \xi)$ act as Parzen windows that localize the kernel around the position ξ. Example of Gram matrices obtained with a global and local kernel are shown in Figure C.2. Here, we consider a simple Gaussian kernel and a uniform input distribution $x \sim \mathcal{U}(-1, 1)$. Note that in order to compute the spectrum coefficients $\{Z_i^2\}$ or the error residuals $e(d)$, the label vector should first be centered:

$$y_\xi(x) = (y(x) - \mu_\xi) \cdot k(\xi, x),$$

where $\mu_\xi = \sum_x y(x)k(\xi, x) / \sum_x k(\xi, x)$. Let $\sum_{x,x'} k_\xi(x, x')$ be the sum of entries in the localized Gram matrix. For Gaussian kernels, we would like the sum of elements in the local kernel to be a constant value c for each location ξ. The appropriate scale parameter σ can be found easily by grid search or by iterating the series $\sigma^{(i+1)} = \sigma^{(i)} + \gamma(c - \sum_{x,x'} k_\xi(x, x'))$ for appropriate step size γ. In the

case of an RBF kernel where distance is computed as an L^p norm, the product of kernels has an interpretation as a distance in a higher dimensional space:

$$\begin{aligned} k_\xi(x, x') &= \exp\big(-\frac{||\xi - x||_p^p}{\sigma^p}\big) \cdot \exp\big(-\frac{||x - x'||_p^p}{\sigma^p}\big) \cdot \exp\big(-\frac{||x' - \xi||_p^p}{\sigma^p}\big) \\ &= \exp\big(-\frac{||\xi - x||_p^p + ||x - x'||_p^p + ||x' - \xi||_p^p}{\sigma^p}\big) \\ &= \exp\big(-\frac{||[x\ x\ \xi] - [\xi\ x'\ x']||_p^p}{\sigma^p}\big) \end{aligned}$$

In the last line, the terms $[x\ x\ \xi]$ and $[\xi\ x'\ x']$ denote the concatenation of the multiple vectors inside the brackets. This higher-dimensional mapping has a geometric interpretation when the input space is one-dimensional. For fixed ξ, the three-dimensional parametric curves $[x\ x\ \xi]$ and $[\xi\ x'\ x']$ intersect at $[\xi\ \xi\ \xi]$ with an angle of 60°. This shows that the expanded distance between x and x' is generally correlated to the one-dimensional distance $x - x'$ (because the angle is acute), but can only be fully minimized if located near ξ.

Estimates of predictive uncertainty can now be obtained from the localized kernel. Using the procedure described at the beginning of Section 2.2 but replacing the global kernel $k(x, x')$ by the local kernel $k_\xi(x, x')$, we first compute the eigenvectors $\{u_i\}_{i=1}^n$ associated with the Gram matrix K_ξ. Then, the localized spectral coefficients $\{Z_i^2\}_{i=1}^n$ are computed as

$$Z_i^2 = (u_i^\top (y - \mu_\xi))^2,$$

where $y - \mu_\xi$ is the vector of labels y centered near ξ. These localized spectral coefficients allow us to build a model of mean and variance at every location ξ of the input space

$$\begin{aligned} \mathrm{E}^{\mathrm{RDE}}[y(\xi)] &= \mu_\xi + \sum_{i=1}^{d} u_i u_i^\top (y - \mu_\xi), \\ \mathrm{Var}^{\mathrm{RDE}}[y(\xi)] &= \frac{\sum_{i=d+1}^{n} Z_i^2}{\sum_{i=1}^{n} k_\xi(x_i, \xi)^2}. \end{aligned}$$

The number of dimensions d should in theory be chosen such that it spans the signal component and discards the rest. The small amount of data intervening in the localized kernel can make such estimation inaccurate. In general, we find empirically, that a small hardcoded value for d, typically $d = 2$ produces reasonable models of predictive mean and variance. Figure C.1 shows the error bars produced by the local RDE model. On the left, the error bars are noisy, suggesting a more unstable behavior than the simple Parzen window. On the other hand, on the right, the local RDE model is able to detect the signal component on the cliff and produces tighter error bars near the boundary.

A last aspect of local RDE is its computational cost. In its naive implementation, it requires the computation of a local Gram matrix and its eigenvalue

decomposition for every test point. Given that such computation has a cost of $O(n^3)$, and assuming that the number of training points and the number of test points are approximately the same, it brings the total computation to $O(n^4)$, which is intractable for more than a few hundred samples. However, for larger datasets, we can exploit the fact that the kernel is both localized and similar for two neighboring locations, in order to improve computational efficiency: First, only the non-zero subregions of the kernel needs to be computed and eigendecomposed. Second, when implemented in an iterative manner (Kim et al. 2003, Günter et al. 2007), the computation of leading principal components can be reused from one location to another.

C.2. Results and Discussion

We test the performance of each method (Gaussian process, Parzen variance estimator, and local RDE) on real data. We consider three datasets: the MNIST dataset, the original version of the UCI Wisconsin breast cancer dataset (Mangasarian and Wolberg 1990) and two variants of the QM7 quantum chemistry dataset (see Appendix A.1). Due to the high computational requirements of the naive implementation of local RDE, we restrict ourselves to subsets of a few hundred samples. On the MNIST dataset, we use 500 training and test samples (only digits "1" and "7"). On the breast cancer dataset, we use 341 training and test samples. For the quantum chemistry dataset, we extract 250 training and test samples corresponding to the smallest molecules in the QM7 dataset, and the task is to predict the corresponding first excitation energy $E^\star_{\text{1st}}$. We consider two representations of molecules: The first one (QM7-noisy) is based only on the 12 largest elements of the Coulomb matrix sorted element-wise. The second one (QM7-clean) is based on all elements of the Coulomb matrix.

Results are given in Figure C.3. The true vs. predictive errors are shown as a scatter plot. Here, the relation between both quantities (true error and predictive error) should be as positively correlated as possible. In order to obtain quantifiable results, we also compute precision-recall curves in the classification setting, and we use $f(x) = 0$ as an implicit classification boundary. For the regression case, we sort samples from the most trusted to the least trusted and plot the cumulative root mean square error (RMSE).

Figure C.3 (left) shows the results for the two classification tasks. The nature of the prediction problem is similar to the example of Figure C.1 (right) where two plateaus are separated by a cliff—here, a classification boundary. We observe that the local RDE (LRDE) performs best on both datasets. LRDE is able to adjust its estimate of noise and dimensionality near the classification boundary and therefore to produce better estimates of local variance. On the other hand, the Gaussian process is not appropriate for this task. We also compute p-values[1] that take both predictive mean and variance in the estimate of reliability. Interestingly,

[1]See Montavon et al. (2013a) for more details on how these p-values are computed.

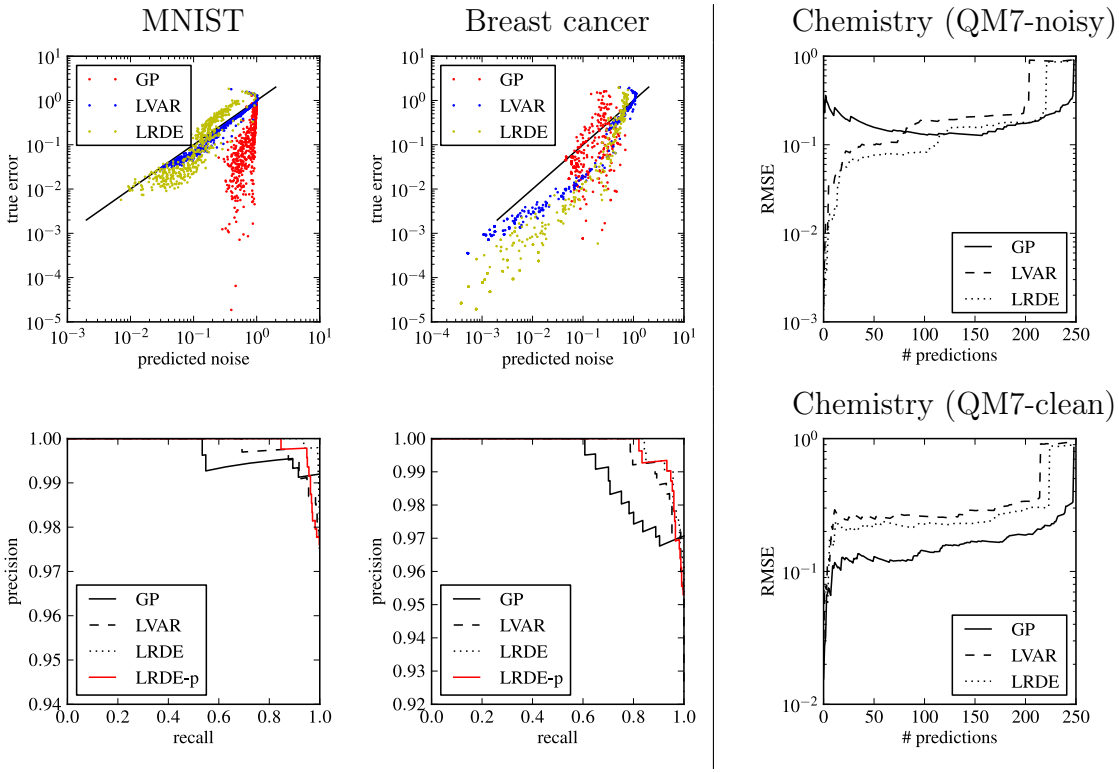


Figure C.3.: *Left:* Plots by Montavon et al. (2013a) showing the predictive vs. true error, and precision-recall curves for the MNIST and breast cancer classification datasets. The LRDE method is performing best in both cases, producing a few highly trusted samples. *Right:* Plots by Montavon et al. (2013a) showing the average error when sorting data from the most to the least trusted sample. In the noisy case, LRDE performs best. However, with the full representation (QM7-clean), Gaussian processes become the method of choice.

these p-values—based here on some Gaussianity assumption—do not improve the precision-recall curves. This suggests that Gaussianity is a poor assumption in these problems and that better statistical tests should be devised.

Figure C.3 (right), shows the regression error for the two variants of the quantum chemistry dataset (noisy and clean). As expected, on the noisy variant of QM7, the local RDE method is able to detect high level of noise when multiple molecules with different molecular properties collapse onto the same point in the feature space. This translates into lower mean square error for highly trusted samples. On the other hand, on the clean variant of QM7, where all input information is available, Gaussian processes perform the best. Indeed, the main source of noise comes now from the uncertainty in the input space rather than the label noise.

Our study has demonstrated within the kernel framework the importance of taking into account both noise and dimensionality in order to compute local reliability estimates. In the first part of this thesis, we described how the multiple

layers of representation in a deep network can be understood as a feature space, or a kernel. Thus, it is in principle possible to apply these local reliability estimates at every layer of a deep network. This may inform the learning algorithm on regions of high noise or regions of high dimensionality in the feature space. This additional information could be in turn exploited in order to dynamically adapt the learning strategy.

Bibliography

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4277–4280, 2012.

Ludovic Arnold and Yann Ollivier. Layer-wise learning of deep generative models. *CoRR*, abs/1212.1524, 2012.

Ludovic Arnold, Anne Auger, Nikolaus Hansen, and Yann Ollivier. Information-geometric optimization algorithms: A unifying picture via invariance principles. *CoRR*, abs/1106.3708, 2011.

Roman M. Balabin and Ekaterina I. Lomakina. Neural network approach to quantum-chemistry data: Accurate prediction of density functional theory energies. *The Journal of Chemical Physics*, 131(7):074104, 2009.

Jorg Behler. Neural network potential-energy surfaces in chemistry: a tool for large-scale simulations. *Physical Chemistry Chemical Physics*, 13:17930–17955, 2011.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In Léon Bottou, Olivier Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, 2007.

Yoshua Bengio and Eric Thibodeau-Laufer. Deep generative stochastic networks trainable by backprop. *CoRR*, abs/1306.1091, 2013.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19*, volume 19, pages 153–160, 2007.

Yoshua Bengio, Jerome Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. Technical Report 1330, Département d'informatique et recherche opérationnelle, Université de Montréal, 2009.

Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In *Proceedings of the 30th International Conference on Machine Learning*. ACM, 2013a.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. *CoRR*, abs/1305.6663, 2013b.

Christopher M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1996.

Lorenz C. Blum and Jean-Louis Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *Journal of the American Chemical Society*, 131(25):8732–8733, 2009.

Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2.

Léon Bottou. Stochastic gradient tricks. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks, Tricks of the Trade, Reloaded*, Lecture Notes in Computer Science (LNCS 7700), pages 430–445. Springer, 2012.

Mikio L. Braun. Accurate bounds for the eigenvalues of the kernel matrix. *Journal of Machine Learning Research*, 7:2303–2328, Nov 2006.

Mikio L. Braun, Joachim Buhmann, and Klaus-Robert Müller. On relevant dimensions in kernel feature spaces. *Journal of Machine Learning Research*, 9: 1875–1908, Aug 2008.

Robert Burbidge, Matthew W. B. Trotter, Bernard F. Buxton, and Sean B. Holden. Drug design by machine learning: Support vector machines for pharmaceutical data analysis. *Computers & Chemistry*, 26(1):5–14, 2002.

Charles F. Cadieu, Ha Hong, Dan Yamins, Nicolas Pinto, Najib J. Majaj, and James J. DiCarlo. The neural representation benchmark and its evaluation on brain and machine. *CoRR*, abs/1301.3530, 2013.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, July 1997.

KyungHyun Cho, Tapani Raiko, and Alexander Ilin. Enhanced gradient and adaptive learning rate for training restricted Boltzmann machines. In *Proceedings of the 28th International Conference on Machine Learning*, pages 105–112, June 2011.

Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *CoRR*, abs/1003.0358, 2010.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Large vocabulary continuous speech recognition with context-dependent DBN-HMMS. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 4688–4691. IEEE, 2011.

Guillaume Desjardins, Aaron C. Courville, Yoshua Bengio, Pascal Vincent, and Olivier Delalleau. Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines. *Journal of Machine Learning Research - Proceedings Track*, pages 145–152, 2010.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, March 2010.

John S. Garofolo, Lori F. Lamel, William M. Fisher, Jonathan G. Fiscus, David S. Pallett, Nancy L. Dahlgren, and Victor Zue. TIMIT acoustic phonetic continuous speech corpus, 1993.

Alan E. Gelfand and Adrian F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85 (410):398–409, 1990.

Zoubin Ghahramani. Unsupervised learning. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 72–112. Springer, 2003.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. *Journal of Machine Learning Research - Proceedings Track*, 15:315–323, April 2011.

Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio. Joint training deep Boltzmann machines for classification. *CoRR*, abs/1301.3568, 2013.

Simon Günter, Nicol N. Schraudolph, and S. V. N. Vishwanathan. Fast iterative kernel principal component analysis. *Journal of Machine Learning Research*, 8: 1893–1918, December 2007.

Corwin Hansch, Albert Leo, and D. H. Hoekman. *Exploring QSAR*. American Chemical Society, 1995.

Katja Hansen, Grégoire Montavon, Franziska Biegler, Siamac Fazli, Matthias Rupp, Matthias Scheffler, O. Anatole von Lilienfeld, Alexandre Tkatchenko, and Klaus-Robert Müller. Assessment and validation of machine learning methods for predicting molecular atomization energies. *Journal of Chemical Theory and Computation*, 9(8):3404–3419, 2013.

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

Lars Hedin. New method for calculating the one-particle Green's function with application to the electron-gas problem. *Physical Review*, 139:A796–A823, Aug 1965.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, August 2002.

Geoffrey E. Hinton. A practical guide to training restricted Boltzmann machines. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer Berlin Heidelberg, 2nd edition, 2012.

Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

Geoffrey E. Hinton and Terrence J. Sejnowski. Learning and relearning in Boltzmann machines. In David E. Rumelhart, James L. McClelland, and Corporate PDP Research Group, editors, *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, pages 282–317. MIT Press, Cambridge, MA, USA, 1986.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

Pierre C. Hohenberg and Walter Kohn. Inhomogeneous electron gas. *Physical Review Online Archive (Prola)*, 136(3B):B864–B871, November 1964.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148:574–591, October 1959.

Yunho Jeon and Chong-Ho Choi. Thermometer coding for multilayer perceptron learning on continuous mapping problems. In *International Joint Conference on Neural Networks*, volume 3, pages 1685–1690, 1999.

Kwang In Kim, Matthias O. Franz, and Bernhard Schölkopf. Kernel Hebbian algorithm for iterative kernel principal component analysis. Technical Report 109, MPI f. biologische Kybernetik, Tuebingen, 6 2003.

Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. ImageNet classification with deep convolutional neural networks. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012.

Yann LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Zurich, Switzerland, 1989. Elsevier. an extended version was published as a technical report of the University of Toronto.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Leon Bottou, Geneviève B. Orr, and Klaus-Robert Müller. Efficient backprop. In Geneviève B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer Berlin Heidelberg, 1998.

Sönke Lorenz, Axel Groß, and Matthias Scheffler. Representing high-dimensional potential-energy surfaces for reactions at surfaces by neural networks. *Chemical Physics Letters*, 395(4–6):210–215, 2004.

Olvi L. Mangasarian and William H. Wolberg. Cancer diagnosis via linear programming. In *SIAM News*, volume 23, pages 1–18, 1990.

James Martens and Ilya Sutskever. Training deep and recurrent networks with Hessian-free optimization. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 479–535. Springer Berlin Heidelberg, 2nd edition, 2012.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

Grégoire Montavon and Klaus-Robert Müller. *Deep Boltzmann Machines and the Centering Trick*, volume 7700 of *LNCS*, chapter 25, pages 621–637. Springer, 2nd edition, 2012.

Grégoire Montavon and Klaus-Robert Müller. Neural networks for computational chemistry: Pitfalls and recommendations. *MRS Online Proceedings Library*, 1523, 2013.

Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. Layer-wise analysis of deep networks with Gaussian kernels. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1678–1686, 2010.

Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. Kernel analysis of deep networks. *Journal of Machine Learning Research*, 10:2579–2597, September 2011.

Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. Deep Boltzmann machines as feed-forward hierarchies. *Journal of Machine Learning Research - Proceedings Track*, 22:798–804, 2012a.

Grégoire Montavon, Katja Hansen, Siamac Fazli, Matthias Rupp, Franziska Biegler, Andreas Ziehe, Alexandre Tkatchenko, O. Anatole von Lilienfeld, and Klaus-Robert Müller. Learning invariant representations of molecules for atomization energy prediction. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 449–457, 2012b.

Grégoire Montavon, Mikio L. Braun, Tammo Krueger, and Klaus-Robert Müller. Analyzing local structure in kernel-based learning: Explanation, complexity, and reliability assessment. *Signal Processing Magazine, IEEE*, 30(4):62–74, 2013a.

Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. A study of representations in deep Boltzmann machines. under review, 2013b.

Grégoire Montavon, Matthias Rupp, Vivekanand Gobre, Alvaro Vazquez-Mayagoitia, Katja Hansen, Alexandre Tkatchenko, Klaus-Robert Müller, and O. Anatole von Lilienfeld. Machine learning of molecular electronic properties in chemical compound space. *New Journal of Physics*, 15(9):095003, 2013c.

Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, March 2001.

Ury Naftaly, Nathan Intrator, and David Horn. Optimal ensemble averaging of neural networks. *Network: Computation in Neural Systems*, 8(3):283–296, 1997.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010.

Radford M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113, July 1992.

Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

John P. Perdew, Matthias Ernzerhof, and Kieron Burke. Rationale for mixing exact exchange with density functional approximations. *The Journal of Chemical Physics*, 105(22):9982–9985, 1996.

Carl Edward Rasmussen. Gaussian processes in machine learning. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 63–71. Springer, 2004.

Salah Rifai, Yann N. Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. The manifold tangent classifier. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2294–2302, 2011a.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, pages 833–840. Omnipress, 2011b.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pages 832–837, 1956.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Matthias Rupp, Alexander Tkatchenko, Klaus-Robert Müller, and O. Anatole von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Letters*, 108:058301, 2012.

Ruslan Salakhutdinov. Learning deep Boltzmann machines using adaptive MCMC. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning*, pages 943–950. Omnipress, 2010.

Ruslan Salakhutdinov and Geoffrey E. Hinton. Deep Boltzmann machines. *Journal of Machine Learning Research - Proceedings Track*, 5:448–455, 2009.

Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep Boltzmann machines. *Journal of Machine Learning Research - Proceedings Track*, 9:693–700, 2010.

Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Proceedings of 25th the International Conference on Machine Learning*, pages 872–879. ACM, 2008.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5): 1299–1319, 1998.

Nicol N. Schraudolph. Centering neural network gradient factors. In Geneviève B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 207–226. Springer Berlin Heidelberg, 1998.

Timon Schroeter, Anton Schwaighofer, Sebastian Mika, Antonius Ter Laak, Detlev Suelzle, Ursula Ganzer, Nikolaus Heinrich, and Klaus-Robert Müller. Estimating the domain of applicability for machine learning QSAR models: A study on aqueous solubility of drug discovery molecules. *Journal of Computer Aided Molecular Design - special issue on "ADME and Physical Properties"*, 21(9): 485–498, 2007.

Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 994–1000, 2005.

Patrice Simard, Yann LeCun, John S. Denker, and Bernard Victorri. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In Geneviève B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 235–269, London, UK, UK, 1998. Springer-Verlag.

John C. Snyder, Matthias Rupp, Katja Hansen, Klaus-Robert Müller, and Kieron Burke. Finding density functionals with machine learning. *Physical Review Letters*, 108(25):253002, Jun 2012.

John C. Snyder, Matthias Rupp, Katja Hansen, Leo Blooston, Klaus-Robert Müller, and Kieron Burke. Orbital-free bond breaking via machine learning. *submitted to Journal of Chemical Physics*, 2013.

Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep Boltzmann machines. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2231–2239, 2012.

Yichuan Tang and Ilya Sutskever. Data normalization in the learning of restricted Boltzmann machines. Technical Report UTML-TR-11-2, Department of Computer Science, University of Toronto, 2011.

Yee Whye Teh and Geoffrey E. Hinton. Rate-coded restricted Boltzmann machines for face recognition. In *Advances in Neural Information Processing Systems 13*, pages 908–914. MIT Press, 2001.

Tijmen Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1064–1071, 2008.

Alexandre Tkatchenko, Robert A. DiStasio, Roberto Car, and Matthias Scheffler. Accurate and efficient method for many-body van der Waals interactions. *Physical Review Letters*, 108:236402, Jun 2012.

Paul E. Utgoff and David J. Stracuzzi. Many-layered learning. *Neural Computation*, 14:2497–2529, 2002.

Laurens van der Maaten and Geoffrey E. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

Andre Wibisono, Jake Bouvrie, Lorenzo Rosasco, and Tomaso Poggio. Learning and invariance in a family of hierarchical kernels. Technical report, Massachusetts Institute of Technology, 2010.

Christopher K. I. Williams and Carl Edward Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems 8*, pages 514–520. MIT Press, 1996.

Michael C. Zerner, Gilda H. Loew, Robert F. Kirchner, and Ulrich T. Mueller-Westerhoff. An intermediate neglect of differential overlap technique for spectroscopy of transition-metal complexes. Ferrocene. *Journal of the American Chemical Society*, 102(2):589–599, 1980.