Week 16 April 17-21: Neural networks and project 2

Morten Hjorth-Jensen Email morten.hjorth-jensen@fys.uio.no[1,2]

[1]Department of Physics and Center fo Computing in Science Education, University of Oslo, Oslo, Norway
[2]Department of Physics and Astronomy and Facility for Rare Ion Beams, Michigan State University, East Lansing, Michigan, USA

Apr 19, 2023

Overview of week 16, April 17-21

1. Neural Networks and Boltzmann Machines, introduction to Boltzmann machines
2. Reading recommendation: Goodfellow et al chapters 17, 18 and 20.1-20.7

- Video of lecture TBA
- Handwritten notes

Schedule. Since the majority of you plan to work on neural networks and Boltzmann machines, we will have lectures on this topic for the rest of the semester. We meet each Thursday at 2.15pm for theory lectures and discussions till approx 4pm. The remaining three hours are dedicated to project work.

Alternatives for project 2

1. Fermion VMC, continuation of project 1
2. Deep learning applied to project 1, either neural networks or Boltzmann machines
3. Hartree-Fock theory and time-dependent theories

4. Many-body methods like coupled-cluster theory or other many-body methods
5. Quantum computing
6. Suggestions from you

Boltzmann Machines

Why use a generative model rather than the more well known discriminative deep neural networks (DNN)?

- Discriminitave methods have several limitations: They are mainly 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. As an example
    1. A model for images could learn to draw new examples of cats and dogs, given a training dataset of images of cats and dogs.
    2. Generate a sample of an ordered or disordered Ising model phase, having been given samples of such phases.
    3. Model the trial function for Monte Carlo calculations
- 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.

A BM is what we would call an undirected probabilistic graphical model with stochastic continuous or discrete units.

It is interpreted as a stochastic recurrent neural network where the state of each unit(neurons/nodes) depends on the units it is connected to. The weights in the network represent thus the strength of the interaction between various units/nodes.

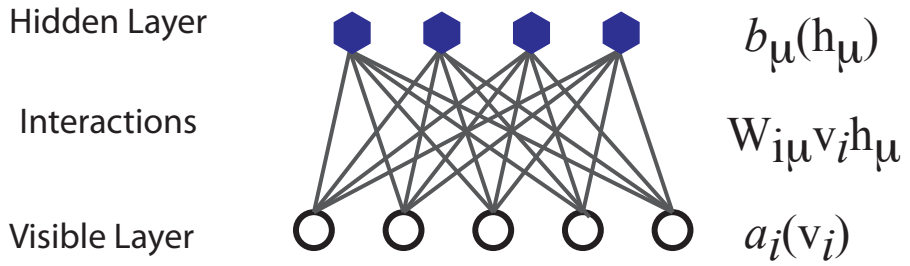
It turns into a Hopfield network if we choose deterministic rather than stochastic units. In contrast to a Hopfield network, a BM is a so-called generative model. It allows us to generate new samples from the learned distribution.

A standard BM network is divided into a set of observable and visible units $\hat{x}$ and a set of unknown hidden units/nodes $\hat{h}$.

Additionally there can be bias nodes for the hidden and visible layers. These biases are normally set to 1.

BMs are stackable, meaning they cwe can train a BM which serves as input to another BM. We can construct deep networks for learning complex PDFs. The layers can be trained one after another, a feature which makes them popular in deep learning

However, they are often hard to train. This leads to the introduction of so-called restricted BMs, or RBMS. Here we take away all lateral connections between nodes in the visible layer as well as connections between nodes in the hidden layer. The network is illustrated in the figure below.



The network

The network layers:

1. A function **x** that 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.

2. The function **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:

1. The Hubbard-Stratonovich transformation
2. The introduction of ghost fields in gauge theory
3. Shadow wave functions in Quantum Monte Carlo simulations

The network parameters, to be optimized/learned:

1. **a** represents the visible bias, a vector of same length as **x**.
2. **b** represents the hidden bias, a vector of same lenght as **h**.
3. W represents the interaction weights, a matrix of size $M \times N$.

Joint distribution. The restricted Boltzmann machine is described by a Boltzmann distribution

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

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

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

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

Network Elements, the energy function. 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 **a**, **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 $E(\mathbf{x}, \mathbf{h})$. The connection between the nodes in the two layers is given by the weights w_{ij}.

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

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

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

Gaussian-Binary RBM: Another varient is the RBM where the visible units are Gaussian while the hidden units remain binary:

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

1. RBMs are Useful when we model continuous data (i.e., we wish $\mathbf{x}$ to be continuous)
2. 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:

1. Softmax and multinomial units
2. Gaussian visible and hidden units
3. Binomial units
4. Rectified linear units

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 $\boldsymbol{\theta} = a_1, ..., a_M, b_1, ..., b_N, w_{11}, ..., w_{MN}$, the log-likelihood is given by

$$\begin{aligned} \mathcal{L}(\{\theta_i\}) &= \langle \log P_\theta(\boldsymbol{x}) \rangle_{data} && (5) \\ &= -\langle E(\boldsymbol{x}; \{\theta_i\}) \rangle_{data} - \log Z(\{\theta_i\}), && (6) \end{aligned}$$

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

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla \mathcal{C}(\boldsymbol{\theta}_k) \tag{7}$$

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

$$\frac{\partial \mathcal{C}(\{\theta_i\})}{\partial \theta_i} = \langle \frac{\partial E(\boldsymbol{x}; \theta_i)}{\partial \theta_i} \rangle_{data} + \frac{\partial \log Z(\{\theta_i\})}{\partial \theta_i} \tag{8}$$

$$= \langle O_i(\boldsymbol{x}) \rangle_{data} - \langle O_i(\boldsymbol{x}) \rangle_{model}, \tag{9}$$

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

$$O_i(\boldsymbol{x}) = \frac{\partial E(\boldsymbol{x}; \theta_i)}{\partial \theta_i}, \tag{10}$$

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

$$\langle O_i(\boldsymbol{x}) \rangle_{model} = \mathrm{Tr} P_\theta(\boldsymbol{x}) O_i(\boldsymbol{x}) = -\frac{\partial \log Z(\{\theta_i\})}{\partial \theta_i}. \tag{11}$$

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

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial w_{ij}} = \langle x_i h_j \rangle_{data} - \langle x_i h_j \rangle_{model} \tag{12}$$

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial a_{ij}} = \langle x_i \rangle_{data} - \langle x_i \rangle_{model} \tag{13}$$

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial b_{ij}} = \langle h_i \rangle_{data} - \langle h_i \rangle_{model}. \tag{14}$$

$$\tag{15}$$

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.

Kullback-Leibler relative entropy. When the goal of the training is to approximate a probability distribution, as it is in generative modeling, another relevant measure is the **Kullback-Leibler divergence**, also known as the relative entropy or Shannon entropy. It is a non-symmetric measure of the dissimilarity between two probability density functions p and q. If p is the unkown probability which we approximate with q, we can measure the difference by

$$\mathrm{KL}(p||q) = \int_{-\infty}^{\infty} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\boldsymbol{x}. \tag{16}$$

Thus, the Kullback-Leibler divergence between the distribution of the training data $f(\boldsymbol{x})$ and the model distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ is

$$\begin{aligned}
\mathrm{KL}(f(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\theta})) &= \int_{-\infty}^{\infty} f(\boldsymbol{x}) \log \frac{f(\boldsymbol{x})}{p(\boldsymbol{x}|\boldsymbol{\theta})} d\boldsymbol{x} && (17)\\
&= \int_{-\infty}^{\infty} f(\boldsymbol{x}) \log f(\boldsymbol{x}) d\boldsymbol{x} - \int_{-\infty}^{\infty} f(\boldsymbol{x}) \log p(\boldsymbol{x}|\boldsymbol{\theta}) d\boldsymbol{x} && (18)\\
&= \langle \log f(\boldsymbol{x}) \rangle_{f(\boldsymbol{x})} - \langle \log p(\boldsymbol{x}|\boldsymbol{\theta}) \rangle_{f(\boldsymbol{x})} && (19)\\
&= \langle \log f(\boldsymbol{x}) \rangle_{data} + \langle E(\boldsymbol{x}) \rangle_{data} + \log Z && (20)\\
&= \langle \log f(\boldsymbol{x}) \rangle_{data} + \mathcal{C}_{LL}. && (21)
\end{aligned}$$

The first term is constant with respect to $\boldsymbol{\theta}$ since $f(\boldsymbol{x})$ is independent of $\boldsymbol{\theta}$. Thus the Kullback-Leibler Divergence is minimal when the second term is minimal. The second term is the log-likelihood cost function, hence minimizing the Kullback-Leibler divergence is equivalent to maximizing the log-likelihood.

To further understand generative models it is useful to study the gradient of the cost function which is needed in order to minimize it using methods like stochastic gradient descent.

The partition function is the generating function of expectation values, in particular there are mathematical relationships between expectation values and the log-partition function. In this case we have

$$\langle \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} \rangle_{model} = \int p(\boldsymbol{x}|\boldsymbol{\theta}) \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} d\boldsymbol{x} = -\frac{\partial \log Z(\theta_i)}{\partial \theta_i}. \tag{22}$$

Here $\langle \cdot \rangle_{model}$ is the expectation value over the model probability distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$.

Setting up for gradient descent calculations

Using the previous relationship we can express the gradient of the cost function as

$$\frac{\partial \mathcal{C}_{LL}}{\partial \theta_i} = \langle \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} \rangle_{data} + \frac{\partial \log Z(\theta_i)}{\partial \theta_i} \tag{23}$$

$$= \langle \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} \rangle_{data} - \langle \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i} \rangle_{model} \tag{24}$$

$$\tag{25}$$

This expression shows that the gradient of the log-likelihood cost function is a **difference of moments**, with one calculated from the data and one calculated from the model. The data-dependent term is called the **positive phase** and the model-dependent term is called the **negative phase** of the gradient. We see now that minimizing the cost function results in lowering the energy of configurations $\boldsymbol{x}$ near points in the training data and increasing the energy of configurations not observed in the training data. That means we increase the model's probability of configurations similar to those in the training data.

The gradient of the cost function also demonstrates why gradients of unsupervised, generative models must be computed differently from for those of for example FNNs. While the data-dependent expectation value is easily calculated based on the samples $\boldsymbol{x}_i$ in the training data, we must sample from the model in order to generate samples from which to caclulate the model-dependent term. We sample from the model by using MCMC-based methods. We can not sample from the model directly because the partition function Z is generally intractable.

As in supervised machine learning problems, the goal is also here to perform well on **unseen** data, that is to have good generalization from the training data. The distribution $f(x)$ we approximate is not the **true** distribution we wish to estimate, it is limited to the training data. Hence, in unsupervised training as well it is important to prevent overfitting to the training data. Thus it is common to add regularizers to the cost function in the same manner as we discussed for say linear regression.

Mathematical details. Because we are restricted to potential functions which are positive it is convenient to express them as exponentials, so that

$$\phi_C(\boldsymbol{x}_C) = e^{-E_C(\boldsymbol{x}_C)} \tag{26}$$

where $E(\boldsymbol{x}_C)$ is called an *energy function*, and the exponential representation is the *Boltzmann distribution*. The joint distribution is defined as the product of potentials.

The joint distribution of the random variables is then

$$
\begin{aligned}
p(\boldsymbol{x}) =& \frac{1}{Z}\prod_{C}\phi_C(\boldsymbol{x}_C) \\
=& \frac{1}{Z}\prod_{C} e^{-E_C(\boldsymbol{x}_C)} \\
=& \frac{1}{Z} e^{-\sum_C E_C(\boldsymbol{x}_C)} \\
=& \frac{1}{Z} e^{-E(\boldsymbol{x})}.
\end{aligned} \tag{27}
$$

$$
p_{BM}(\boldsymbol{x},\boldsymbol{h}) = \frac{1}{Z_{BM}} e^{-\frac{1}{T}E_{BM}(\boldsymbol{x},\boldsymbol{h})}, \tag{28}
$$

with the partition function

$$
Z_{BM} = \int\int e^{-\frac{1}{T}E_{BM}(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{h}})} d\tilde{\boldsymbol{x}} d\tilde{\boldsymbol{h}}. \tag{29}
$$

T is a physics-inspired parameter named temperature and will be assumed to be 1 unless otherwise stated. The energy function of the Boltzmann machine determines the interactions between the nodes and is defined

$$
\begin{aligned}
E_{BM}(\boldsymbol{x},\boldsymbol{h}) =& -\sum_{i,k}^{M,K} a_i^k \alpha_i^k(x_i) - \sum_{j,l}^{N,L} b_j^l \beta_j^l(h_j) - \sum_{i,j,k,l}^{M,N,K,L} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(h_j) \\
&- \sum_{i,m=i+1,k}^{M,M,K} \alpha_i^k(x_i) v_{im}^k \alpha_m^k(x_m) - \sum_{j,n=j+1,l}^{N,N,L} \beta_j^l(h_j) u_{jn}^l \beta_n^l(h_n).
\end{aligned} \tag{30}
$$

Here $\alpha_i^k(x_i)$ and $\beta_j^l(h_j)$ are one-dimensional transfer functions or mappings from the given input value to the desired feature value. They can be arbitrary functions of the input variables and are independent of the parameterization (parameters referring to weight and biases), meaning they are not affected by training of the model. The indices k and l indicate that there can be multiple transfer functions per variable. Furthermore, a_i^k and b_j^l are the visible and hidden bias. w_{ij}^{kl} are weights of the **inter-layer** connection terms which connect visible and hidden units. v_{im}^k and u_{jn}^l are weights of the **intra-layer** connection terms which connect the visible units to each other and the hidden units to each other, respectively.

We remove the intra-layer connections by setting v_{im} and u_{jn} to zero. The expression for the energy of the RBM is then

$$
E_{RBM}(\boldsymbol{x},\boldsymbol{h}) = -\sum_{i,k}^{M,K} a_i^k \alpha_i^k(x_i) - \sum_{j,l}^{N,L} b_j^l \beta_j^l(h_j) - \sum_{i,j,k,l}^{M,N,K,L} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(h_j). \tag{31}
$$

resulting in

$$
\begin{aligned}
P_{RBM}(\boldsymbol{x}) =& \int P_{RBM}(\boldsymbol{x}, \tilde{\boldsymbol{h}}) d\tilde{\boldsymbol{h}} \\
=& \frac{1}{Z_{RBM}} \int e^{-E_{RBM}(\boldsymbol{x}, \tilde{\boldsymbol{h}})} d\tilde{\boldsymbol{h}} \\
=& \frac{1}{Z_{RBM}} \int e^{\sum_{i,k} a_i^k \alpha_i^k(x_i) + \sum_{j,l} b_j^l \beta_j^l(\tilde{h}_j) + \sum_{i,j,k,l} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(\tilde{h}_j)} d\tilde{\boldsymbol{h}} \\
=& \frac{1}{Z_{RBM}} e^{\sum_{i,k} a_i^k \alpha_i^k(x_i)} \int \prod_j^N e^{\sum_l b_j^l \beta_j^l(\tilde{h}_j) + \sum_{i,k,l} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(\tilde{h}_j)} d\tilde{\boldsymbol{h}} \\
=& \frac{1}{Z_{RBM}} e^{\sum_{i,k} a_i^k \alpha_i^k(x_i)} \Bigg(\int e^{\sum_l b_1^l \beta_1^l(\tilde{h}_1) + \sum_{i,k,l} \alpha_i^k(x_i) w_{i1}^{kl} \beta_1^l(\tilde{h}_1)} d\tilde{h}_1 \\
& \times \int e^{\sum_l b_2^l \beta_2^l(\tilde{h}_2) + \sum_{i,k,l} \alpha_i^k(x_i) w_{i2}^{kl} \beta_2^l(\tilde{h}_2)} d\tilde{h}_2 \\
& \times ... \\
& \times \int e^{\sum_l b_N^l \beta_N^l(\tilde{h}_N) + \sum_{i,k,l} \alpha_i^k(x_i) w_{iN}^{kl} \beta_N^l(\tilde{h}_N)} d\tilde{h}_N \Bigg) \\
=& \frac{1}{Z_{RBM}} e^{\sum_{i,k} a_i^k \alpha_i^k(x_i)} \prod_j^N \int e^{\sum_l b_j^l \beta_j^l(\tilde{h}_j) + \sum_{i,k,l} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(\tilde{h}_j)} d\tilde{h}_j
\end{aligned}
\tag{32}
$$

Similarly

$$
\begin{aligned}
P_{RBM}(\boldsymbol{h}) =& \frac{1}{Z_{RBM}} \int e^{-E_{RBM}(\tilde{\boldsymbol{x}}, \boldsymbol{h})} d\tilde{\boldsymbol{x}} \\
=& \frac{1}{Z_{RBM}} e^{\sum_{j,l} b_j^l \beta_j^l(h_j)} \prod_i^M \int e^{\sum_k a_i^k \alpha_i^k(\tilde{x}_i) + \sum_{j,k,l} \alpha_i^k(\tilde{x}_i) w_{ij}^{kl} \beta_j^l(h_j)} d\tilde{x}_i
\end{aligned}
\tag{33}
$$

Using Bayes theorem

$$
\begin{aligned}
P_{RBM}(\boldsymbol{h}|\boldsymbol{x}) =& \frac{P_{RBM}(\boldsymbol{x}, \boldsymbol{h})}{P_{RBM}(\boldsymbol{x})} \\
=& \frac{\frac{1}{Z_{RBM}} e^{\sum_{i,k} a_i^k \alpha_i^k(x_i) + \sum_{j,l} b_j^l \beta_j^l(h_j) + \sum_{i,j,k,l} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(h_j)}}{\frac{1}{Z_{RBM}} e^{\sum_{i,k} a_i^k \alpha_i^k(x_i)} \prod_j^N \int e^{\sum_l b_j^l \beta_j^l(\tilde{h}_j) + \sum_{i,k,l} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(\tilde{h}_j)} d\tilde{h}_j} \\
=& \prod_j^N \frac{e^{\sum_l b_j^l \beta_j^l(h_j) + \sum_{i,k,l} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(h_j)}}{\int e^{\sum_l b_j^l \beta_j^l(\tilde{h}_j) + \sum_{i,k,l} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(\tilde{h}_j)} d\tilde{h}_j}
\end{aligned}
\tag{34}
$$

Similarly

$$P_{RBM}(\boldsymbol{x}|\boldsymbol{h}) = \frac{P_{RBM}(\boldsymbol{x},\boldsymbol{h})}{P_{RBM}(\boldsymbol{h})}$$
$$= \prod_i^M \frac{e^{\sum_k a_i^k \alpha_i^k(x_i) + \sum_{j,k,l} \alpha_i^k(x_i) w_{ij}^{kl} \beta_j^l(h_j)}}{\int e^{\sum_k a_i^k \alpha_i^k(\tilde{x}_i) + \sum_{j,k,l} \alpha_i^k(\tilde{x}_i) w_{ij}^{kl} \beta_j^l(h_j)} d\tilde{x}_i} \tag{35}$$

The original RBM had binary visible and hidden nodes. They were showned to be universal approximators of discrete distributions. It was also shown that adding hidden units yields strictly improved modelling power. The common choice of binary values are 0 and 1. However, in some physics applications, -1 and 1 might be a more natural choice. We will here use 0 and 1.

$$E_{BB}(\boldsymbol{x}, \mathbf{h}) = -\sum_i^M x_i a_i - \sum_j^N b_j h_j - \sum_{i,j}^{M,N} x_i w_{ij} h_j. \tag{36}$$

$$p_{BB}(\boldsymbol{x},\boldsymbol{h}) = \frac{1}{Z_{BB}} e^{\sum_i^M a_i x_i + \sum_j^N b_j h_j + \sum_{ij}^{M,N} x_i w_{ij} h_j} \tag{37}$$
$$= \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a} + \boldsymbol{b}^T \boldsymbol{h} + \boldsymbol{x}^T \boldsymbol{W} \boldsymbol{h}} \tag{38}$$

with the partition function

$$Z_{BB} = \sum_{\boldsymbol{x},\boldsymbol{h}} e^{\boldsymbol{x}^T \boldsymbol{a} + \boldsymbol{b}^T \boldsymbol{h} + \boldsymbol{x}^T \boldsymbol{W} \boldsymbol{h}}. \tag{39}$$

Marginal Probability Density Functions. In order to find the probability of any configuration of the visible units we derive the marginal probability density function.

$$
\begin{aligned}
p_{BB}(\boldsymbol{x}) =& \sum_{\boldsymbol{h}} p_{BB}(\boldsymbol{x}, \boldsymbol{h}) \qquad (40)\\
=& \frac{1}{Z_{BB}} \sum_{\boldsymbol{h}} e^{\boldsymbol{x}^T \boldsymbol{a} + \boldsymbol{b}^T \boldsymbol{h} + \boldsymbol{x}^T \boldsymbol{W} \boldsymbol{h}} \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \sum_{\boldsymbol{h}} e^{\sum_j^N (b_j + \boldsymbol{x}^T \boldsymbol{w}_{*j}) h_j} \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \sum_{\boldsymbol{h}} \prod_j^N e^{(b_j + \boldsymbol{x}^T \boldsymbol{w}_{*j}) h_j} \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \Bigg(\sum_{h_1} e^{(b_1 + \boldsymbol{x}^T \boldsymbol{w}_{*1}) h_1} \times \sum_{h_2} e^{(b_2 + \boldsymbol{x}^T \boldsymbol{w}_{*2}) h_2} \times \\
& \ldots \times \sum_{h_2} e^{(b_N + \boldsymbol{x}^T \boldsymbol{w}_{*N}) h_N} \Bigg) \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \prod_j^N \sum_{h_j} e^{(b_j + \boldsymbol{x}^T \boldsymbol{w}_{*j}) h_j} \\
=& \frac{1}{Z_{BB}} e^{\boldsymbol{x}^T \boldsymbol{a}} \prod_j^N (1 + e^{b_j + \boldsymbol{x}^T \boldsymbol{w}_{*j}}). \qquad (41)
\end{aligned}
$$

A similar derivation yields the marginal probability of the hidden units

$$
p_{BB}(\boldsymbol{h}) = \frac{1}{Z_{BB}} e^{\boldsymbol{b}^T \boldsymbol{h}} \prod_i^M (1 + e^{a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h}}). \qquad (42)
$$

Conditional Probability Density Functions. We derive the probability of the hidden units given the visible units using Bayes' rule

$$
\begin{aligned}
p_{BB}(\boldsymbol{h}|\boldsymbol{x}) =& \frac{p_{BB}(\boldsymbol{x},\boldsymbol{h})}{p_{BB}(\boldsymbol{x})} \\
=& \frac{\frac{1}{Z_{BB}} e^{\boldsymbol{x}^T\boldsymbol{a}+\boldsymbol{b}^T\boldsymbol{h}+\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{h}}}{\frac{1}{Z_{BB}} e^{\boldsymbol{x}^T\boldsymbol{a}} \prod_j^N (1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}})} \\
=& \frac{e^{\boldsymbol{x}^T\boldsymbol{a}} e^{\sum_j^N (b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})h_j}}{e^{\boldsymbol{x}^T\boldsymbol{a}} \prod_j^N (1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}})} \\
=& \prod_j^N \frac{e^{(b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})h_j}}{1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}}} \\
=& \prod_j^N p_{BB}(h_j|\boldsymbol{x}). \qquad (43)
\end{aligned}
$$

From this we find the probability of a hidden unit being "on" or "off":

$$
\begin{aligned}
p_{BB}(h_j=1|\boldsymbol{x}) =& \frac{e^{(b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})h_j}}{1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}}} \qquad (44) \\
=& \frac{e^{(b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})}}{1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}}} \qquad (45) \\
=& \frac{1}{1+e^{-(b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j})}}, \qquad (46)
\end{aligned}
$$

and

$$
p_{BB}(h_j=0|\boldsymbol{x}) = \frac{1}{1+e^{b_j+\boldsymbol{x}^T\boldsymbol{w}_{*j}}}. \qquad (47)
$$

Similarly we have that the conditional probability of the visible units given the hidden are

$$
\begin{aligned}
p_{BB}(\boldsymbol{x}|\boldsymbol{h}) =& \prod_i^M \frac{e^{(a_i+\boldsymbol{w}_{i*}^T\boldsymbol{h})x_i}}{1+e^{a_i+\boldsymbol{w}_{i*}^T\boldsymbol{h}}} \qquad (48) \\
=& \prod_i^M p_{BB}(x_i|\boldsymbol{h}). \qquad (49)
\end{aligned}
$$

$$
\begin{aligned}
p_{BB}(x_i=1|\boldsymbol{h}) =& \frac{1}{1+e^{-(a_i+\boldsymbol{w}_{i*}^T\boldsymbol{h})}} \qquad (50) \\
p_{BB}(x_i=0|\boldsymbol{h}) =& \frac{1}{1+e^{a_i+\boldsymbol{w}_{i*}^T\boldsymbol{h}}}. \qquad (51)
\end{aligned}
$$

Gaussian-Binary Restricted Boltzmann Machines. Inserting into the expression for $E_{RBM}(\boldsymbol{x}, \boldsymbol{h})$ in equation results in the energy

$$\begin{aligned} E_{GB}(\boldsymbol{x}, \boldsymbol{h}) =& \sum_{i}^{M} \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_{j}^{N} b_j h_j - \sum_{ij}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2} \\ =& ||\frac{\boldsymbol{x} - \boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 - \boldsymbol{b}^T \boldsymbol{h} - (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W} \boldsymbol{h}. \end{aligned} \tag{52}$$

Joint Probability Density Function.

$$\begin{aligned} p_{GB}(\boldsymbol{x}, \boldsymbol{h}) =& \frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T \boldsymbol{h} + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W} \boldsymbol{h}} \\ =& \frac{1}{Z_{GB}} e^{-\sum_i^M \frac{(x_i - a_i)^2}{2\sigma_i^2} + \sum_j^N b_j h_j + \sum_{ij}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2}} \\ =& \frac{1}{Z_{GB}} \prod_{ij}^{M,N} e^{-\frac{(x_i - a_i)^2}{2\sigma_i^2} + b_j h_j + \frac{x_i w_{ij} h_j}{\sigma_i^2}}, \end{aligned} \tag{53}$$

with the partition function given by

$$Z_{GB} = \int \sum_{\tilde{\boldsymbol{h}}}^{\tilde{\boldsymbol{H}}} e^{-||\frac{\tilde{\boldsymbol{x}}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T \tilde{\boldsymbol{h}} + (\frac{\tilde{\boldsymbol{x}}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W} \tilde{\boldsymbol{h}}} d\tilde{\boldsymbol{x}}. \tag{54}$$

Marginal Probability Density Functions. We proceed to find the marginal probability densitites of the Gaussian-binary RBM. We first marginalize over the binary hidden units to find $p_{GB}(\boldsymbol{x})$

$$\begin{aligned} p_{GB}(\boldsymbol{x}) =& \sum_{\tilde{\boldsymbol{h}}}^{\tilde{\boldsymbol{H}}} p_{GB}(\boldsymbol{x}, \tilde{\boldsymbol{h}}) \\ =& \frac{1}{Z_{GB}} \sum_{\tilde{\boldsymbol{h}}}^{\tilde{\boldsymbol{H}}} e^{-||\frac{\boldsymbol{x}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T \tilde{\boldsymbol{h}} + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W} \tilde{\boldsymbol{h}}} \\ =& \frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2} \prod_{j}^{N} (1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}). \end{aligned} \tag{55}$$

We next marginalize over the visible units. This is the first time we marginalize over continuous values. We rewrite the exponential factor dependent on $\boldsymbol{x}$ as a Gaussian function before we integrate in the last step.

$$
\begin{aligned}
p_{GB}(\boldsymbol{h}) &= \int p_{GB}(\tilde{\boldsymbol{x}}, \boldsymbol{h}) d\tilde{\boldsymbol{x}} \\
&= \frac{1}{Z_{GB}} \int e^{-||\frac{\tilde{\boldsymbol{x}}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T\boldsymbol{h} + (\frac{\tilde{\boldsymbol{x}}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W}\boldsymbol{h}} d\tilde{\boldsymbol{x}} \\
&= \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \int \prod_i^M e^{-\frac{(\tilde{x}_i - a_i)^2}{2\sigma_i^2} + \frac{\tilde{x}_i \boldsymbol{w}_{i*}^T \boldsymbol{h}}{\sigma_i^2}} d\tilde{\boldsymbol{x}} \\
&= \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \Bigg(\int e^{-\frac{(\tilde{x}_1 - a_1)^2}{2\sigma_1^2} + \frac{\tilde{x}_1 \boldsymbol{w}_{1*}^T \boldsymbol{h}}{\sigma_1^2}} d\tilde{x}_1 \\
&\quad \times \int e^{-\frac{(\tilde{x}_2 - a_2)^2}{2\sigma_2^2} + \frac{\tilde{x}_2 \boldsymbol{w}_{2*}^T \boldsymbol{h}}{\sigma_2^2}} d\tilde{x}_2 \\
&\quad \times ... \\
&\quad \times \int e^{-\frac{(\tilde{x}_M - a_M)^2}{2\sigma_M^2} + \frac{\tilde{x}_M \boldsymbol{w}_{M*}^T \boldsymbol{h}}{\sigma_M^2}} d\tilde{x}_M \Bigg) \\
&= \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \int e^{-\frac{(\tilde{x}_i - a_i)^2 - 2\tilde{x}_i \boldsymbol{w}_{i*}^T \boldsymbol{h}}{2\sigma_i^2}} d\tilde{x}_i \\
&= \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \int e^{-\frac{\tilde{x}_i^2 - 2\tilde{x}_i(a_i + \tilde{x}_i \boldsymbol{w}_{i*}^T \boldsymbol{h}) + a_i^2}{2\sigma_i^2}} d\tilde{x}_i \\
&= \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \int e^{-\frac{\tilde{x}_i^2 - 2\tilde{x}_i(a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h}) + (a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h})^2 - (a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h})^2 + a_i^2}{2\sigma_i^2}} d\tilde{x}_i \\
&= \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \int e^{-\frac{(\tilde{x}_i - (a_i + \boldsymbol{w}_{i*}^T \boldsymbol{h}))^2 - a_i^2 - 2a_i \boldsymbol{w}_{i*}^T \boldsymbol{h} - (\boldsymbol{w}_{i*}^T \boldsymbol{h})^2 + a_i^2}{2\sigma_i^2}} d\tilde{x}_i \\
&= \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M e^{\frac{2a_i \boldsymbol{w}_{i*}^T \boldsymbol{h} + (\boldsymbol{w}_{i*}^T \boldsymbol{h})^2}{2\sigma_i^2}} \int e^{-\frac{(\tilde{x}_i - a_i - \boldsymbol{w}_{i*}^T \boldsymbol{h})^2}{2\sigma_i^2}} d\tilde{x}_i \\
&= \frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \sqrt{2\pi\sigma_i^2} e^{\frac{2a_i \boldsymbol{w}_{i*}^T \boldsymbol{h} + (\boldsymbol{w}_{i*}^T \boldsymbol{h})^2}{2\sigma_i^2}}. \qquad (56)
\end{aligned}
$$

Conditional Probability Density Functions. We finish by deriving the conditional probabilities.

$$\begin{aligned} p_{GB}(\boldsymbol{h}|\boldsymbol{x}) =& \frac{p_{GB}(\boldsymbol{x},\boldsymbol{h})}{p_{GB}(\boldsymbol{x})} \\ =& \frac{\frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T\boldsymbol{h} + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W}\boldsymbol{h}}}{\frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2} \prod_j^N (1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}})} \\ =& \prod_j^N \frac{e^{(b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}) h_j}}{1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}} \\ =& \prod_j^N p_{GB}(h_j|\boldsymbol{x}). \end{aligned} \tag{57}$$

The conditional probability of a binary hidden unit h_j being on or off again takes the form of a sigmoid function

$$\begin{aligned} p_{GB}(h_j = 1|\boldsymbol{x}) =& \frac{e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}}{1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}} \\ =& \frac{1}{1 + e^{-b_j - (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}} \end{aligned} \tag{58}$$

$$p_{GB}(h_j = 0|\boldsymbol{x}) = \frac{1}{1 + e^{b_j + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{w}_{*j}}}. \tag{59}$$

The conditional probability of the continuous $\boldsymbol{x}$ now has another form, however.

$$
\begin{aligned}
p_{GB}(\boldsymbol{x}|\boldsymbol{h}) =& \frac{p_{GB}(\boldsymbol{x},\boldsymbol{h})}{p_{GB}(\boldsymbol{h})} \\
=& \frac{\frac{1}{Z_{GB}} e^{-||\frac{\boldsymbol{x}-\boldsymbol{a}}{2\boldsymbol{\sigma}}||^2 + \boldsymbol{b}^T\boldsymbol{h} + (\frac{\boldsymbol{x}}{\boldsymbol{\sigma}^2})^T \boldsymbol{W}\boldsymbol{h}}}{\frac{1}{Z_{GB}} e^{\boldsymbol{b}^T\boldsymbol{h}} \prod_i^M \sqrt{2\pi\sigma_i^2} e^{\frac{2a_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + (\boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}}} \\
=& \prod_i^M \frac{1}{\sqrt{2\pi\sigma_i^2}} \frac{e^{-\frac{(x_i-a_i)^2}{2\sigma_i^2} + \frac{x_i \boldsymbol{w}_{i*}^T\boldsymbol{h}}{2\sigma_i^2}}}{e^{\frac{2a_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + (\boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}}} \\
=& \prod_i^M \frac{1}{\sqrt{2\pi\sigma_i^2}} \frac{e^{-\frac{x_i^2 - 2a_i x_i + a_i^2 - 2x_i \boldsymbol{w}_{i*}^T\boldsymbol{h}}{2\sigma_i^2}}}{e^{\frac{2a_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + (\boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}}} \\
=& \prod_i^M \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{x_i^2 - 2a_i x_i + a_i^2 - 2x_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + 2a_i \boldsymbol{w}_{i*}^T\boldsymbol{h} + (\boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}} \\
=& \prod_i^M \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x_i - b_i - \boldsymbol{w}_{i*}^T\boldsymbol{h})^2}{2\sigma_i^2}} \\
=& \prod_i^M \mathcal{N}(x_i | b_i + \boldsymbol{w}_{i*}^T\boldsymbol{h}, \sigma_i^2) \qquad (60) \\
\Rightarrow p_{GB}(x_i|\boldsymbol{h}) =& \mathcal{N}(x_i | b_i + \boldsymbol{w}_{i*}^T\boldsymbol{h}, \sigma_i^2). \qquad (61)
\end{aligned}
$$

The form of these conditional probabilities explains the name "Gaussian" and the form of the Gaussian-binary energy function. We see that the conditional probability of x_i given $\boldsymbol{h}$ is a normal distribution with mean $b_i + \boldsymbol{w}_{i*}^T\boldsymbol{h}$ and variance σ_i^2.

Neural Quantum States

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

$$
F_{rbm}(\boldsymbol{x}, \mathbf{h}) = \frac{1}{Z} e^{-\frac{1}{T_0} E(\boldsymbol{x}, \mathbf{h})} \qquad (62)
$$

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

$$
\begin{aligned}
F_{rbm}(\mathbf{x}) &= \sum_{\mathbf{h}} F_{rbm}(\mathbf{x}, \mathbf{h}) \qquad (63) \\
&= \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})} \qquad (64)
\end{aligned}
$$

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

$$\Psi(\mathbf{X}) = F_{rbm}(\mathbf{x}) \tag{65}$$

$$= \frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(\mathbf{x},\mathbf{h})} \tag{66}$$

$$= \frac{1}{Z}\sum_{\{h_j\}} e^{-\sum_i^M \frac{(x_i-a_i)^2}{2\sigma^2}+\sum_j^N b_j h_j+\sum_{i,j}^{M,N} \frac{x_i w_{ij} h_j}{\sigma^2}} \tag{67}$$

$$= \frac{1}{Z} e^{-\sum_i^M \frac{(x_i-a_i)^2}{2\sigma^2}} \prod_j^N (1 + e^{b_j+\sum_i^M \frac{x_i w_{ij}}{\sigma^2}}) \tag{68}$$

$$\tag{69}$$

The above wavefunction is the most general one because it allows for complex valued wavefunctions. However it fundamentally changes the probabilistic foundation of the RBM, because what is usually a probability in the RBM framework is now a an amplitude. This means that a lot of the theoretical framework usually used to interpret the model, i.e. graphical models, conditional probabilities, and Markov random fields, breaks down. If we assume the wavefunction to be postive definite, however, we can use the RBM to represent the squared wavefunction, and thereby a probability. This also makes it possible to sample from the model using Gibbs sampling, because we can obtain the conditional probabilities.

$$|\Psi(\mathbf{X})|^2 = F_{rbm}(\mathbf{X}) \tag{70}$$

$$\Rightarrow \Psi(\mathbf{X}) = \sqrt{F_{rbm}(\mathbf{X})} \tag{71}$$

$$= \frac{1}{\sqrt{Z}}\sqrt{\sum_{\{h_j\}} e^{-E(\mathbf{X},\mathbf{h})}} \tag{72}$$

$$= \frac{1}{\sqrt{Z}}\sqrt{\sum_{\{h_j\}} e^{-\sum_i^M \frac{(X_i-a_i)^2}{2\sigma^2}+\sum_j^N b_j h_j+\sum_{i,j}^{M,N}\frac{X_i w_{ij} h_j}{\sigma^2}}} \tag{73}$$

$$= \frac{1}{\sqrt{Z}} e^{-\sum_i^M \frac{(X_i-a_i)^2}{4\sigma^2}}\sqrt{\sum_{\{h_j\}}\prod_j^N e^{b_j h_j+\sum_i^M \frac{X_i w_{ij} h_j}{\sigma^2}}} \tag{74}$$

$$= \frac{1}{\sqrt{Z}} e^{-\sum_i^M \frac{(X_i-a_i)^2}{4\sigma^2}}\sqrt{\prod_j^N\sum_{h_j} e^{b_j h_j+\sum_i^M \frac{X_i w_{ij} h_j}{\sigma^2}}} \tag{75}$$

$$= \frac{1}{\sqrt{Z}} e^{-\sum_i^M \frac{(X_i-a_i)^2}{4\sigma^2}}\prod_j^N\sqrt{e^0 + e^{b_j+\sum_i^M \frac{X_i w_{ij}}{\sigma^2}}} \tag{76}$$

$$= \frac{1}{\sqrt{Z}} e^{-\sum_i^M \frac{(X_i-a_i)^2}{4\sigma^2}}\prod_j^N\sqrt{1 + e^{b_j+\sum_i^M \frac{X_i w_{ij}}{\sigma^2}}} \tag{77}$$

$$\tag{78}$$

Cost function. This is where we deviate from what is common in machine learning. Rather than defining a cost function based on some dataset, our cost function is the energy of the quantum mechanical system. From the variational principle we know that minizing this energy should lead to the ground state wavefunction. As stated previously the local energy is given by

$$E_L = \frac{1}{\Psi}\hat{\mathbf{H}}\Psi, \tag{79}$$

and the gradient is

$$G_i = \frac{\partial\langle E_L\rangle}{\partial\alpha_i} = 2(\langle E_L\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha_i}\rangle - \langle E_L\rangle\langle\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha_i}\rangle), \tag{80}$$

where $\alpha_i = a_1, ..., a_M, b_1, ..., b_N, w_{11}, ..., w_{MN}$.

We use that $\frac{1}{\Psi}\frac{\partial\Psi}{\partial\alpha_i} = \frac{\partial \ln\Psi}{\partial\alpha_i}$, and find

$$\ln\Psi(\mathbf{X}) = -\ln Z - \sum_m^M \frac{(X_m-a_m)^2}{2\sigma^2} + \sum_n^N \ln(1+e^{b_n+\sum_i^M \frac{X_i w_{in}}{\sigma^2}}). \tag{81}$$

This gives

$$\frac{\partial}{\partial a_m}\ln\Psi = \frac{1}{\sigma^2}(X_m - a_m) \tag{82}$$

$$\frac{\partial}{\partial b_n}\ln\Psi = \frac{1}{e^{-b_n-\frac{1}{\sigma^2}\sum_i^M X_i w_{in}}+1} \tag{83}$$

$$\frac{\partial}{\partial w_{mn}}\ln\Psi = \frac{X_m}{\sigma^2(e^{-b_n-\frac{1}{\sigma^2}\sum_i^M X_i w_{in}}+1)}. \tag{84}$$

If $\Psi = \sqrt{F_{rbm}}$ we have

$$\ln\Psi(\mathbf{X}) = -\frac{1}{2}\ln Z - \sum_m^M \frac{(X_m - a_m)^2}{4\sigma^2} + \frac{1}{2}\sum_n^N \ln(1 + e^{b_n+\sum_i^M \frac{X_i w_{in}}{\sigma^2}}), \tag{85}$$

which results in

$$\frac{\partial}{\partial a_m}\ln\Psi = \frac{1}{2\sigma^2}(X_m - a_m) \tag{86}$$

$$\frac{\partial}{\partial b_n}\ln\Psi = \frac{1}{2(e^{-b_n-\frac{1}{\sigma^2}\sum_i^M X_i w_{in}}+1)} \tag{87}$$

$$\frac{\partial}{\partial w_{mn}}\ln\Psi = \frac{X_m}{2\sigma^2(e^{-b_n-\frac{1}{\sigma^2}\sum_i^M X_i w_{in}}+1)}. \tag{88}$$

Let us assume again that our Hamiltonian is

$$\hat{\mathbf{H}} = \sum_p^P(-\frac{1}{2}\nabla_p^2 + \frac{1}{2}\omega^2 r_p^2) + \sum_{p<q}\frac{1}{r_{pq}}, \tag{89}$$

where the first summation term represents the standard harmonic oscillator part and the latter the repulsive interaction between two electrons. Natural units ($\hbar = c = e = m_e = 1$) are used, and P is the number of particles. This gives us the following expression for the local energy (D being the number of dimensions)

$$E_L = \frac{1}{\Psi}\mathbf{H}\Psi \tag{90}$$

$$= \frac{1}{\Psi}(\sum_{p}^{P}(-\frac{1}{2}\nabla_p^2 + \frac{1}{2}\omega^2 r_p^2) + \sum_{p<q}\frac{1}{r_{pq}})\Psi \tag{91}$$

$$= -\frac{1}{2}\frac{1}{\Psi}\sum_{p}^{P}\nabla_p^2\Psi + \frac{1}{2}\omega^2\sum_{p}^{P}r_p^2 + \sum_{p<q}\frac{1}{r_{pq}} \tag{92}$$

$$= -\frac{1}{2}\frac{1}{\Psi}\sum_{p}^{P}\sum_{d}^{D}\frac{\partial^2\Psi}{\partial x_{pd}^2} + \frac{1}{2}\omega^2\sum_{p}^{P}r_p^2 + \sum_{p<q}\frac{1}{r_{pq}} \tag{93}$$

$$= \frac{1}{2}\sum_{p}^{P}\sum_{d}^{D}(-(\frac{\partial}{\partial x_{pd}}\ln\Psi)^2 - \frac{\partial^2}{\partial x_{pd}^2}\ln\Psi + \omega^2 x_{pd}^2) + \sum_{p<q}\frac{1}{r_{pq}}. \tag{94}$$

$$\tag{95}$$

Letting each visible node in the Boltzmann machine represent one coordinate of one particle, we obtain

$$E_L = \frac{1}{2}\sum_{m}^{M}(-(\frac{\partial}{\partial v_m}\ln\Psi)^2 - \frac{\partial^2}{\partial v_m^2}\ln\Psi + \omega^2 v_m^2) + \sum_{p<q}\frac{1}{r_{pq}}, \tag{96}$$

where we have that

$$\frac{\partial}{\partial x_m}\ln\Psi = -\frac{1}{\sigma^2}(x_m - a_m) + \frac{1}{\sigma^2}\sum_{n}^{N}\frac{w_{mn}}{e^{-b_n - \frac{1}{\sigma^2}\sum_i^M x_i w_{in}} + 1} \tag{97}$$

$$\frac{\partial^2}{\partial x_m^2}\ln\Psi = -\frac{1}{\sigma^2} + \frac{1}{\sigma^4}\sum_{n}^{N}\omega_{mn}^2\frac{e^{b_n + \frac{1}{\sigma^2}\sum_i^M x_i w_{in}}}{(e^{b_n + \frac{1}{\sigma^2}\sum_i^M x_i w_{in}} + 1)^2}. \tag{98}$$

We now have all the expressions neeeded to calculate the gradient of the expected local energy with respect to the RBM parameters $\frac{\partial\langle E_L\rangle}{\partial\alpha_i}$.

If we use $\Psi = \sqrt{F_{rbm}}$ we obtain

$$\frac{\partial}{\partial x_m}\ln\Psi = -\frac{1}{2\sigma^2}(x_m - a_m) + \frac{1}{2\sigma^2}\sum_{n}^{N}\frac{w_{mn}}{e^{-b_n - \frac{1}{\sigma^2}\sum_i^M x_i w_{in}} + 1} \tag{99}$$

$$\frac{\partial^2}{\partial x_m^2}\ln\Psi = -\frac{1}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{n}^{N}\omega_{mn}^2\frac{e^{b_n + \frac{1}{\sigma^2}\sum_i^M x_i w_{in}}}{(e^{b_n + \frac{1}{\sigma^2}\sum_i^M x_i w_{in}} + 1)^2}. \tag{100}$$

The difference between this equation and the previous one is that we multiply by a factor 1/2.

Python version for the two non-interacting particles

```
# 2-electron VMC code for 2dim quantum dot with importance sampling
# Using gaussian rng for new positions and Metropolis- Hastings
# Added restricted boltzmann machine method for dealing with the wavefunction
# RBM code based heavily off of:
# https://github.com/CompPhysics/ComputationalPhysics2/tree/gh-pages/doc/Programs/BoltzmannMachin
from math import exp, sqrt
from random import random, seed, normalvariate
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import sys



# Trial wave function for the 2-electron quantum dot in two dims
def WaveFunction(r,a,b,w):
    sigma=1.0
    sig2 = sigma**2
    Psi1 = 0.0
    Psi2 = 1.0
    Q = Qfac(r,b,w)

    for iq in range(NumberParticles):
        for ix in range(Dimension):
            Psi1 += (r[iq,ix]-a[iq,ix])**2

    for ih in range(NumberHidden):
        Psi2 *= (1.0 + np.exp(Q[ih]))

    Psi1 = np.exp(-Psi1/(2*sig2))

    return Psi1*Psi2

# Local energy  for the 2-electron quantum dot in two dims, using analytical local energy
def LocalEnergy(r,a,b,w):
    sigma=1.0
    sig2 = sigma**2
    locenergy = 0.0

    Q = Qfac(r,b,w)

    for iq in range(NumberParticles):
        for ix in range(Dimension):
            sum1 = 0.0
            sum2 = 0.0
            for ih in range(NumberHidden):
                sum1 += w[iq,ix,ih]/(1+np.exp(-Q[ih]))
                sum2 += w[iq,ix,ih]**2 * np.exp(Q[ih]) / (1.0 + np.exp(Q[ih]))**2

            dlnpsi1 = -(r[iq,ix] - a[iq,ix]) /sig2 + sum1/sig2
            dlnpsi2 = -1/sig2 + sum2/sig2**2
            locenergy += 0.5*(-dlnpsi1*dlnpsi1 - dlnpsi2 + r[iq,ix]**2)

    if(interaction==True):
        for iq1 in range(NumberParticles):
            for iq2 in range(iq1):
                distance = 0.0
```

```
                for ix in range(Dimension):
                    distance += (r[iq1,ix] - r[iq2,ix])**2

                locenergy += 1/sqrt(distance)

    return locenergy

# Derivate of wave function ansatz as function of variational parameters
def DerivativeWFansatz(r,a,b,w):

    sigma=1.0
    sig2 = sigma**2

    Q = Qfac(r,b,w)

    WfDer = np.empty((3,),dtype=object)
    WfDer = [np.copy(a),np.copy(b),np.copy(w)]

    WfDer[0] = (r-a)/sig2
    WfDer[1] = 1 / (1 + np.exp(-Q))

    for ih in range(NumberHidden):
        WfDer[2][:,:,ih] = w[:,:,ih] / (sig2*(1+np.exp(-Q[ih])))

    return  WfDer

# Setting up the quantum force for the two-electron quantum dot, recall that it is a vector
def QuantumForce(r,a,b,w):

    sigma=1.0
    sig2 = sigma**2

    qforce = np.zeros((NumberParticles,Dimension), np.double)
    sum1 = np.zeros((NumberParticles,Dimension), np.double)

    Q = Qfac(r,b,w)

    for ih in range(NumberHidden):
        sum1 += w[:,:,ih]/(1+np.exp(-Q[ih]))

    qforce = 2*(-(r-a)/sig2 + sum1/sig2)

    return qforce

def Qfac(r,b,w):
    Q = np.zeros((NumberHidden), np.double)
    temp = np.zeros((NumberHidden), np.double)

    for ih in range(NumberHidden):
        temp[ih] = (r*w[:,:,ih]).sum()

    Q = b + temp

    return Q

# Computing the derivative of the energy and the energy
def EnergyMinimization(a,b,w):

    NumberMCcycles= 10000
    # Parameters in the Fokker-Planck simulation of the quantum force
    D = 0.5
```

```
TimeStep = 0.05
# positions
PositionOld = np.zeros((NumberParticles,Dimension), np.double)
PositionNew = np.zeros((NumberParticles,Dimension), np.double)
# Quantum force
QuantumForceOld = np.zeros((NumberParticles,Dimension), np.double)
QuantumForceNew = np.zeros((NumberParticles,Dimension), np.double)

# seed for rng generator
seed()
energy = 0.0
DeltaE = 0.0

EnergyDer = np.empty((3,),dtype=object)
DeltaPsi = np.empty((3,),dtype=object)
DerivativePsiE = np.empty((3,),dtype=object)
EnergyDer = [np.copy(a),np.copy(b),np.copy(w)]
DeltaPsi = [np.copy(a),np.copy(b),np.copy(w)]
DerivativePsiE = [np.copy(a),np.copy(b),np.copy(w)]
for i in range(3): EnergyDer[i].fill(0.0)
for i in range(3): DeltaPsi[i].fill(0.0)
for i in range(3): DerivativePsiE[i].fill(0.0)


#Initial position
for i in range(NumberParticles):
    for j in range(Dimension):
        PositionOld[i,j] = normalvariate(0.0,1.0)*sqrt(TimeStep)
wfold = WaveFunction(PositionOld,a,b,w)
QuantumForceOld = QuantumForce(PositionOld,a,b,w)

#Loop over MC MCcycles
for MCcycle in range(NumberMCcycles):
    #Trial position moving one particle at the time
    for i in range(NumberParticles):
        for j in range(Dimension):
            PositionNew[i,j] = PositionOld[i,j]+normalvariate(0.0,1.0)*sqrt(TimeStep)+\
                               QuantumForceOld[i,j]*TimeStep*D
        wfnew = WaveFunction(PositionNew,a,b,w)
        QuantumForceNew = QuantumForce(PositionNew,a,b,w)

        GreensFunction = 0.0
        for j in range(Dimension):
            GreensFunction += 0.5*(QuantumForceOld[i,j]+QuantumForceNew[i,j])*\
                              (D*TimeStep*0.5*(QuantumForceOld[i,j]-QuantumForceNew[i,j])-
                              PositionNew[i,j]+PositionOld[i,j])

        GreensFunction = exp(GreensFunction)
        ProbabilityRatio = GreensFunction*wfnew**2/wfold**2
        #Metropolis-Hastings test to see whether we accept the move
        if random() <= ProbabilityRatio:
            for j in range(Dimension):
                PositionOld[i,j] = PositionNew[i,j]
                QuantumForceOld[i,j] = QuantumForceNew[i,j]
            wfold = wfnew
    #print("wf new:        ", wfnew)
    #print("force on 1 new:", QuantumForceNew[0,:])
    #print("pos of 1 new:  ", PositionNew[0,:])
    #print("force on 2 new:", QuantumForceNew[1,:])
    #print("pos of 2 new:  ", PositionNew[1,:])
    DeltaE = LocalEnergy(PositionOld,a,b,w)
```

```
            DerPsi = DerivativeWFansatz(PositionOld,a,b,w)

            DeltaPsi[0] += DerPsi[0]
            DeltaPsi[1] += DerPsi[1]
            DeltaPsi[2] += DerPsi[2]

            energy += DeltaE

            DerivativePsiE[0] += DerPsi[0]*DeltaE
            DerivativePsiE[1] += DerPsi[1]*DeltaE
            DerivativePsiE[2] += DerPsi[2]*DeltaE

        # We calculate mean values
        energy /= NumberMCcycles
        DerivativePsiE[0] /= NumberMCcycles
        DerivativePsiE[1] /= NumberMCcycles
        DerivativePsiE[2] /= NumberMCcycles
        DeltaPsi[0] /= NumberMCcycles
        DeltaPsi[1] /= NumberMCcycles
        DeltaPsi[2] /= NumberMCcycles
        EnergyDer[0]  = 2*(DerivativePsiE[0]-DeltaPsi[0]*energy)
        EnergyDer[1]  = 2*(DerivativePsiE[1]-DeltaPsi[1]*energy)
        EnergyDer[2]  = 2*(DerivativePsiE[2]-DeltaPsi[2]*energy)
        return energy, EnergyDer


    #Here starts the main program with variable declarations
    NumberParticles = 2
    Dimension = 2
    NumberHidden = 2

    interaction=False

    # guess for parameters
    a=np.random.normal(loc=0.0, scale=0.001, size=(NumberParticles,Dimension))
    b=np.random.normal(loc=0.0, scale=0.001, size=(NumberHidden))
    w=np.random.normal(loc=0.0, scale=0.001, size=(NumberParticles,Dimension,NumberHidden))
    # Set up iteration using stochastic gradient method
    Energy = 0
    EDerivative = np.empty((3,),dtype=object)
    EDerivative = [np.copy(a),np.copy(b),np.copy(w)]
    # Learning rate eta, max iterations, need to change to adaptive learning rate
    eta = 0.001
    MaxIterations = 50
    iter = 0
    np.seterr(invalid='raise')
    Energies = np.zeros(MaxIterations)
    EnergyDerivatives1 = np.zeros(MaxIterations)
    EnergyDerivatives2 = np.zeros(MaxIterations)

    while iter < MaxIterations:
        Energy, EDerivative = EnergyMinimization(a,b,w)
        agradient = EDerivative[0]
        bgradient = EDerivative[1]
        wgradient = EDerivative[2]
        a -= eta*agradient
        b -= eta*bgradient
        w -= eta*wgradient
        Energies[iter] = Energy
        print("Energy:",Energy)
        #EnergyDerivatives1[iter] = EDerivative[0]
```

```
    #EnergyDerivatives2[iter] = EDerivative[1]
    #EnergyDerivatives3[iter] = EDerivative[2]


    iter += 1

#nice printout with Pandas
import pandas as pd
from pandas import DataFrame
pd.set_option('max_columns', 6)
data ={'Energy':Energies}#,'A Derivative':EnergyDerivatives1,'B Derivative':EnergyDerivatives2,'W

frame = pd.DataFrame(data)
print(frame)
```