

The Restricted Boltzmann Machine

Vilde Flugsrud

Department of Physics, UiO

April 12, 2018

Table of Contents

Introduction

The Restricted Boltzmann machine

Structure

Joint distribution and the Energy function

Defining different types of RBMs

Binary-Binary RBM

Gaussian-Binary RBM

Other RBMs

Sampling

Metropolis sampling

Gibbs sampling

Cost function

Optimization / Training

RBMs for the quantum many body problem

Choose the right RBM

Representing the wave function

Choose the cost function

The physical system

Table of Contents

Introduction

The Restricted Boltzmann machine

Structure

Joint distribution and the Energy function

Defining different types of RBMs

Binary-Binary RBM

Gaussian-Binary RBM

Other RBMs

Sampling

Metropolis sampling

Gibbs sampling

Cost function

Optimization / Training

RBMs for the quantum many body problem

Choose the right RBM

Representing the wave function

Choose the cost function

The physical system

- ▶ Why use a generative ("dreaming") model rather than the more well known discriminative deep neural networks (DNN)?
 - ▶ Discriminative methods have several limitations: They are supervised learning methods, thus requiring labeled data. And there are tasks they cannot accomplish, like drawing new examples from an unknown probability distribution.
 - ▶ A generative model can learn to represent and sample from a probability distribution. The core idea is to learn a parametric model of the probability distribution from which the training data was drawn. For:
 - ▶ A model for images could learn to draw new examples of cats and dogs, given a training dataset of images of cats and dogs.
 - ▶ Generate a sample of an ordered or disordered Ising model phase, having been given samples of such phases.

- ▶ Some similarities and differences from DNNs:
 - ▶ Both use gradient-descent based learning procedures for minimizing cost functions
 - ▶ Energy based models don't use backpropagation and automatic differentiation for computing gradients, instead turning to Markov Chain Monte Carlo methods.
 - ▶ DNNs often have several hidden layers. A restricted Boltzmann machine has only one hidden layer, however several RBMs can be stacked to make up Deep Belief Networks, of which they constitute the building blocks.
- ▶ History: The RBM was developed by amongst others Geoffrey Hinton, called by some the "Godfather of Deep Learning", working with the University of Toronto and Google.

Table of Contents

Introduction

The Restricted Boltzmann machine

Structure

Joint distribution and the Energy function

Defining different types of RBMs

Binary-Binary RBM

Gaussian-Binary RBM

Other RBMs

Sampling

Metropolis sampling

Gibbs sampling

Cost function

Optimization / Training

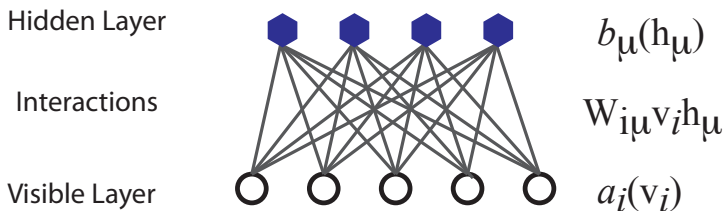
RBMs for the quantum many body problem

Choose the right RBM

Representing the wave function

Choose the cost function

The physical system



We here see some of the essential components of the restricted Boltzmann machine:

The network layers:

- ▶ \mathbf{x} represents the visible layer, a vector of M elements (nodes). This layer represents both what the RBM might be given as training input, *and* what we want it to be able to reconstruct. This might for example be the pixels of an image, the spin values of the Ising model, or coefficients representing speech.
- ▶ \mathbf{h} represents the hidden, or latent, layer. A vector of N elements (nodes). Also called "feature detectors".

The goal of the hidden layer is to increase the model's expressive power. We encode complex interactions between visible variables by introducing additional, hidden variables that interact with visible degrees of freedom in a simple manner, yet still reproduce the complex correlations between visible degrees in the data once marginalized over (integrated out).

Examples of this trick being employed in physics:

- ▶ The Hubbard-Stratonovich transformation
- ▶ The introduction of ghost fields in gauge theory

The network parameters, to be optimized/learned:

- ▶ \mathbf{a} represents the visible bias, a vector of same length as \mathbf{x} .
- ▶ \mathbf{b} represents the hidden bias, a vector of same length as \mathbf{h} .
- ▶ W represents the interaction weights, a matrix of size $M \times N$.

Joint distribution and the Energy function

The restricted Boltzmann machine is described by a Boltzmann distribution

$$P_{rbm}(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-\frac{1}{T_0} E(\mathbf{x}, \mathbf{h})} \quad (1)$$

where Z is the normalization constant or partition function, defined as

$$Z = \int \int e^{-\frac{1}{T_0} E(\mathbf{x}, \mathbf{h})} d\mathbf{x} d\mathbf{h} \quad (2)$$

It is common to ignore T_0 by setting it to one.

The function $E(\mathbf{x}, \mathbf{h})$ gives the *energy* of a configuration (pair of vectors) (\mathbf{x}, \mathbf{h}) . The lower the energy of a configuration, the higher the probability of it. This function also depends on the parameters \mathbf{a} , \mathbf{b} and W . Thus, when we adjust them during the learning procedure, we are adjusting the energy function to best fit our problem.

Defining different types of RBMs

There are different variants of RBMs, and the differences lie in the types of visible and hidden units we choose as well as in the implementation of the energy function $E(\mathbf{x}, \mathbf{h})$.

Binary-Binary RBM

RBM's were first developed using binary units in both the visible and hidden layer. The corresponding energy function is defined as follows:

$$E(\mathbf{x}, \mathbf{h}) = - \sum_i^M x_i a_i - \sum_j^N b_j h_j - \sum_{i,j}^{M,N} x_i w_{ij} h_j \quad (3)$$

where the binary values taken on by the nodes are most commonly 0 and 1.

Gaussian-Binary RBM

Another variant is the RBM where the visible units are Gaussian while the hidden units remain binary:

$$E(\mathbf{x}, \mathbf{h}) = \sum_i^M \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_j^N b_j h_j - \sum_{i,j}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2} \quad (4)$$

- ▶ Useful when we model continuous data (i.e., we wish \mathbf{x} to be continuous)
- ▶ Requires a smaller learning rate, since there's no upper bound to the value a component might take in the reconstruction

Other types of units include:

- ▶ Softmax and multinomial units
- ▶ Gaussian visible and hidden units
- ▶ Binomial units
- ▶ Rectified linear units

Sampling

In order to sample from the RBM probability distribution it is common to use Markov Chain Monte Carlo (MCMC) algorithms such as Metropolis-Hastings or Gibbs sampling.

Metropolis sampling

Metropolis sampling starts by suggesting a new configuration \mathbf{x}^{k+1} . In the brute force method this is done by some random change of the visible units. The new configuration is then accepted with the acceptance probability

$$A(\mathbf{x}^k \rightarrow \mathbf{x}^{k+1}) = \min(1, \frac{P(\mathbf{x}^{k+1})}{P(\mathbf{x}^k)}) \quad (5)$$

Where we need the marginalized probability

$$P(\mathbf{x}) = \sum_{\mathbf{h}} P_{rbm}(\mathbf{x}, \mathbf{h}) \quad (6)$$

$$= \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})} \quad (7)$$

Gibbs sampling

In this method we sample from the joint probability $P_{rbm}(\mathbf{x}, \mathbf{h})$ by way of a two step sampling process. We alternately update the visible and hidden units. New samples are generated according to the conditional probabilities $P(x_i|\mathbf{h})$ and $P(h_j|\mathbf{x})$ respectively and accepted with the probability of 1. While the the visible nodes are dependent on the hidden nodes and vice versa, the nodes are independent of other nodes within the same layer. This is due to there being no intra layer interactions in the restricted Boltzmann machine. The conditional probabilities are often referred to as the activation functions in the neural networks context due to their role in determining the node outputs.

For the binary-binary RBM they are

$$P(h_j = 1|\mathbf{x}) = \frac{1}{1 + e^{-b_j - \sum_i x_i w_{ij}}} \quad (8)$$

$$P(x_i = 1|\mathbf{h}) = \frac{1}{1 + e^{-a_i - \sum_j h_j w_{ij}}} \quad (9)$$

where we recognize the logistic sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$. For the Gaussian-Binary RBM the conditional probabilities are

$$P(x_i|\mathbf{h}) = \mathcal{N}(x_i; a_i + \sum_j h_j w_{ij}, \sigma^2) \quad (10)$$

$$P(h_j = 1|\mathbf{x}) = \frac{1}{1 + e^{-b_j - \frac{1}{\sigma^2} \sum_i x_i w_{ij}}} \quad (11)$$

while the visible units now follow a normal distribution, we see the hidden units again follow the logistic sigmoid function.

Cost function

When working with a training dataset, the most common training approach is maximizing the log-likelihood of the training data. The log likelihood characterizes the log-probability of generating the observed data using our generative model. Using this method our cost function is chosen as the negative log-likelihood. The learning then consists of trying to find parameters that maximize the probability of the dataset, and is known as Maximum Likelihood Estimation (MLE).

Denoting the parameters as $\theta = a_1, \dots, a_M, b_1, \dots, b_N, w_{11}, \dots, w_{MN}$, the log-likelihood is given by

$$\mathcal{L}(\{\theta_i\}) = \langle \log P_{\theta}(\mathbf{x}) \rangle_{data} \quad (12)$$

$$= -\langle E(\mathbf{x}; \{\theta_i\}) \rangle_{data} - \log Z(\{\theta_i\}) \quad (13)$$

where we used that the normalization constant does not depend on the data, $\langle \log Z(\{\theta_i\}) \rangle = \log Z(\{\theta_i\})$. Our cost function is the negative log-likelihood, $\mathcal{C}(\{\theta_i\}) = -\mathcal{L}(\{\theta_i\})$

Optimization / Training

The training procedure of choice often is Stochastic Gradient Descent (SGD). It consists of a series of iterations where we update the parameters according to the equation

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla \mathcal{C}(\boldsymbol{\theta}_k) \quad (14)$$

at each k -th iteration. There are a range of variants of the algorithm which aim at making the learning rate η more adaptive so the method might be more efficient while remaining stable.

We now need the gradient of the cost function in order to minimize it. We find that

$$\frac{\partial \mathcal{C}(\{\theta_i\})}{\partial \theta_i} = \left\langle \frac{\partial E(\mathbf{x}; \theta_i)}{\partial \theta_i} \right\rangle_{data} + \frac{\partial \log Z(\{\theta_i\})}{\partial \theta_i} \quad (15)$$

$$= \langle O_i(\mathbf{x}) \rangle_{data} - \langle O_i(\mathbf{x}) \rangle_{model} \quad (16)$$

where in order to simplify notation we defined the "operator"

$$O_i(\mathbf{x}) = \frac{\partial E(\mathbf{x}; \theta_i)}{\partial \theta_i} \quad (17)$$

and used the statistical mechanics relationship between expectation values and the log-partition function:

$$\langle O_i(\mathbf{x}) \rangle_{model} = \text{Tr} P_\theta(\mathbf{x}) O_i(\mathbf{x}) = - \frac{\partial \log Z(\{\theta_i\})}{\partial \theta_i} \quad (18)$$

$$\frac{\partial \mathcal{C}(\{\theta_i\})}{\partial \theta_i} = \langle O_i(\mathbf{x}) \rangle_{data} - \langle O_i(\mathbf{x}) \rangle_{model} \quad (19)$$

The data-dependent term in the gradient is known as the positive phase of the gradient, while the model-dependent term is known as the negative phase of the gradient. The aim of the training is to lower the energy of configurations that are near observed data points (increasing their probability), and raising the energy of configurations that are far from observed data points (decreasing their probability).

The gradient of the negative log-likelihood cost function of a Binary-Binary RBM is then

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial w_{ij}} = \langle x_i h_j \rangle_{data} - \langle x_i h_j \rangle_{model} \quad (20)$$

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial a_i} = \langle x_i \rangle_{data} - \langle x_i \rangle_{model} \quad (21)$$

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial b_j} = \langle h_j \rangle_{data} - \langle h_j \rangle_{model} \quad (22)$$

$$(23)$$

To get the expectation values with respect to the *data*, we set the visible units to each of the observed samples in the training data, then update the hidden units according to the conditional probability found before. We then average over all samples in the training data to calculate expectation values with respect to the data.

To get the expectation values with respect to the *model*, we use Gibbs sampling. We can either initialize the \mathbf{x} randomly or with a training sample. While we ideally want a large number of Gibbs iterations $n \rightarrow \infty$, one might decide to truncate it earlier for efficiency. Doing this while having initialized \mathbf{x} with a training data vector is referred to as contrastive divergence (CD), because one is then closer to approximating the gradient of this function than the negative log-likelihood. The contrastive divergence function is the difference between two Kullback-Leibler divergences (also called relative entropy), which measure how one probability distribution diverges from a second, expected probability distribution (in this case the estimated one from the ground truth one).

Table of Contents

Introduction

The Restricted Boltzmann machine

Structure

Joint distribution and the Energy function

Defining different types of RBMs

Binary-Binary RBM

Gaussian-Binary RBM

Other RBMs

Sampling

Metropolis sampling

Gibbs sampling

Cost function

Optimization / Training

RBM for the quantum many body problem

Choose the right RBM

Representing the wave function

Choose the cost function

The physical system

The idea of applying RBMs to quantum many body problems was presented by G. Carleo and M. Troyer, working with ETH Zurich and Microsoft Research.

Some of their motivation included

- ▶ "The wave function Ψ is a monolithic mathematical quantity that contains all the information on a quantum state, be it a single particle or a complex molecule. In principle, an exponential amount of information is needed to fully encode a generic many-body quantum state."
- ▶ There are still interesting open problems, including fundamental questions ranging from the dynamical properties of high-dimensional systems to the exact ground-state properties of strongly interacting fermions.

- ▶ The difficulty lies in finding a general strategy to reduce the exponential complexity of the full many-body wave function down to its most essential features. That is

→ Dimensional reduction (24)

→ Feature extraction (25)

- ▶ Among the most successful techniques to attack these challenges, artificial neural networks play a prominent role.
- ▶ Want to understand whether an artificial neural network may adapt to describe a quantum system.

Carleo and Troyer applied the RBM to the quantum mechanical spin lattice systems of the Ising model and Heisenberg model, with encouraging results. Our goal is to test the method on systems of moving particles. For the spin lattice systems it was natural to use a binary-binary RBM, with the nodes taking values of 1 and -1. For moving particles, on the other hand, we want the visible nodes to be continuous, representing position coordinates. Thus, we start by choosing a Gaussian-binary RBM, where the visible nodes are continuous and hidden nodes take on values of 0 or 1. If eventually we would like the hidden nodes to be continuous as well the rectified linear units seem like the most relevant choice.

The wavefunction should be a probability amplitude depending on \mathbf{x} . The RBM model is given by the joint distribution of \mathbf{x} and \mathbf{h}

$$F_{rbm}(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-\frac{1}{T_0} E(\mathbf{x}, \mathbf{h})} \quad (26)$$

To find the marginal distribution of \mathbf{x} we set:

$$F_{rbm}(\mathbf{x}) = \sum_{\mathbf{h}} F_{rbm}(\mathbf{x}, \mathbf{h}) \quad (27)$$

$$= \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})} \quad (28)$$

Now this is what we use to represent the wave function, calling it a neural-network quantum state (NQS)

$$\Psi(\mathbf{X}) = F_{rbm}(\mathbf{x}) \quad (29)$$

$$= \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})} \quad (30)$$

$$= \frac{1}{Z} \sum_{\{h_j\}} e^{-\sum_i^M \frac{(x_i - a_i)^2}{2\sigma^2} + \sum_j^N b_j h_j + \sum_{i,j}^{M,N} \frac{x_i w_{ij} h_j}{\sigma^2}} \quad (31)$$

$$= \frac{1}{Z} e^{-\sum_i^M \frac{(x_i - a_i)^2}{2\sigma^2}} \prod_j^N (1 + e^{b_j + \sum_i^M \frac{x_i w_{ij}}{\sigma^2}}) \quad (32)$$

$$(33)$$

Choose the cost function

Now we don't necessarily have training data (unless we generate it by using some other method). However, what we do have is the variational principle which allows us to obtain the ground state wave function by minimizing the expectation value of the energy of a trial wavefunction (corresponding to the untrained NQS). Similarly to the traditional variational Monte Carlo method then, it is the local energy we wish to minimize.

The gradient to use for the stochastic gradient descent procedure is then

$$G_i = \frac{\partial \langle E_L \rangle}{\partial \theta_i} = 2(\langle E_L \frac{1}{\Psi} \frac{\partial \Psi}{\partial \theta_i} \rangle - \langle E_L \rangle \langle \frac{1}{\Psi} \frac{\partial \Psi}{\partial \theta_i} \rangle) \quad (34)$$

Where the local energy is given by

$$E_L = \frac{1}{\Psi} \hat{H} \Psi \quad (35)$$

The physical system

As described in more detail in the project, we start by investigating a harmonic oscillator system, with the Hamiltonian given by

$$\hat{H} = \sum_p^P \left(-\frac{1}{2} \nabla_p^2 + \frac{1}{2} \omega^2 r_p^2 \right) + \sum_{p < q} \frac{1}{r_{pq}} \quad (36)$$