Classification of medical data using Restricted Boltzmann Machines

Markku Aalto

University of Tampere
School of Information Sciences
Computer Science
M.Sc. Thesis
Supervisor: Martti Juhola
March 2014

University of Tampere

School of Information Sciences

Computer Science

Markku Aalto: Classification of medical data using Restricted Boltzmann Machines

M.Sc. Thesis, 49 pages

March 2014

Abstract

Restricted Boltzmann Machines are generative models commonly used for feature extraction and for training deep neural networks. In this thesis, their applicability for classification of medical data is researched. Three different approaches are evaluated using two small medical data sets. It is shown that the resulting classifiers are able to form sensible models of the data, having competitive performance when compared to other methods on these data sets.

Keywords and terms: Boltzmann Machine, Restricted Boltzmann Machine, neural network, classification, heterogeneous data

Contents

Chapter 1

Introduction

Restricted Boltzmann Machine (RBM) is a type of generative neural network. Its properties are described in detail in Section 3. Originally, similar models were introduced by Smolensky (1986), who called them "harmoniums". Later on, they became known as topologically restricted version of Boltzmann Machines, when Hinton (2002) suggested them as a Product of Experts model.

One of their most common uses today is as a building block for deep neural networks. This deep learning is often applied to various artificial intelligence tasks, such as image or speech recognition. In these domains the complex structure of the data necessitates the use of deep networks, and RBMs are considered one of the most efficient and effective ways of training them, achieving several state-of-the-art results. This area is reviewed in Bengio (2009), Bengio et al. (2012), and Hinton (2007).

These AI-related data sets are typically very large and homogeneous. In this

thesis, however, the applicability of RBMs to the task of classification of two smallish medical data sets (described in Section 2) is explored. Given that this is somewhat atypical application for this type of network, the relevant concerns are not thoroughly addressed in existing literature. Noise, overfitting, and other statistical problems are mostly associated with smaller and incomplete data sets and thus they are not of primary importance for typical deep networks.

There is some evidence that generative networks can perform quite well even with small training sets (Ng and Jordan, 2002). In Section 4, three different methods of using RBMs for classification are tested. The focus is on generative training followed by discriminative fine-tuning.

The classifiers were implemented in Python, taking advantage of Theano library (Bergstra et al., 2010). Theano optimizes and compiles the defined calculations for increased runtime performance.

Chapter 2

Data

2.1 Incontinence

The first data set concerns female urinary incontinence. Total of 529 patients are separated into five different classes, four of which are for the most common incontinence diagnoses and one is for continent patients. Class breakdown can be seen in Table 2.1. Class sizes are clearly imbalanced and quite small, especially for the last three classes.

For each patient, there are 13 variables - five quantitative and eight binary ones. Out of all values, 17.9 % are missing. There are 87 complete cases.

For comparison, some previous results are given in Table 2.2. The column headers are defined in Equations 4.1 - 4.8. These results were achieved using logistic regression on Expectation-Maximization imputed data (Laurikkala et al., 2001). Results for the last two classes could not be reliably computed due to their small size.

Table 2.1: Diagnosis frequencies in the Incontinence data set.

Incontinence class	Absolute frequency	Relative frequency (%)
Stress	323	61.1
Mixed	140	26.5
Sensory urge	33	6.2
Motor urge	15	2.8
Continent	18	3.4

Table 2.2: Previous classification results for the Incontinence data set.

Incontinence class	Sensitivity	Accuracy
Stress	0.86	0.83
Mixed	0.88	0.88
Sensory urge	0.50	0.99

Table 2.3: Diagnosis frequencies in the Vertigo data set.

Vertigo class	Absolute frequency	Relative frequency (%)
Vestibular schwannoma	130	16
Benign positional vertigo	146	18
Ménière's disease	313	38
Sudden deafness	41	5
Traumatic vertigo	65	8
Vestibular neuritis	120	15

2.2 Vertigo

The second data set is somewhat larger and more complex set about vertiginous patients. There are 815 patients divided into six classes, as shown in Table 2.3. Out of the total of 38 variables, 16 are quantitative, 10 are ordinal, 11 are binary, and one is nominal with 4 possible values. Handling ordinal variables correctly can be quite difficult and requires specific domain knowledge, so they are simply considered nominal in the rest of the work. This results in a loss of information about the ordering. About 11 % of the values are missing.

Previous results for this data set can be seen in Table 2.4. See Equations 4.1 - 4.8 for the header definitions. A set of perceptron neural networks (NetSet) were used for these results (Siermala et al., 2008).

Table 2.4: Previous classification results for the Vertigo data set.

Incontinence class	Sensitivity	Specificity	Precision	Accuracy
Vestibular schwannoma	0.76	0.98	0.85	0.92
Benign positional vertigo	0.81	0.90	0.60	0.88
Ménière's disease	0.73	0.91	0.82	0.84
Sudden deafness	1.00	1.00	0.89	1.00
Traumatic vertigo	0.92	0.98	0.87	0.97
Vestibular neuritis	0.86	0.98	0.93	0.95

2.3 Preprocessing

To deal with missing values, the data were imputed using Random Forest method with the missForest package (Stekhoven and Buehlmann, 2012) for R. Moreover, quantitative variables were normalized to zero mean and unit variance for reasons described in Section 3.2.2.

Chapter 3

Foundations

3.1 Boltzmann Machine

3.1.1 Modeling data

Boltzmann Machine (BM) is an energy-based neural network for modeling a set of binary vectors (see Ackley et al. (1985), Hinton and Sejnowski (1986)). It consists of stochastic binary units that are symmetrically connected to each other. The units are often divided into visible and hidden units. Visible units hold the actual observations that are modeled - they are comparable to the input layer of a perceptron. Hidden units can be thought of as explanations or feature detectors for the visible units. In practice they increase the modeling capacity of the network.

Let $\mathbf{W} = [w_{ij}]$ be the weight matrix for the connections between the units, where w_{ij} is the weight of the connection between units i and j, which can be

either visible or hidden units. As the connections are symmetrical, $w_{ij} = w_{ji}$. Each unit also has a bias; $\mathbf{a} = [a_i]$ is the bias vector for the visible units and $\mathbf{b} = [b_k]$ is for the hidden units.

Now, the network assigns an energy to every configuration of units

$$E(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta}) = -\sum_i a_i v_i - \sum_k b_k h_k - \sum_{i,j>i} w_{ij} v_i v_j - \sum_{k,l>k} w_{kl} h_k h_l - \sum_{i,k} w_{ik} v_i h_k \tag{3.1}$$

where $\mathbf{v}$ and $\mathbf{h}$ are binary vectors of the states of visible and hidden units, respectively, and $\boldsymbol{\theta}$ is vector of the model parameters $[\mathbf{W}, \mathbf{a}, \mathbf{b}]$. Here i and j iterate over the indices of visible units, while k and l do the same for hidden units. To unpack, the first two terms in Equation 3.1 come from visible and hidden biases, the next two from the connections within visible and hidden units (pairwise connections only counted once) and the last term from the connections between visible and hidden units. A probability distribution of unit state configurations is defined based on that energy

$$P(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta}) = \frac{e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})}}{Z_{\boldsymbol{\theta}}} \tag{3.2}$$

where

$$Z_{\boldsymbol{\theta}} = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})} \tag{3.3}$$

is the partition function (sometimes called the normalizing constant). We are mostly interested in the visible units, so the hidden ones can be marginalized away

$$P(\mathbf{v}|\boldsymbol{\theta}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})}}{Z_{\boldsymbol{\theta}}}. \tag{3.4}$$

For notational convenience, we can define free energy as

$$F(\mathbf{v}|\boldsymbol{\theta}) = -\log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}|\boldsymbol{\theta})} \tag{3.5}$$

allowing us to rewrite the previous equations as

$$P(\mathbf{v}|\boldsymbol{\theta}) = \frac{e^{-F(\mathbf{v}|\boldsymbol{\theta})}}{Z_{\boldsymbol{\theta}}} \tag{3.6}$$

$$Z_{\boldsymbol{\theta}} = \sum_{\mathbf{v}} e^{-F(\mathbf{v}|\boldsymbol{\theta})}. \tag{3.7}$$

To clarify how a BM models data, Figure 3.1 and Table 3.1 show an example. It should be noted that in this case it was possible to calculate the exact value for the partition function and thus the exact probabilities. As this requires summing over every possible configuration of the network, it is computationally intractable for realistically sized networks. Dealing with this problem will be a common theme later on. Also, the example net is not fully connected. This is equivalent to simply having a fixed weight of 0 for the lacking connections. BM itself does not restrict the topology.

Sampling from the network is difficult due to intractability of Z. There is, however, a Markov Chain Monte Carlo method for getting unbiased samples (although inefficiently). First, from Equation 3.2 it follows that

$$P(v_i = 1|\mathbf{v}_{\setminus i}, \mathbf{h}, \mathbf{W}, \mathbf{a}) = \sigma(\sum_{j \neq i} w_{ij} v_j + \sum_k w_{ik} h_k + a_i) \tag{3.8}$$

$$P(h_k = 1|\mathbf{h}_{\setminus k}, \mathbf{v}, \mathbf{W}, \mathbf{b}) = \sigma(\sum_{l \neq k} w_{kl} h_l + \sum_i w_{ki} v_i + b_k) \tag{3.9}$$

where $\mathbf{v}_{\setminus i}$ denotes vector $\mathbf{v}$ without ith element (that is, every state except for v_i is given), and $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function. Now, if you initialize the states randomly, and then update the state of each unit one at a time by applying these equations, eventually the Markov chain will reach stationary distribution and the network will end up in a configuration in accordance with the probabilities induced by the model parameters (Hinton

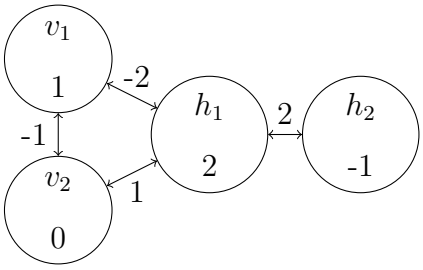

Figure 3.1: A small Boltzmann Machine with two visible units and two hidden units. The circles representing units have their respective label and bias written inside them. Between the units there are connections and their weights.

and Sejnowski, 1986). This process is called Gibbs sampling and in this case updating the state of every unit once is one full Gibbs step. In general, many steps are required to reach stationary distribution.

3.1.2 Training

For the network to be useful, it needs to model some actual data. Assume there is a training set $\mathcal{D}$ that consists of n binary vectors. The goal is to maximize the probability that when one samples from the network n times, one ends up with $\mathcal{D}$. This is equivalent to maximizing the product of probabilities the network assigns to vectors in $\mathcal{D}$, which, in turn, is the same as maximizing the sum of log probabilities of vectors in $\mathcal{D}$.

Table 3.1: Probability calculations for the Boltzmann Machine in Figure 3.1. The first four columns show the states of the respective units, with every configuration listed. Values in the last three columns are rounded to the given precision.

v_1	v_2	h_1	h_2	$E(\mathbf{v}, \mathbf{h}\|\boldsymbol{\theta})$	$e^{-E(\mathbf{v},\mathbf{h}\|\boldsymbol{\theta})}$	$P(\mathbf{v}, \mathbf{h}\|\boldsymbol{\theta})$	$P(\mathbf{v}\|\boldsymbol{\theta})$
0	0	0	0	0	1.0	0.008	
0	0	0	1	1	0.4	0.003	0.22
0	0	1	0	-2	7.4	0.057	
0	0	1	1	-3	20.1	0.154	
0	1	0	0	0	1.0	0.008	
0	1	0	1	1	0.4	0.003	0.58
0	1	1	0	-3	20.1	0.154	
0	1	1	1	-4	54.6	0.419	
1	0	0	0	-1	2.7	0.021	
1	0	0	1	0	1.0	0.008	0.11
1	0	1	0	-1	2.7	0.021	
1	0	1	1	-2	7.4	0.057	
1	1	0	0	0	1.0	0.008	
1	1	0	1	1	0.4	0.003	0.09
1	1	1	0	-1	2.7	0.021	
1	1	1	1	-2	7.4	0.057	
					$Z_{\boldsymbol{\theta}} \approx 130.2$		

Given a data vector $\mathbf{v} \in \mathcal{D}$, one can get the following partial derivatives:

$$\frac{\partial \log P(\mathbf{v}|\boldsymbol{\theta})}{\partial w_{ij}} = \langle x_i x_j \rangle_{\text{data}} - \langle x_i x_j \rangle_{\text{model}} \tag{3.10}$$

$$\frac{\partial \log P(\mathbf{v}|\boldsymbol{\theta})}{\partial a_i} = \langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{model}} \tag{3.11}$$

$$\frac{\partial \log P(\mathbf{v}|\boldsymbol{\theta})}{\partial b_k} = \langle h_k \rangle_{\text{data}} - \langle h_k \rangle_{\text{model}} \tag{3.12}$$

where $\langle y \rangle_P$ denotes the expected value of y in the probability distribution P and x_i is the state of unit i, which can be either hidden or visible unit. The distributions data and model refer to $P(\mathbf{h}|\mathbf{v}, \boldsymbol{\theta})$ and $P(\mathbf{h}, \mathbf{v}|\boldsymbol{\theta})$, respectively. One can sample from the first one by clamping the visible units to states according to $\mathbf{v}$ and running the Markov chain described in Section 3.1.1 over the rest of the units. For the second one, the same thing can be done with no clamping, allowing all states to be updated.

These two terms are often called the positive phase and the negative phase. Positive phase finds hidden configurations that work well with the visible configuration and decreases their energy (thus making the global configuration more probable). Negative phase finds global configurations that the net thinks are probable and increases their energy (getting rid of spurious minima).

From these derivatives one can directly get update rules for the parameters

$$\Delta w_{ij} = \epsilon(\langle x_i x_j \rangle_{\text{data}} - \langle x_i x_j \rangle_{\text{model}}) \tag{3.13}$$

$$\Delta a_i = \epsilon(\langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{model}}) \tag{3.14}$$

$$\Delta b_k = \epsilon(\langle h_k \rangle_{\text{data}} - \langle h_k \rangle_{\text{model}}) \tag{3.15}$$

where ϵ is the learning rate.

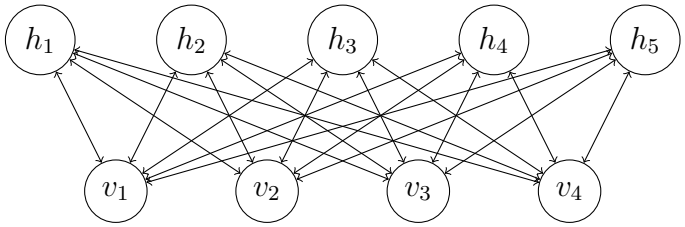

Figure 3.2: Restricted Boltzmann Machine with four visible units in the bottom layer and five hidden units in the top layer.

The theory here is sound, but there is a significant problem. It can take a relatively long time to get a sample and a lot of them are needed to get good approximations for the expected values. While it is possible to train a BM this way, it is not quite practical for large networks. In the next section, Restricted Boltzmann Machines are examined to find ways to speed up the training.

3.2 Restricted Boltzmann Machine

Restricted Boltzmann Machine (RBM) is simply a BM with restricted topology. It has exactly two layers, one for visible units and one for hidden units. There are no connections within the layers. In effect, the units form a bipartite graph (usually a complete one). An example is shown in Figure 3.2.

The equations in Section 3.1 also hold true for RBMs, but some of them can be simplified due to the lack of intra-layer connections. Additionally, hidden units become mutually independent given visible units and vice versa.

It is fair to ask whether RBMs are weaker models than general BMs, but it has been shown that RBMs can represent any discrete data distribution

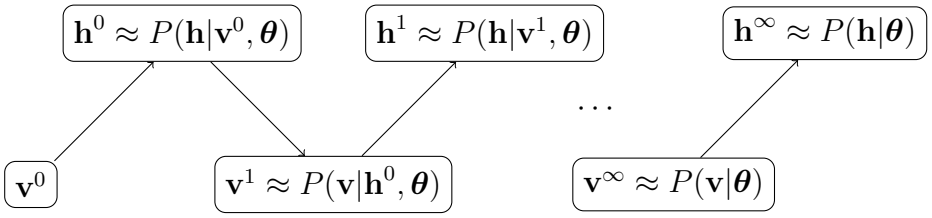

Figure 3.3: Gibbs sampling from an RBM. $\mathbf{v}^t$ is the vector of states of visible units at time t, and $\mathbf{h}^t$ is the same for hidden units.

given enough hidden units (Le Roux and Bengio, 2008).

To get samples, one can use Gibbs sampling in a more efficient way than with BMs in general. Instead of updating the state of a single unit at a time, one can update a whole layer in parallel, alternating between the layers. This is much faster than sequential updates when using modern, multi-threaded processors. This sampling process is shown in Figure 3.3. For positive phase, one would set $\mathbf{v}^0$ to a data vector from the training set and compute $\mathbf{h}^0$ from it. With visible units being clamped and hidden units being mutually independent given the visible units, stationary distribution would be reached in this single step and the states could be used for collecting statistics. Negative phase statistics could be collected by initializing $\mathbf{v}^0$ randomly and running the Markov chain until convergence, and then taking the samples.

Using an RBM solves the efficiency problem for positive phase, but getting samples for the negative phase is still too slow. To speed up the negative phase, Contrastive Divergence learning procedure makes some changes to the process to make it practical.

3.2.1 Contrastive Divergence

In Contrastive Divergence (CD), there are two ways to make sampling for the negative phase faster (Hinton, 2002). Firstly, instead of initializing the states randomly, they are set to a data point from the training set, as one would do in the positive phase. As the data and model distributions will be close to each other (at least after some training has already occurred), the starting point will already be more probable in the model and closer to convergence. Secondly, instead of running the chain to convergence, only a fixed number of Gibbs steps will be taken. Using the notation of Figure 3.3, the negative phase statistics will be collected from states $\mathbf{v}^n$ and $\mathbf{h}^n$ with n being the number of full Gibbs steps. This might seem like a risky move. If stationary distribution is not reached, then the samples are not truly from the model distribution, as they will be biased by the initial configuration. Using the expected values from these samples means that we are not directly maximizing the log probability any more. It has indeed been shown that the values received by CD are not derivatives of any function (Sutskever and Tieleman, 2010). Nevertheless, even CD-1 (CD with one Gibbs step) appears to work quite well in practice. More steps can be taken to better approximate the log probability gradient, if there are resources to do so.

One problem with CD is that the mixing rate of the Markov chain tends to slow down as learning progresses. This can be remedied to some extent by increasing the amount of Gibbs steps per update, but this also slows down the learning. Another learning procedure called Persistent Contrastive Divergence (PCD) uses a slightly modified way of getting samples from the model distribution (Tieleman, 2008). Instead of restarting the chains after

every parameter update, a fixed set of persistent chains is used. At first, a batch of samples is generated just as in CD negative phase. After updating the model parameters, the Markov chains for negative phase are continued from the last sampling points, instead of restarting them from training data points. This relies on the assumptions that the model changes only a little between each update and that the previous negative samples are good representations of the previous model distribution, making them even better initialization points than actual data points. The points induced by these chains are called fantasy particles. In effect, they roam around the energy landscape finding configurations likely in the model and increasing their energies. It is also possible to use multiple Gibbs steps per each update for PCD as it is for CD, but PCD seems to be more robust in this regard and a single step is often good enough.

3.2.2 Heterogeneous data

So far only binary units have been considered. Many data sets, including the ones used in this thesis, can not be efficiently represented in binary form (at least without sophisticated pre-processing). While there are several modifications of RBMs that use different types of units to allow for modeling continuous or other types of data, it is necessary to combine multiple unit types in a single network to handle the data sets in this study.

Categorical variables can be modeled by a set of binary units with the additional constraint that only one of them can be active at a time (Hinton, 2010). These binary units form a softmax unit, where the probability of each

binary unit being active is normalized by the whole group. If all visible units form a single softmax and x_i is the input for unit v_i, the probabilities are

$$P(v_i = 1|\mathbf{h}, \mathbf{W}, \mathbf{a}) = \frac{e^{x_i}}{\sum_j e^{x_j}}. \tag{3.16}$$

One way to deal with continuous variables is to use linear units with Gaussian noise (Hinton, 2010). In a network where the visible units are such Gaussian units and the hidden units are binary, the energy function becomes

$$E(\mathbf{v}, \mathbf{h}|\boldsymbol{\theta}) = \sum_i \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ji} \tag{3.17}$$

where σ_i is the standard deviation of the Gaussian noise for the visible unit v_i. Learning the standard deviations is possible, but it adds more complexity. This can be circumvented by normalizing the data such that each continuous variable has zero mean and unit variance. With normalized variables one can use fixed standard deviation and noiseless reconstruction of the visible units. Visible units will then behave as linear units where the updated value is the mean input. Stochasticity of the network will be preserved as the hidden units are still stochastic binary units.

Using mean updates for visible units is also possible for other unit types. With binary and softmax units this simply means using the probabilities directly instead of sampling a stochastic value according to them.

When multiple visible unit types exist in a single RBM, the energy function becomes a combination of their respective energy functions (Tran et al., 2011). Every visible unit contributes energy depending on its type: $(v_i - a_i)^2/2\sigma_i^2$ for linear visible units and $a_i v_i$ for binary ones. The hidden units are always binary so there is no change and the connections between hidden

and visible units end up as $v_i h_j w_{ji}$, if the visible units use fixed unit variance. In this context the categorical variables work as multiple binary ones.

3.2.3 Regularization

Data sets used in this study are quite small with many missing values. Due to this, overfitting is a significant problem. The gist is that the model with this problem describes the training data very well, but it is too specific and does not generalize to data it was not trained with. This is called the bias-variance tradeoff as the goal is to minimize both the bias and the variance of model error, but these tend do work against each other.

Common ways of preventing overfitting are to simply reduce the modeling capacity by using fewer hidden units or using different types of weight-decay. Weight-decay also has other benefits when used on RBMs (Hinton, 2010). Penalizing extreme weights is useful for making sure that units are actually active and not just stuck on or off. Keeping the weights close to zero also makes the mixing rate of Markov chain faster and thus improves the learning.

Instead of or in addition to weight-decay, it is also possible to use a weight constraint (Hinton et al., 2012). This means that there is a fixed upper limit for the L2 norm of incoming weight vector of every hidden unit. If a weight update would make the norm cross that limit, the weights are scaled so that they stay within bounds. As a result, it is possible to use higher learning rates to more effectively search the weight-space while keeping the weights reasonable and avoiding divergence.

3.2.4 Dropout

Another way of dealing with the bias-variance tradeoff is to use ensemble learning, where multiple models are trained with different subsets of training data and their results averaged. An effective way of achieving this with RBMs is to use dropout technique (Hinton et al., 2012). For each training case, hidden units are dropped out of the network with some probability p. The learning procedure and weight update is then done as if those dropped out units and weights related to them did not exist.

In effect, this results in a whole family of networks with weight sharing. To get their averaged result, one can activate every hidden unit, do an upwards pass to update the hidden unit states, and then a downwards pass where the outputs of hidden units are multiplied by p. If the model is trained so that approximately half the hidden units are active at a time, then activating all of them would cause about twice as much input to visible units, hence the scaling.

From another perspective, this reduces the co-adaptation of the hidden units, as they can not rely on other hidden units being there. If every subset of hidden units should be a reasonable model, it pushes every unit to be independently useful.

An example of dropout can be seen in Figure 3.4. For the next weight update the network will be trained as if only the three remaining hidden units and their connections existed. The weights and biases that correspond to the dashed units and connections will not change during this training.

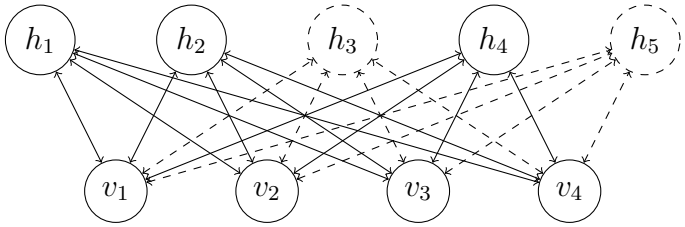

Figure 3.4: Restricted Boltzmann Machine with hidden units h_3 and h_5 dropped out.

3.2.5 Momentum

In gradient descent learning, it is common to use momentum to speed up the learning process. Sometimes directly following the steepest descent is not the optimal direction for minimizing the objective function in the long term, as it might be almost perpendicular to the direction of local minima, repeatedly crossing an energy ravine. Using momentum, the local opposing trends tend to cancel out and velocity will build towards common direction.

Nesterov's Accelerated Gradient (NAG) is a similar method that is used in this work. In classical momentum (CM), new velocity is calculated from current velocity and gradient, after which it is added to the current weights. In NAG, the gradient is calculated after a "simulated" update of weights with current velocity (Sutskever et al., 2013).

CM can be written as

$$v_{t+1} = \mu v_t + \epsilon g(\theta_t) \tag{3.18}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{3.19}$$

where v is the velocity, t is the time step, μ is the momentum coefficient, ϵ is the learning rate, θ is the parameter to optimize, and $g(\theta)$ is the gradient

at θ. Compared to this, NAG is

$$v_{t+1} = \mu v_t + \epsilon g(\theta_t + \mu v_t) \tag{3.20}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{3.21}$$

so the only difference is that the new gradient is calculated at the point where the current parameter would end up, if the current velocity was already applied.

Chapter 4

Classification

4.1 Testing setup

The Incontinence data set can be conveniently split into 15 batches of 34 data points with identical class composition in every batch by leaving a few samples out. In the following tests, the samples are randomly divided into these batches, and the tests are run by using 14 of these batches as the training set and the one leftover batch as the testing set. This is repeated 15 times so that each batch is the testing batch once. This whole process is repeated six times to collect the final statistics.

With the Vertigo data set the testing procedure is a bit simpler. For every run, 10 % of the data points are randomly selected into the testing set with the rest forming the training set. This is repeated 50 times to collect the statistics. It should be noted that while the total amount of tested samples is the same between different test runs, their class compositions might differ.

The results are presented in two tables per each combination classification method and data set. The first table shows specifically how testing samples belonging to each class were classified. If the number in row x and column y is n, that means that n testing samples that belong to class x were classified as y. Correctly classified samples are on the diagonal. Due to the difference in class compositions noted in the last paragraph, the absolute numbers for the Vertigo data set are not directly comparable between different methods.

The second table shows summary statistics for every class. If we define

$$\text{TP (True Positive)} = \#\text{ of correctly classified positive samples} \tag{4.1}$$

$$\text{TN (True Negative)} = \#\text{ of correctly classified negative samples} \tag{4.2}$$

$$\text{FP (False Positive)} = \#\text{ of incorrectly classified negative samples} \tag{4.3}$$

$$\text{FN (False Negative)} = \#\text{ of incorrectly classified positive samples,} \tag{4.4}$$

then

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP+ FN}} \tag{4.5}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN+ FP}} \tag{4.6}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}} \tag{4.7}$$

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{TP+ TN + FP + FN}}. \tag{4.8}$$

Here positive means that the sample belongs to the class in question and negative means that it belongs to another class.

4.2 Time considerations

Training a simple RBM mostly involves matrix multiplications and element-wise operations, both of which can be performed efficiently in parallel. Some additional difficulties are caused by using multiple unit types in a single network and techniques like dropout. The training time is obviously affected by the usual factors, such as the size of the training set, the number of epochs, the size of the mini-batches used in the training and so on. Out of RBM-specific meta-parameters, the most important ones are the amount of hidden units and the number of Gibbs steps taken per model update (e.g. CD-1 vs. CD-10).

After the RBM is trained, classifying a new sample mostly requires calculating its free energy, which can be done in linear time as a function of the amount of units in the network. For all the used classification methods, this testing phase is practically instantaneous.

In the following tests, the meta-parameters were adjusted such that a complete set of tests for a single data set and method could be performed in about eight hours or less using a decent laptop. Test sets are comprised of several dozen individual tests, so training and testing a single model usually took from five to ten minutes.

4.3 Joint-density Restricted Boltzmann Machine

4.3.1 Description

The training of RBMs is a form of unsupervised learning, as it is concerned with modeling the distribution of data rather than mapping it to labels. In Vertigo and Incontinence data sets, however, all data are labeled. The simplest way to include the labels in the model is to concatenate them with the features by making the label a categorical variable and to train a joint-density model of them.

Given an RBM trained in this way, it is theoretically possible to classify new samples by clamping the visible features of the sample and then trying every possible state for the visible unit corresponding to the label. For each of these combinations the probability can be calculated and the most probable label for the given features can be selected.

Unfortunately, due to the intractable partition function, these probabilities are difficult to calculate directly. Free energies of the visible units can be easily calculated, and because there is only one network, the partition function is the same for every combination of features and label. Thus, the free energies are directly comparable and the combination with the smallest free energy is also the most probable.

Another consideration is that CD/PCD learning is purely generative. Sampling from the network would result in the specific combination of features

and label with its respective probability. Training a good generative model is a very general and difficult problem, though. The classification capability comes essentially as a free side-effect with the generative model.

In practice it might be beneficial to be a bit less ambitious and focus explicitly on the classification performance, as that is the only thing we need in the end. This kind of discriminative training can be done by using the log probability of correct classification of test data directly as the objective function (Hinton, 2010). In these tests, generative training was first used to find the general structure of the data, followed by discriminative fine-tuning to improve final performance.

4.3.2 Results

For the Incontinence data set, 5000 epochs of PCD-1 learning were performed using 50 hidden units. Both L2 weight-decay and weight constraint were used. During training, momentum coefficient was slowly increased while learning rate was decreased. After this generative training, additional 1000 epochs of discriminative training with small learning rate were done to fine-tune the network for discrimination. The results are shown in Table 4.1. The main problem was that the Mixed class proved to be difficult to separate from the others, except for the Continent class.

With Vertigo the network had 100 hidden units and 1000 epochs of PCD-1 were done followed by another 1000 epochs of discriminative fine-tuning. See Table 4.2 for the results. The classification seemed quite consistent overall without any major problems.

Table 4.1: Joint-density RBM classification results for the Incontinence data set.

(a) Class breakdown

Class / Class	Stress	Mixed	Sensory urge	Motor urge	Continent
Stress	1716	152	8	0	14
Mixed	90	684	33	3	0
Sensory urge	5	84	94	0	5
Motor urge	0	29	3	58	0
Continent	2	0	0	0	88

(b) Summary statistics

Class	Sensitivity	Specificity	Precision	Accuracy
Stress	0.91	0.92	0.95	0.91
Mixed	0.84	0.88	0.72	0.87
Sensory urge	0.50	0.98	0.68	0.96
Motor urge	0.64	1.00	0.95	0.99
Continent	0.98	0.99	0.82	0.99

Table 4.2: Joint-density RBM classification results for the Vertigo data set.

(a) Class breakdown

Class \ Class	VS	BPV	MD	SD	TV	VN
Vestibular schwannoma (VS)	569	16	71	6	0	6
Benign positional vertigo (BPV)	4	607	81	0	16	16
Ménière's disease (MD)	25	48	1422	8	6	18
Sudden deafness (SD)	9	6	33	171	0	0
Traumatic vertigo (TV)	0	18	5	0	287	14
Vestibular neuritis (VN)	0	20	18	0	9	533

(b) Summary statistics

Class	Sensitivity	Specificity	Precision	Accuracy
Vestibular schwannoma	0.85	0.99	0.94	0.97
Benign positional vertigo	0.84	0.97	0.85	0.94
Ménière's disease	0.93	0.92	0.87	0.92
Sudden deafness	0.78	1.00	0.92	0.98
Traumatic vertigo	0.89	0.99	0.90	0.98
Vestibular neuritis	0.92	0.98	0.91	0.98

4.4 Multiple Restricted Boltzmann Machines

4.4.1 Description

Another approach is to train a separate RBM for each class. Again, it would be theoretically possible to simply calculate how probable a data vector is in each of those RBMs and select the most probable one, but the exact probabilities are intractable. With multiple networks and differing partition functions, not even the free energies are directly comparable.

If $\mathbf{v}$ is a data vector and A is a class, then

$$P(\mathbf{v}|A) = P(\mathbf{v}|\boldsymbol{\theta}_A) = \frac{e^{-F(\mathbf{v}|\boldsymbol{\theta}_A)}}{Z_A} \tag{4.9}$$

with $\boldsymbol{\theta}_A$ being the model parameters for the RBM trained for class A, and $Z_{\boldsymbol{\theta}_A}$ being abbreviated as Z_A. We can then use Bayes' theorem for classification

$$\begin{aligned} P(A|\mathbf{v}) &= \frac{P(\mathbf{v}|\boldsymbol{\theta}_A)}{\sum_X P(\mathbf{v}|\boldsymbol{\theta}_X)} \\ &= \frac{\frac{1}{Z_A} e^{-F(\mathbf{v}|\boldsymbol{\theta}_A)}}{\sum_X \frac{1}{Z_X} e^{-F(\mathbf{v}|\boldsymbol{\theta}_X)}} \\ &= \frac{e^{-F(\mathbf{v}|\boldsymbol{\theta}_A)}}{\sum_X \frac{Z_A}{Z_X} e^{-F(\mathbf{v}|\boldsymbol{\theta}_X)}} \end{aligned} \tag{4.10}$$

where X iterates over every class. To calculate class probabilities at least the ratios between the different partition functions need to be known. One way to find working values is to simply use maximum likelihood learning to discriminatively find values that work well for the classification task (Schmah et al., 2008). Unfortunately, attempts to find direct values for the partition functions and thus consistent ratios in parallel for more than three classes proved to be unsuccessful in the current study.

To get around this problem, each ratio was found separately by only considering the relevant two classes during training. It should be noted that the ratios found this way are not consistent (for example, $\frac{Z_A}{Z_B}\frac{Z_B}{Z_C} \neq \frac{Z_A}{Z_C}$), but they can nevertheless be used for calculating approximate probabilities.

4.4.2 Results

For Incontinence, the RBMs had 50 hidden units. They were trained for 2000 epochs using CD-10. Momentum, decreasing learning rate and L2 weight-decay were utilized as before. Additionally, 50 % dropout was used for hidden units. To find the ratios, 5000 epochs of maximum likelihood learning were performed for each pair of classes. The results can be seen in Table 4.3. Here the problem with the Mixed class is even more prominent. Especially Sensory urge has notably low sensitivity of just 0.12 with bulk of the samples classified as Mixed and the rest divided mostly between Sensory urge and Motor urge. Due to the three middle classes getting muddled up, the statistics for those classes are quite weak.

With Vertigo mostly the same parameters as with Incontinence were used, except that the RBMs had 75 hidden units and they were trained for 3000 epochs. Results can be seen in Table 4.4. There seem to be quite a lot of misses without any apparent trend. Notably Sudden deafness has low precision while Benign positional vertigo has low sensitivity. Overall accuracy still remains decent despite these problems.

There are several possible ways to improve the classification. Obviously the method used for finding partition function ratios is far from optimal. The

Table 4.3: Multiple RBM classification results for the Incontinence data set.

(a) Class breakdown

Class / Class	Stress	Mixed	Sensory urge	Motor urge	Continent
Stress	1631	238	3	3	15
Mixed	42	742	11	9	6
Sensory urge	3	125	22	24	6
Motor urge	0	31	1	58	0
Continent	1	0	0	0	89

(b) Summary statistics

Class	Sensitivity	Specificity	Precision	Accuracy
Stress	0.86	0.96	0.97	0.90
Mixed	0.92	0.82	0.65	0.85
Sensory urge	0.12	0.99	0.59	0.94
Motor urge	0.64	0.99	0.62	0.98
Continent	0.99	0.99	0.77	0.99

same training set was used for training the RBMs and for finding the ratios. The problem is that the specific training samples used for training the RBMs are bound to have very low free energies as the network is optimized for them. Thus, the free energies of those samples are not representative of the class in general and result in bad ratios. The reason for not using separate training and testing sets is that some classes only had a few samples, and losing any training samples made the model rapidly worse. It would be possible to only do the splitting for larger classes, but it was not done due to consistency and simplicity.

Another possible problem is that the training for RBMs is purely generative without any discriminative training signal. Discriminative fine-tuning significantly improved the classification performance of Joint-density RBMs and while the situation is not directly comparable, it is reasonable that it could also improve the performance here. This was not done here, because training the RBMs in parallel discriminatively would have required significant architectural overhaul in the implementation.

Table 4.4: Multiple RBM classification results for the Vertigo data set.

(a) Class breakdown

Class / Class	VS	BPV	MD	SD	TV	VN
Vestibular schwannoma (VS)	446	0	5	46	0	0
Benign positional vertigo (BPV)	58	614	188	17	14	20
Ménière's disease (MD)	74	56	1243	30	5	15
Sudden deafness (SD)	0	0	13	92	0	2
Traumatic vertigo (TV)	12	42	39	8	310	21
Vestibular neuritis (VN)	32	10	47	6	14	571

(b) Summary statistics

Class	Sensitivity	Specificity	Precision	Accuracy
Vestibular schwannoma	0.90	0.95	0.72	0.94
Benign positional vertigo	0.67	0.97	0.85	0.90
Ménière's disease	0.87	0.89	0.81	0.88
Sudden deafness	0.86	0.97	0.46	0.97
Traumatic vertigo	0.72	0.99	0.90	0.96
Vestibular neuritis	0.84	0.98	0.91	0.96

4.5 Deep Belief Network

4.5.1 Description

RBMs are often used as building blocks for deep networks (Bengio et al., 2012). One such network is Deep Belief Network (DBN), which can be trained by stacking RBMs on top of one another. The process is started by training a single RBM as the bottom layer as usual. Then another RBM is trained that uses the hidden units of the previous RBM as its visible units. Training set for the second RBM is generated by taking an original training sample, setting it as the visible units of the bottom RBM and then updating the states of the hidden units of the bottom RBM according to their marginal probabilities. This is repeated for all training samples and the resulting vectors of hidden unit states form a training set for the second RBM.

For classification purposes one can add an output layer for the labels on top of the network, and then fine-tune it for discrimination using standard backpropagation. In effect, this works like a Multi-Layer Perceptron (MLP) except that the weights are pre-trained by the greedy layerwise process of training the RBMs.

The reason for this pre-training is that simple backpropagation becomes increasingly inefficient as the number of layers grows. Pre-trained network can already extract underlying structure of the data, so the backpropagation has to perform mainly demarcation for the labels to fine-tune it for discrimination. Another significant benefit is that the pre-training is unsupervised. In many data sets, only a small subset of the data is labeled. Labeled data

is only required for the fine-tuning so this allows all available data to be efficiently used.

In these data sets, however, all data is labeled and there is not a lot of it. It could be said that the amount of data does not really warrant a deep network in the first place. This multi-layer approach has been justified by showing that with enough hidden units and correct initialization, increasing the number of layers improves the lower bound of the log probability of the training data when the network is used as a generative model (Hinton et al., 2006). In preliminary tests, however, the classification performance did not improve with additional layers.

With shallow networks, the pre-training is mostly inconsequential as the backpropagation is quite capable of finding good parameters for them. There are also many similarities with Joint-density RBM and DBN with one hidden layer. It was then decided to use three hidden layers for these tests as a compromise between differentiating from Joint-density RBM and not having unnecessary layers.

4.5.2 Results

For the Incontinence set, three hidden layers of 30 units each is used. The RBMs are trained for 1000 epochs using CD-10. After the pre-training, a logistic regression layer is added on top and trained using backpropagation for 5000 epochs. See Table 4.5 for the results. They are very similar to the ones achieved by the Joint-density RBM, except that there are slightly more misses.

Table 4.5: DBN classification results for the Incontinence data set.

(a) Class breakdown

Class / Class	Stress	Mixed	Sensory urge	Motor urge	Continent
Stress	1730	139	8	0	13
Mixed	109	659	35	4	3
Sensory urge	4	88	72	10	6
Motor urge	2	12	14	62	0
Continent	10	0	0	0	80

(b) Summary statistics

Class	Sensitivity	Specificity	Precision	Accuracy
Stress	0.92	0.89	0.93	0.91
Mixed	0.81	0.89	0.73	0.87
Sensory urge	0.40	0.98	0.56	0.95
Motor urge	0.69	1.00	0.82	0.99
Continent	0.89	0.99	0.78	0.99

With Vertigo the parameters were otherwise similar except that each hidden layer had 50 units. Results are in Table 4.6. These, in turn, resemble the Multiple RBM results a lot. The same classes are problematic and the rest are very similar, with DBN generally being slightly better.

In these tests the discriminative training was performed directly after the layer-wise training of the RBMs. It would also be possible, after training the RBMs, to improve the generative performance of the whole network using a contrastive wake-sleep algorithm (Hinton et al., 2006).

Table 4.6: DBN classification results for the Vertigo data set.

(a) Class breakdown

Class / Class	VS	BPV	MD	SD	TV	VN
Vestibular schwannoma (VS)	464	0	10	29	0	0
Benign positional vertigo (BPV)	56	614	205	26	14	29
Ménière's disease (MD)	81	65	1233	26	3	9
Sudden deafness (SD)	1	0	10	99	2	1
Traumatic vertigo (TV)	7	50	30	10	316	16
Vestibular neuritis (VN)	23	9	50	4	4	554

(b) Summary statistics

Class	Sensitivity	Specificity	Precision	Accuracy
Vestibular schwannoma	0.92	0.95	0.73	0.95
Benign positional vertigo	0.65	0.96	0.83	0.89
Ménière's disease	0.87	0.88	0.80	0.88
Sudden deafness	0.88	0.98	0.51	0.97
Traumatic vertigo	0.74	0.99	0.93	0.97
Vestibular neuritis	0.86	0.98	0.91	0.96

Chapter 5

Conclusions

Joint-density RBM appeared to be the best classifier in these tests, while also being the simplest one. With the Incontinence data set it had quite similar performance when compared with the existing results (see Tables 2.2 and 4.1). The sensitivity was slightly higher for the Stress class and slightly lower for the Mixed class. Both had problems with the Sensory urge class, only having 0.50 sensitivity. Overall accuracy was somewhat better for the Stress class, slightly worse for the Sensory urge and approximately equal for the Mixed class. The last two classes, Motor urge and Continent, had no previous results but they were both reasonably well classified, except that Motor urge had only 0.64 sensitivity.

Results for the Vertigo data set were also quite close with the previous ones (see Tables 2.4 and 4.2). In terms of sensitivity, Ménière's disease had notably better one while Sudden deafness had worse. As Ménière's disease was the largest class, this helped in raising the accuracy higher for every class except

for Sudden deafness. The second largest class, Benign positional vertigo, also had overall better statistics. Smaller classes had very similar results, with the exception of Sudden deafness, which had excellent existing results.

There is certainly room for improvement just by adjusting meta-parameters and using more computational resources. The ones used for these tests were selected simply by manually testing a few options for each value and choosing a reasonably working combination. The classification methods themselves could also be improved. Some of their possible shortfalls are discussed in their respective sections.

Numerous different ways of improving RBM learning have been suggested in the literature. For example, using rectified linear units instead of binary units as hidden units (Dahl et al., 2013) has been beneficial in some cases. There has also been attempts to address common learning problems by using adaptive learning rate or other learning rate schedules (Cho et al., 2011), or by using other learning procedures than CD/PCD, such as parallel tempering (Desjardins et al., 2010). The problem with implementing many of these is that they tend to make the process more complex, making it harder to reason about it. They might also add even more meta-parameters making it more difficult to find optimal combinations. Moreover, it is not always obvious how they should work together when implemented at the same time.

Another source of bias is the imputation of missing values. Some of the classes only have a very small amount of samples, which makes the imputation harder and its effects more significant. It would be possible to use unimputed data for training using a method similar to dropout (see Section 3.2.4), where the dropped out units are the visible units corresponding to

missing values. This approach has been successfully used for collaborative filtering (Salakhutdinov et al., 2007).

In summary, generative networks such as RBMs appear to be a promising alternative for these kinds of classification tasks, where only a limited amount of quite noisy data is available. The area is progressing quickly with constant new research into different methods and techniques. It can be expected that with increasing practical experience and insight into these models, there will be a good general understanding of how to optimize them for different problems.

Bibliography

Ackley, David H, Geoffrey E Hinton, and Terrence J Sejnowski (1985), A learning algorithm for Boltzmann machines. *Cognitive Science*, 9, 147–169.

Bengio, Yoshua (2009), Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2, 1–127.

Bengio, Yoshua, Aaron C Courville, and Pascal Vincent (2012), Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538.

Bergstra, James, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio (2010), Theano: a CPU and GPU Math Expression Compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Cho, KyungHyun, Tapani Raiko, and Alexander Ilin (2011), Enhanced Gradient and Adaptive Learning Rate for Training Restricted Boltzmann Machines. In *Proceedings of the 28th International Conference on Machine*

Learning (ICML-11) (Lise Getoor and Tobias Scheffer, eds.), ICML '11, 105–112, ACM, New York, NY, USA.

Dahl, George E, Tara N Sainath, and Geoffrey E Hinton (2013), Improving deep neural networks for LVCSR using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 8609–8613, IEEE.

Desjardins, Guillaume, Aaron Courville, and Yoshua Bengio (2010), Adaptive parallel tempering for stochastic maximum likelihood learning of RBMs. *arXiv preprint arXiv:1012.3476*.

Hinton, Geoffrey (2010), A practical guide to training restricted Boltzmann machines. Technical Report UTML TR 2010-003, Department of Computer Science, University of Toronto.

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

Hinton, Geoffrey E (2007), Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11, 428–434.

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

Hinton, Geoffrey E and Terrance J Sejnowski (1986), Learning and relearning in Boltzmann machines. *MIT Press, Cambridge, Mass*, 1, 282–317.

Hinton, Geoffrey E, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov (2012), Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Laurikkala, Jorma, Martti Juhola, Seppo Lammi, Jorma Penttinen, and Pauliina Aukee (2001), Analysis of the imputed female urinary incontinence data for the evaluation of expert system parameters. *Computers in Biology and Medicine*, 31, 239–257.

Le Roux, Nicolas and Yoshua Bengio (2008), Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, 20, 1631–1649.

Ng, Andrew Y and Michael I Jordan (2002), On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in Neural Information Processing Systems*, 2, 841–848.

Salakhutdinov, Ruslan, Andriy Mnih, and Geoffrey Hinton (2007), Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, 791–798, ACM.

Schmah, Tanya, Geoffrey E Hinton, Steven L Small, Stephen Strother, and Richard S Zemel (2008), Generative versus discriminative training of RBMs for classification of fMRI images. In *Advances in Neural Information Processing Systems*, 1409–1416.

Siermala, Markku, Martti Juhola, and Erna Kentala (2008), Neural network classification of otoneurological data and its visualization. *Computers in Biology and Medicine*, 38, 858–866.

Smolensky, Paul (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations*, chapter Information pro-

cessing in dynamical systems: foundations of harmony theory, 194–281. MIT Press, Cambridge, MA, USA.

Stekhoven, Daniel J and Peter Buehlmann (2012), MissForest - nonparametric missing value imputation for mixed-type data. *Bioinformatics*, 28, 112–118.

Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton (2013), On the importance of initialization and momentum in deep learning. In *Proceedings of The 30th International Conference on Machine Learning*, volume 28, 1139–1147, JMLR.org.

Sutskever, Ilya and Tijmen Tieleman (2010), On the convergence properties of contrastive divergence. In *International Conference on Artificial Intelligence and Statistics*, 789–795.

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

Tran, Truyen, Dinh Phung, and Svetha Venkatesh (2011), Mixed-variate restricted Boltzmann machines. In *ACML 2011: Proceedings of the 3rd Asian Conference on Machine Learning*, 213–229, JMLR.org.