

Third part, quantum mechanical studies

Morten Hjorth-Jensen¹

Department of Physics and Center for Computing in Science Education,
University of Oslo, Norway¹

Geilo Winter School, March 10-20, 2025

Many-body physics, Quantum Monte Carlo and deep learning

Given a hamiltonian H and a trial wave function Ψ_T , the variational principle states that the expectation value of $\langle H \rangle$, defined through

$$\langle E \rangle = \frac{\int d\mathbf{R} \Psi_T^*(\mathbf{R}) H(\mathbf{R}) \Psi_T(\mathbf{R})}{\int d\mathbf{R} \Psi_T^*(\mathbf{R}) \Psi_T(\mathbf{R})},$$

is an upper bound to the ground state energy E_0 of the hamiltonian H , that is

$$E_0 \leq \langle E \rangle.$$

In general, the integrals involved in the calculation of various expectation values are multi-dimensional ones. Traditional integration methods such as the Gauss-Legendre will not be adequate for say the computation of the energy of a many-body system. **Basic philosophy:** Let a neural network find the optimal wave function

Quantum Monte Carlo Motivation

Basic steps

Choose a trial wave function $\psi_T(\mathbf{R})$.

$$P(\mathbf{R}, \alpha) = \frac{|\psi_T(\mathbf{R}, \alpha)|^2}{\int |\psi_T(\mathbf{R}, \alpha)|^2 d\mathbf{R}}.$$

This is our model, or likelihood/probability distribution function (PDF). It depends on some variational parameters α . The approximation to the expectation value of the Hamiltonian is now

$$\langle E[\alpha] \rangle = \frac{\int d\mathbf{R} \Psi_T^*(\mathbf{R}, \alpha) H(\mathbf{R}) \Psi_T(\mathbf{R}, \alpha)}{\int d\mathbf{R} \Psi_T^*(\mathbf{R}, \alpha) \Psi_T(\mathbf{R}, \alpha)}.$$

Quantum Monte Carlo Motivation

Define a new quantity

$$E_L(\mathbf{R}, \alpha) = \frac{1}{\psi_T(\mathbf{R}, \alpha)} H \psi_T(\mathbf{R}, \alpha),$$

called the local energy, which, together with our trial PDF yields

$$\langle E[\alpha] \rangle = \int P(\mathbf{R}) E_L(\mathbf{R}, \alpha) d\mathbf{R} \approx \frac{1}{N} \sum_{i=1}^N E_L(\mathbf{R}_i, \alpha)$$

with N being the number of Monte Carlo samples.

Energy derivatives

The local energy as function of the variational parameters defines now our **objective/cost** function.

To find the derivatives of the local energy expectation value as function of the variational parameters, we can use the chain rule and the hermiticity of the Hamiltonian.

Let us define (with the notation $\langle E[\alpha] \rangle = \langle E_L \rangle$)

$$\bar{E}_{\alpha_i} = \frac{d\langle E_L \rangle}{d\alpha_i},$$

as the derivative of the energy with respect to the variational parameter α_i ; We define also the derivative of the trial function (skipping the subindex T) as

$$\bar{\Psi}_i = \frac{d\Psi}{d\alpha_i}.$$

Derivatives of the local energy

The elements of the gradient of the local energy are

$$\bar{E}_i = 2 \left(\langle \frac{\bar{\Psi}_i}{\Psi} E_L \rangle - \langle \frac{\bar{\Psi}_i}{\Psi} \rangle \langle E_L \rangle \right).$$

From a computational point of view it means that you need to compute the expectation values of

$$\langle \frac{\bar{\Psi}_i}{\Psi} E_L \rangle,$$

and

$$\langle \frac{\bar{\Psi}_i}{\Psi} \rangle \langle E_L \rangle$$

These integrals are evaluated using MC integration (with all its possible error sources). Use methods like stochastic gradient or other minimization methods to find the optimal parameters.

Monte Carlo methods and Neural Networks

Machine Learning and the Deuteron by Kebble and Rios and
Variational Monte Carlo calculations of $A \leq 4$ nuclei with an
artificial neural-network correlator ansatz by Adams et al.

Adams et al:

$$H_{LO} = - \sum_i \frac{\vec{\nabla}_i^2}{2m_N} + \sum_{i < j} (C_1 + C_2 \vec{\sigma}_i \cdot \vec{\sigma}_j) e^{-r_{ij}^2 \Lambda^2 / 4} + D_0 \sum_{i < j < k} \sum_{\text{cyc}} e^{-(r_{ik}^2 + r_{ij}^2) \Lambda^2 / 4}, \quad (1)$$

where m_N is the mass of the nucleon, $\vec{\sigma}_i$ is the Pauli matrix acting on nucleon i , and \sum_{cyc} stands for the cyclic permutation of i , j , and k . The low-energy constants C_1 and C_2 are fit to the deuteron binding energy and to the neutron-neutron scattering length

Deep learning neural networks, Variational Monte Carlo calculations of $A \leq 4$ nuclei with an artificial neural-network correlator ansatz by Adams et al.

An appealing feature of the neural network ansatz is that it is more general than the more conventional product of two- and three-body spin-independent Jastrow functions

$$|\Psi_V^J\rangle = \prod_{i < j < k} \left(1 - \sum_{\text{cyc}} u(r_{ij})u(r_{jk})\right) \prod_{i < j} f(r_{ij}) |\Phi\rangle, \quad (2)$$

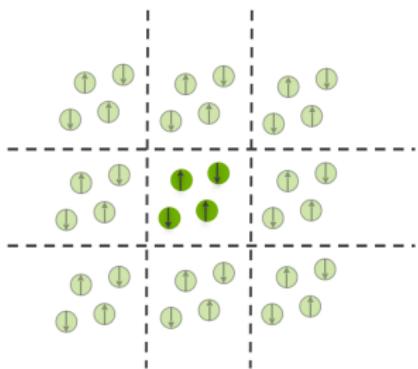
which is commonly used for nuclear Hamiltonians that do not contain tensor and spin-orbit terms. The above function is replaced by a four-layer Neural Network.

Ansatz for a fermionic state function, Jane Kim et al,
Commun Phys 7, 148 (2024)

$$\Psi_T(\mathbf{X}) = \exp U(\mathbf{X}) \Phi(\mathbf{X}).$$

1. Build in fermion antisymmetry for network compactness
2. Permutation-invariant Jastrow function improves ansatz flexibility
3. Build U and Φ functions from fully connected, deep neural networks
4. Use Slater determinant (or Pfaffian) Φ to enforce antisymmetry with single particle wavefunctions represented by neural networks

Nuclear matter setup

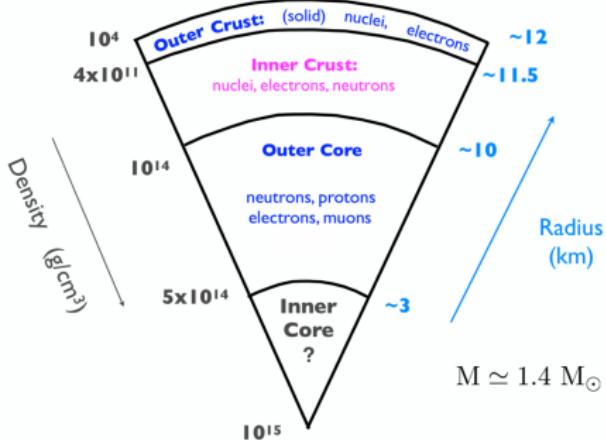


- Periodic boundary conditions and coordinate system

$$\mathbf{r}_i \rightarrow \tilde{\mathbf{r}}_i = \left\{ \sin\left(\frac{2\pi}{L}\mathbf{r}_i\right), \cos\left(\frac{2\pi}{L}\mathbf{r}_i\right) \right\}$$

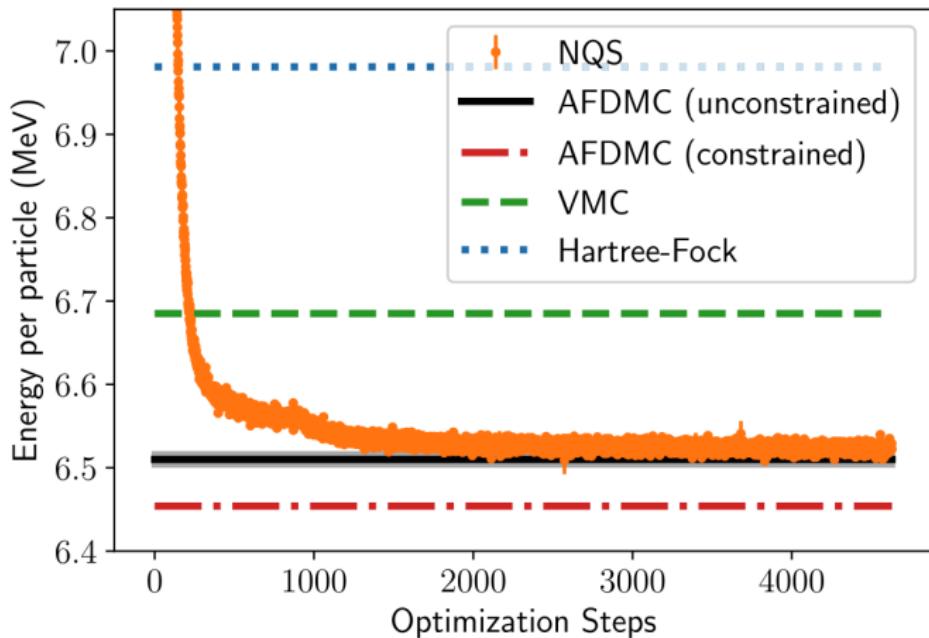
- Potential energy contribution from particle images
- Remove Coulomb potential

Neutron star structure

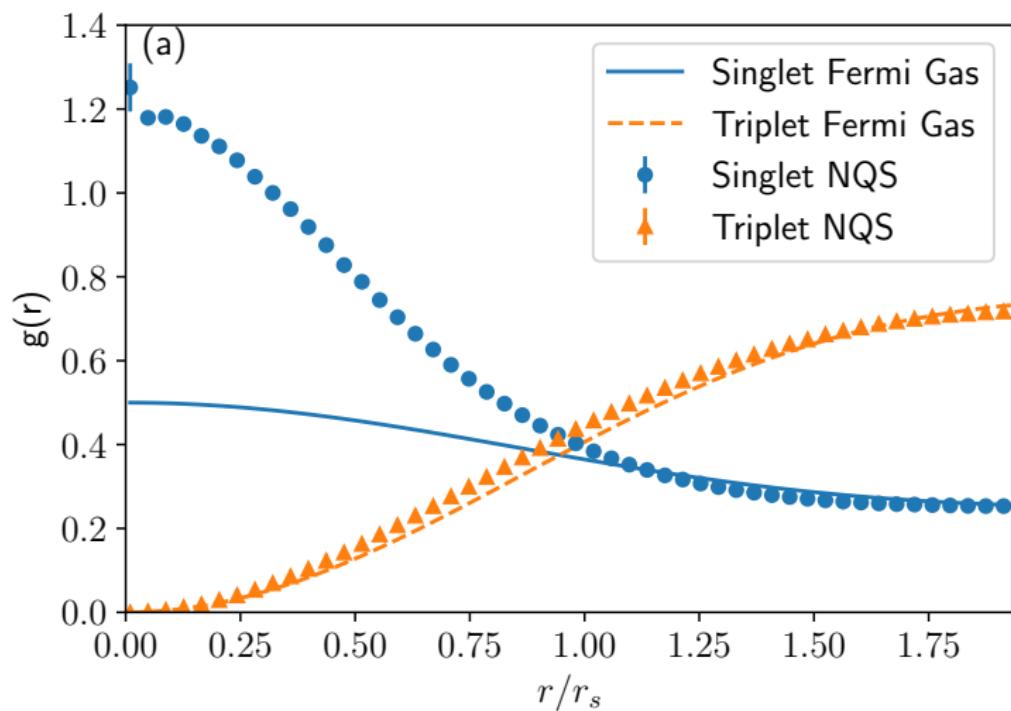


- Mostly neutrons but composition varies with density
- Nuclei in crust are squeezed into uniform matter in core
- Likely neutron superfluid in inner crust and outer core
- Calculations currently focus on inner crust

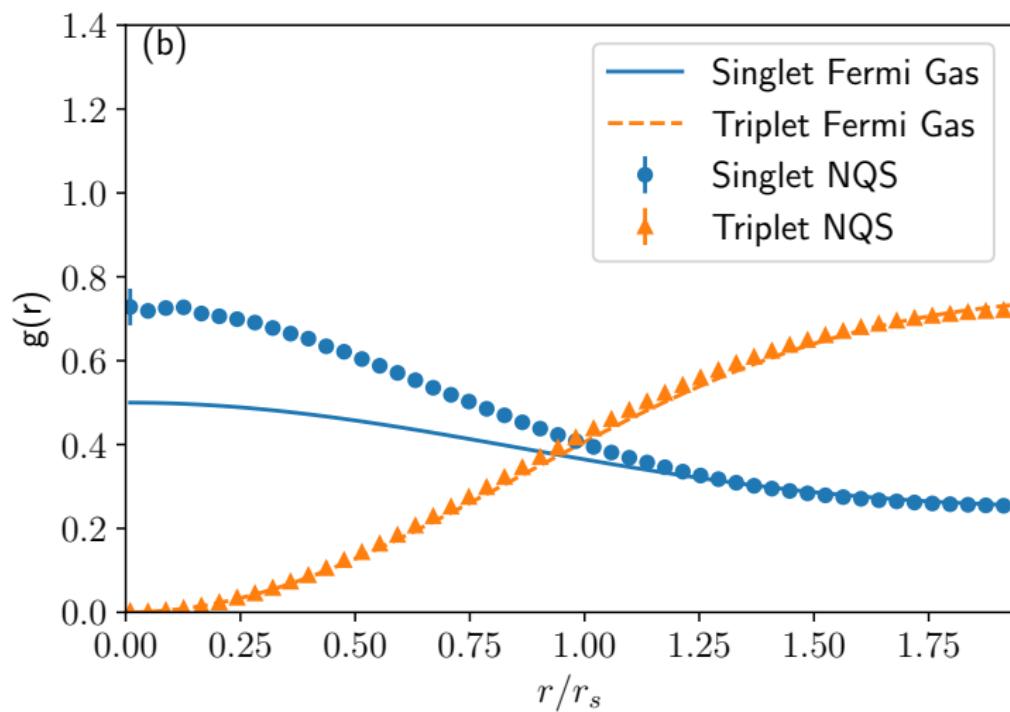
Dilute neutron star matter from neural-network quantum states by Fore et al, Physical Review Research 5, 033062 (2023) at density $\rho = 0.04 \text{ fm}^{-3}$



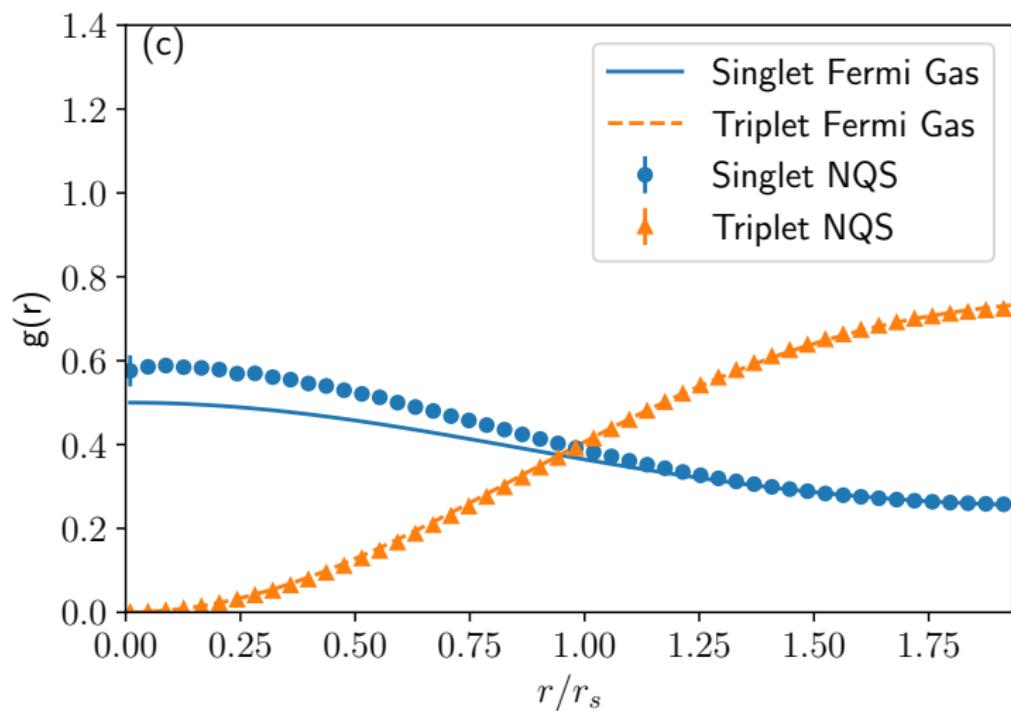
Pairing and Spin-singlet and triplet two-body distribution functions at $\rho = 0.01 \text{ fm}^{-3}$



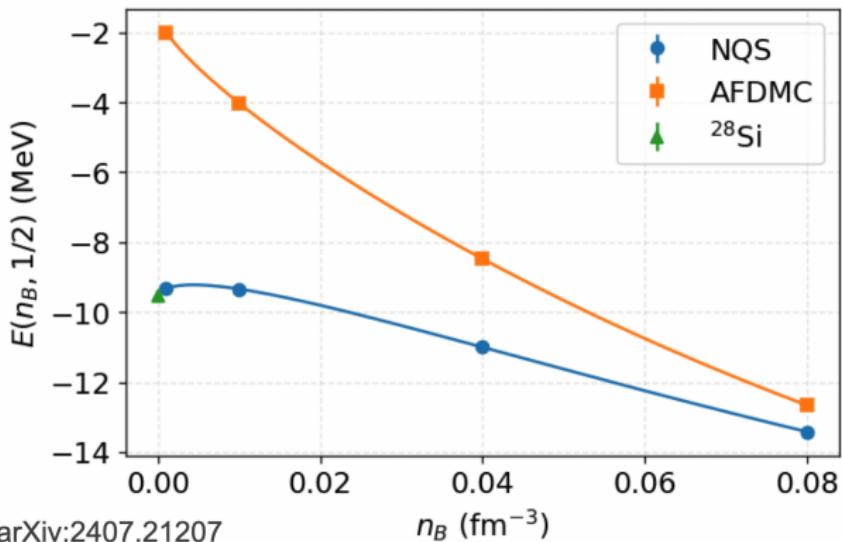
Pairing and Spin-singlet and triplet two-body distribution functions at $\rho = 0.04 \text{ fm}^{-3}$



Pairing and Spin-singlet and triplet two-body distribution functions at $\rho = 0.08 \text{ fm}^{-3}$

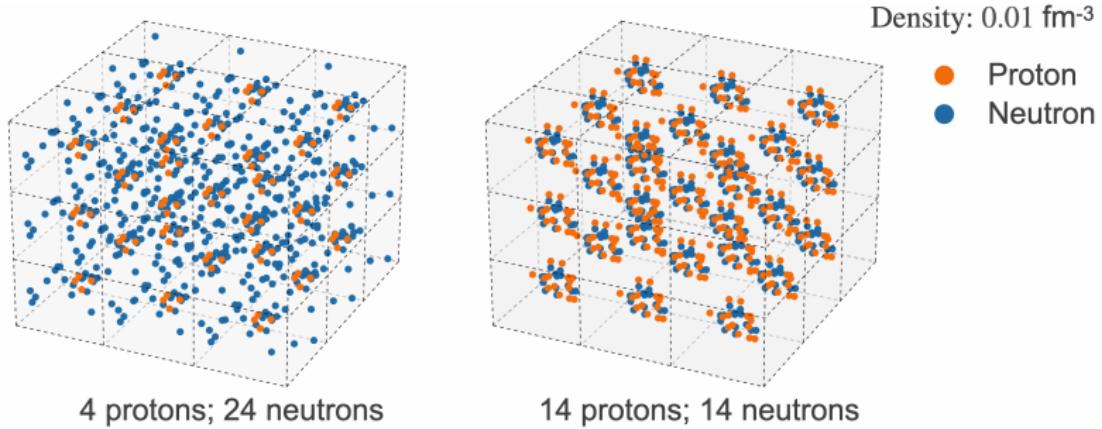


Symmetric nuclear matter



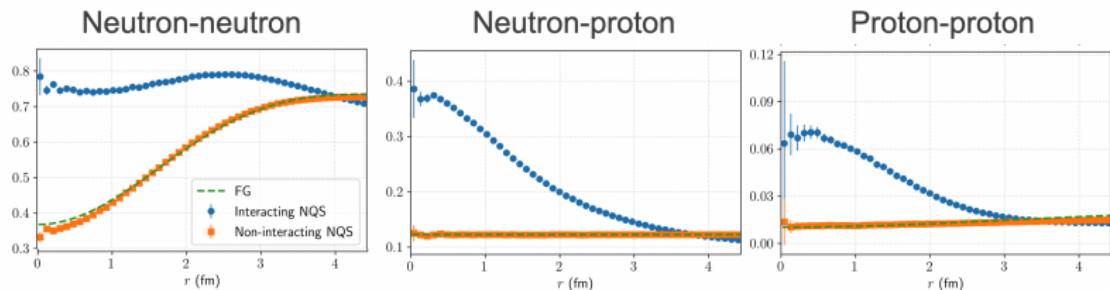
B. Fore, arXiv:2407.21207

Self-emerging clustering



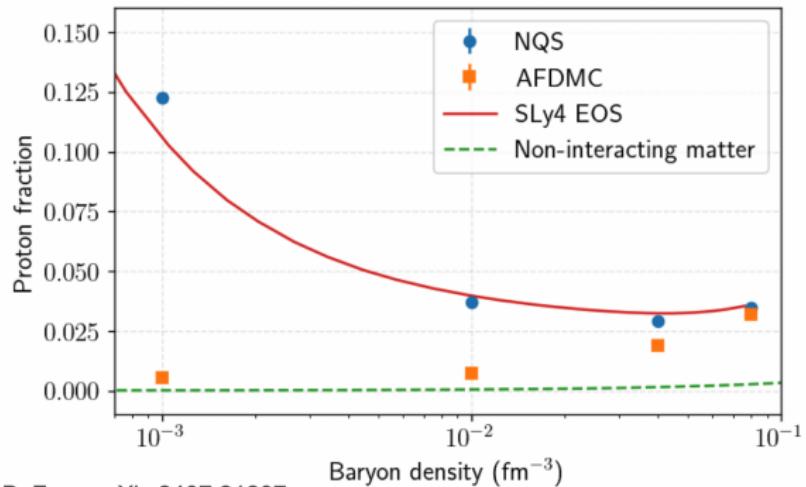
B. Fore, arXiv:2407.21207

Clustering: Two-body pair distributions



B. Fore, arXiv:2407.21207

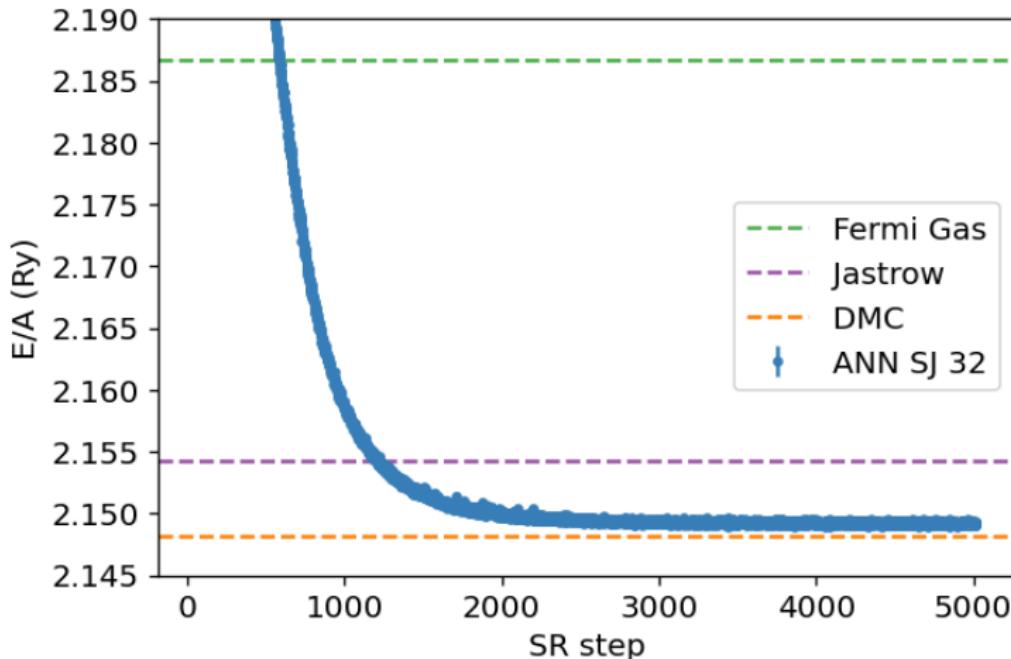
Nuclear matter proton fraction



Assumptions:

- Charge neutrality
 $n_p = n_e$
- Beta equilibrium
 $\mu_e = \mu_n - \mu_p$

The electron gas in three dimensions with $N = 14$ electrons
(Wigner-Seitz radius $r_s = 2$ a.u.), Gabriel Pescia, Jane Kim
et al. arXiv.2305.07240,



Essential elements of generative models

The aim of generative methods is to train a probability distribution p . There are several methods:

1. Energy based models, with the family of Boltzmann distributions as a typical example
2. Variational autoencoders, based on our discussions on autoencoders
3. Generative adversarial networks (GANs) and
4. Diffusion models

Energy models

Let us define a domain \mathbf{X} of stochastic variables

$\mathbf{X} = \{x_0, x_1, \dots, x_{n-1}\}$ with a pertinent probability distribution

$$p(\mathbf{X}) = \prod_{x_i \in \mathbf{X}} p(x_i),$$

where we have assumed that the random variables x_i are all independent and identically distributed (iid).

We will now assume that we can define this function in terms of optimization parameters Θ , which could be the biases and weights of deep network, and a set of hidden variables we also assume to be random variables which are iid. The domain of these variables is $\mathbf{H} = \{h_0, h_1, \dots, h_{m-1}\}$.

Probability model

We define a probability

$$p(x_i, h_j; \Theta) = \frac{f(x_i, h_j; \Theta)}{Z(\Theta)},$$

where $f(x_i, h_j; \Theta)$ is a function which we assume is larger or equal than zero and obeys all properties required for a probability distribution and $Z(\Theta)$ is a normalization constant. Inspired by statistical mechanics, we call it often for the partition function. It is defined as (assuming that we have discrete probability distributions)

$$Z(\Theta) = \sum_{x_i \in \mathcal{X}} \sum_{h_j \in \mathcal{H}} f(x_i, h_j; \Theta).$$

Marginal and conditional probabilities

We can in turn define the marginal probabilities

$$p(x_i; \Theta) = \frac{\sum_{h_j \in H} f(x_i, h_j; \Theta)}{Z(\Theta)},$$

and

$$p(h_i; \Theta) = \frac{\sum_{x_i \in X} f(x_i, h_i; \Theta)}{Z(\Theta)}.$$

Change of notation

Note the change to a vector notation. A variable like \mathbf{x} represents now a specific **configuration**. We can generate an infinity of such configurations. The final partition function is then the sum over all such possible configurations, that is

$$Z(\Theta) = \sum_{x_i \in \mathbf{X}} \sum_{h_j \in \mathbf{H}} f(x_i, h_j; \Theta),$$

changes to

$$Z(\Theta) = \sum_{\mathbf{x}} \sum_{\mathbf{h}} f(\mathbf{x}, \mathbf{h}; \Theta).$$

If we have a binary set of variable x_i and h_j and M values of x_i and N values of h_j we have in total 2^M and 2^N possible \mathbf{x} and \mathbf{h} configurations, respectively.

We see that even for the modest binary case, we can easily approach a number of configuration which is not possible to deal with.

Optimization problem

At the end, we are not interested in the probabilities of the hidden variables. The probability we thus want to optimize is

$$p(\mathbf{X}; \Theta) = \prod_{x_i \in \mathbf{X}} p(x_i; \Theta) = \prod_{x_i \in \mathbf{X}} \left(\frac{\sum_{h_j \in \mathcal{H}} f(x_i, h_j; \Theta)}{Z(\Theta)} \right),$$

which we rewrite as

$$p(\mathbf{X}; \Theta) = \frac{1}{Z(\Theta)} \prod_{x_i \in \mathbf{X}} \left(\sum_{h_j \in \mathcal{H}} f(x_i, h_j; \Theta) \right).$$

Further simplifications

We simplify further by rewriting it as

$$p(\mathbf{X}; \Theta) = \frac{1}{Z(\Theta)} \prod_{x_i \in \mathbf{X}} f(x_i; \Theta),$$

where we used $p(x_i; \Theta) = \sum_{h_j \in \mathcal{H}} f(x_i, h_j; \Theta)$. The optimization problem is then

$$\arg \max_{\Theta \in \mathbb{R}^p} p(\mathbf{X}; \Theta).$$

Optimizing the logarithm instead

Computing the derivatives with respect to the parameters Θ is easier (and equivalent) with taking the logarithm of the probability. We will thus optimize

$$\arg \max_{\Theta \in \mathbb{R}^p} \log p(\mathbf{X}; \Theta),$$

which leads to

$$\nabla_{\Theta} \log p(\mathbf{X}; \Theta) = 0.$$

Expression for the gradients

This leads to the following equation

$$\nabla_{\Theta} \log p(\mathbf{X}; \Theta) = \nabla_{\Theta} \left(\sum_{x_i \in \mathbf{X}} \log f(x_i; \Theta) \right) - \nabla_{\Theta} \log Z(\Theta) = 0.$$

The first term is called the positive phase and we assume that we have a model for the function f from which we can sample values. Below we will develop an explicit model for this. The second term is called the negative phase and is the one which leads to more difficulties.

The derivative of the partition function

The partition function, defined above as

$$Z(\Theta) = \sum_{x_i \in \mathcal{X}} \sum_{h_j \in \mathcal{H}} f(x_i, h_j; \Theta),$$

is in general the most problematic term. In principle both x and h can span large degrees of freedom, if not even infinitely many ones, and computing the partition function itself is often not desirable or even feasible. The above derivative of the partition function can however be written in terms of an expectation value which is in turn evaluated using Monte Carlo sampling and the theory of Markov chains, popularly shortened to MCMC (or just MC²).

Explicit expression for the derivative

We can rewrite

$$\nabla_{\Theta} \log Z(\Theta) = \frac{\nabla_{\Theta} Z(\Theta)}{Z(\Theta)},$$

which reads in more detail

$$\nabla_{\Theta} \log Z(\Theta) = \frac{\nabla_{\Theta} \sum_{x_i \in \mathcal{X}} f(x_i; \Theta)}{Z(\Theta)}.$$

We can rewrite the function f (we have assumed that is larger or equal than zero) as $f = \exp \log f$. We can then rewrite the last equation as

$$\nabla_{\Theta} \log Z(\Theta) = \frac{\sum_{x_i \in \mathcal{X}} \nabla_{\Theta} \exp \log f(x_i; \Theta)}{Z(\Theta)}.$$

Final expression

Taking the derivative gives us

$$\nabla_{\Theta} \log Z(\Theta) = \frac{\sum_{x_i \in \mathcal{X}} f(x_i; \Theta) \nabla_{\Theta} \log f(x_i; \Theta)}{Z(\Theta)},$$

which is the expectation value of $\log f$

$$\nabla_{\Theta} \log Z(\Theta) = \sum_{x_i \in \mathcal{X}} p(x_i; \Theta) \nabla_{\Theta} \log f(x_i; \Theta),$$

that is

$$\nabla_{\Theta} \log Z(\Theta) = \mathbb{E}(\log f(x_i; \Theta)).$$

This quantity is evaluated using Monte Carlo sampling, with Gibbs sampling as the standard sampling rule. Before we discuss the explicit algorithms, we need to remind ourselves about Markov chains and sampling rules like the Metropolis-Hastings algorithm and Gibbs sampling.

Introducing the energy model

As we will see below, a typical Boltzmann machines employs a probability distribution

$$p(\mathbf{x}, \mathbf{h}; \Theta) = \frac{f(\mathbf{x}, \mathbf{h}; \Theta)}{Z(\Theta)},$$

where $f(\mathbf{x}, \mathbf{h}; \Theta)$ is given by a so-called energy model. If we assume that the random variables x_i and h_j take binary values only, for example $x_i, h_j = \{0, 1\}$, we have a so-called binary-binary model where

$$f(\mathbf{x}, \mathbf{h}; \Theta) = -E(\mathbf{x}, \mathbf{h}; \Theta) = \sum_{x_i \in \mathbf{X}} x_i a_i + \sum_{h_j \in \mathbf{H}} b_j h_j + \sum_{x_i \in \mathbf{X}, h_j \in \mathbf{H}} x_i w_{ij} h_j,$$

where the set of parameters are given by the biases and weights $\Theta = \{\mathbf{a}, \mathbf{b}, \mathbf{W}\}$. Note the vector notation instead of x_i and h_j for f . The vectors \mathbf{x} and \mathbf{h} represent a specific instance of stochastic variables x_i and h_j . These arrangements of \mathbf{x} and \mathbf{h} lead to a specific energy configuration.

More compact notation

With the above definition we can write the probability as

$$p(\mathbf{x}, \mathbf{h}; \Theta) = \frac{\exp(\mathbf{a}^T \mathbf{x} + \mathbf{b}^T \mathbf{h} + \mathbf{x}^T \mathbf{W} \mathbf{h})}{Z(\Theta)},$$

where the biases \mathbf{a} and \mathbf{h} and the weights defined by the matrix \mathbf{W} are the parameters we need to optimize.

Anticipating results to be derived

Since the binary-binary energy model is linear in the parameters a_i , b_j and w_{ij} , it is easy to see that the derivatives with respect to the various optimization parameters yield expressions used in the evaluation of gradients like

$$\frac{\partial E(\mathbf{x}, \mathbf{h}; \Theta)}{\partial w_{ij}} = -x_i h_j,$$

and

$$\frac{\partial E(\mathbf{x}, \mathbf{h}; \Theta)}{\partial a_i} = -x_i,$$

and

$$\frac{\partial E(\mathbf{x}, \mathbf{h}; \Theta)}{\partial b_j} = -h_j.$$

Basics of the Boltzmann machine

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.

More about the basics

A standard BM network is divided into a set of observable and visible units x and a set of unknown hidden units/nodes 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 can be trained one after another, a feature which makes them popular in deep learning

Difficult to train

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 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".

Goal of hidden layer

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).

The parameters

The network parameters, to be optimized/learned:

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

Note that we have specified the lengths of bmx and \mathbf{h} . These lengths define the number of visible and hidden units, respectively.

Joint distribution

The restricted Boltzmann machine is described by a Boltzmann distribution

$$P_{rbm}(\mathbf{x}, \mathbf{h}, \Theta) = \frac{1}{Z(\Theta)} \exp(-E(\mathbf{x}, \mathbf{h}, \Theta)),$$

where Z is the normalization constant or partition function discussed earlier and defined as

$$Z(\Theta) = \int \int \exp(-E(\mathbf{x}, \mathbf{h}, \Theta)) d\mathbf{x} d\mathbf{h}.$$

It is common to set the temperature T to one. It is omitted in the equations above. The energy is thus a dimensionless function.

Network Elements, the energy function

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

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:

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

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:

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

This type of RBMs are useful when we model continuous data (i.e., we wish \mathbf{x} to be continuous). The parameter σ_i^2 is meant to represent a variance and is often just set to one.

Energy-based models and Langevin sampling

See discussions in Foster, chapter 7 on energy-based models at
https://github.com/davidADSP/Generative_Deep_Learning_2nd_Edition/tree/main/notebooks/07_ebm/01_ebm
That notebook is based on a recent article by Du and Mordatch,
Implicit generation and modeling with energy-based models,
see <https://arxiv.org/pdf/1903.08689.pdf>.

Tensor-flow examples

1. To create Boltzmann machine using Keras, see Babcock and Bali chapter 4, see https://github.com/PacktPublishing/Hands-On-Generative-AI-with-Python-and-TensorFlow-2/blob/master/Chapter_4/models/rbm.py
2. See also Foster, chapter 7 on energy-based models at https://github.com/davidADSP/Generative_Deep_Learning_2nd_Edition/tree/main/notebooks/07_ebm/01_ebm

Efficient solutions of fermionic systems using artificial neural networks, Nordhagen et al, Frontiers in Physics 11, 2023

The Hamiltonian of the quantum dot is given by

$$\hat{H} = \hat{H}_0 + \hat{V},$$

where \hat{H}_0 is the many-body HO Hamiltonian, and \hat{V} is the inter-electron Coulomb interactions. In dimensionless units,

$$\hat{V} = \sum_{i < j}^N \frac{1}{r_{ij}},$$

with $r_{ij} = \sqrt{r_i^2 - r_j^2}$.

Separable Hamiltonian with the relative motion part ($r_{ij} = r$)

$$\hat{H}_r = -\nabla_r^2 + \frac{1}{4}\omega^2 r^2 + \frac{1}{r},$$

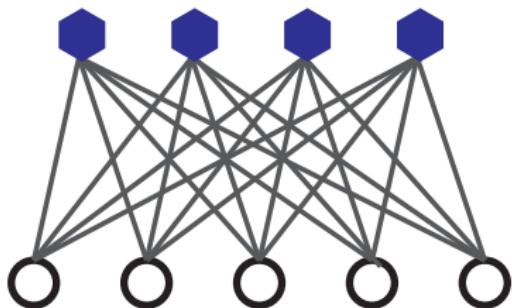
Analytical solutions in two and three dimensions (M. Taut 1993 and 1994).

The structure of the RBM network

Hidden Layer

Interactions

Visible Layer



$$b_\mu(h_\mu)$$

$$W_{i\mu} v_i h_\mu$$

$$a_i(v_i)$$

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".

Goals

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).

The network parameters, to be optimized/learned:

1. \mathbf{a} represents the visible bias, a vector of same length as \mathbf{x} .
2. \mathbf{b} represents the hidden bias, a vector of same lenght as \mathbf{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

$$P_{\text{rbm}}(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} \exp -E(\mathbf{x}, \mathbf{h}),$$

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

$$Z = \int \int \exp -E(\mathbf{x}, \mathbf{h}) d\mathbf{x} d\mathbf{h}.$$

Note the absence of the inverse temperature in these equations.

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 \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 (Energy based models)

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:

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

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

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

Gaussian binary

Gaussian-Binary RBM:

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

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

Representing the wave function

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}

$$P_{\text{rbm}}(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} \exp -E(\mathbf{x}, \mathbf{h}).$$

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

$$P_{\text{rbm}}(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{h}} \exp -E(\mathbf{x}, \mathbf{h}).$$

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

$$|\Psi(\mathbf{X})|^2 = P_{\text{rbm}}(\mathbf{x}).$$

Define the cost function

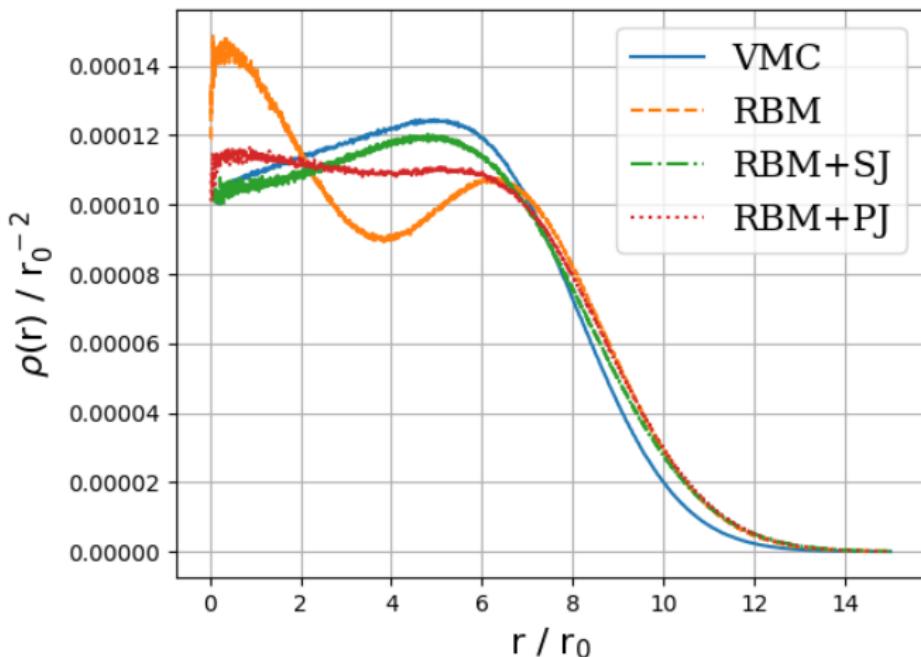
Now we don't necessarily have training data (unless we generate it by using some other method). However, what we do have is the variational principle which allows us to obtain the ground state wave function by minimizing the expectation value of the energy of a trial wavefunction (corresponding to the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

$$C_i = \frac{\partial \langle E_L \rangle}{\partial \theta_i} = 2(\langle E_L \frac{1}{\Psi} \frac{\partial \Psi}{\partial \theta_i} \rangle - \langle E_L \rangle \langle \frac{1}{\Psi} \frac{\partial \Psi}{\partial \theta_i} \rangle),$$

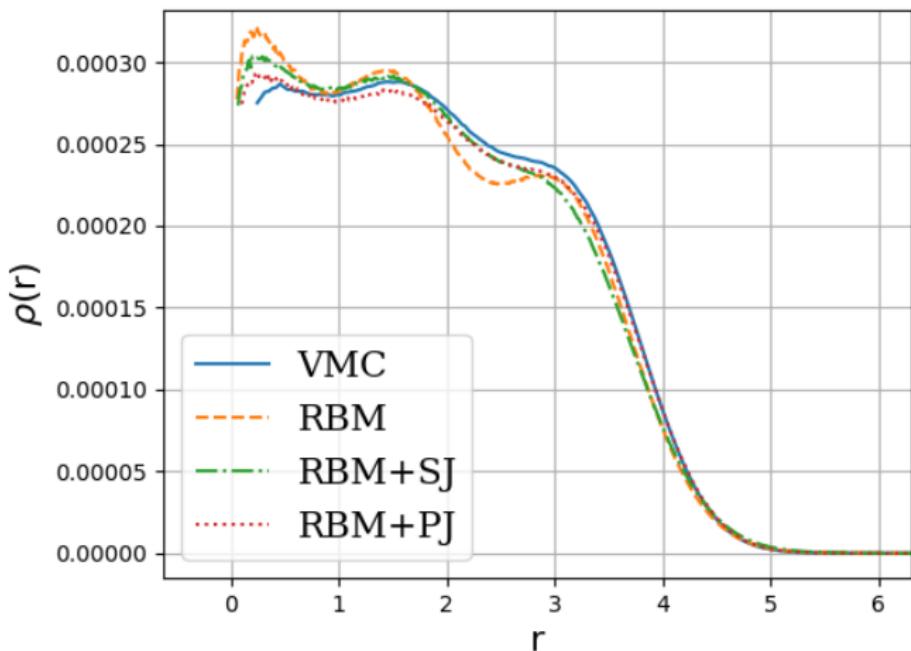
where the local energy is given by

$$E_L = \frac{1}{\Psi} \hat{H} \Psi.$$

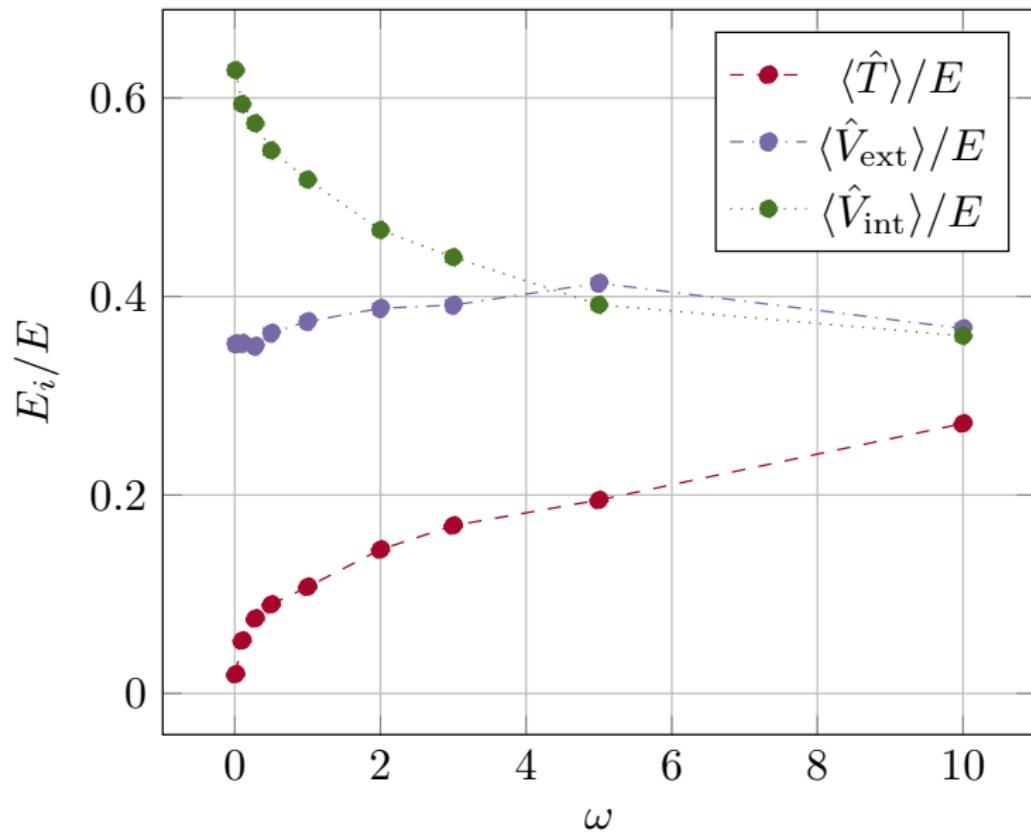
Quantum dots and Boltzmann machines, onebody densities
 $N = 6$, $\hbar\omega = 0.1$ a.u.



Onebody densities $N = 30$, $\hbar\omega = 1.0$ a.u.



Expectation values as functions of the oscillator frequency



Code example

This part is best seen using the jupyter-notebook

```
# 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 wavef
# RBM code based heavily off of:
# https://github.com/CompPhysics/ComputationalPhysics2/tree/gh-pages/d
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
```

Additional material: 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

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

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

$$\Theta_{k+1} = \Theta_k - \eta \nabla \mathcal{C}(\Theta_k)$$

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.

Gradients

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

$$\begin{aligned}\frac{\partial \mathcal{C}(\{\Theta_i\})}{\partial \Theta_i} &= \left\langle \frac{\partial E(\mathbf{x}; \Theta_i)}{\partial \Theta_i} \right\rangle_{data} + \frac{\partial \log Z(\{\Theta_i\})}{\partial \Theta_i} \\ &= \langle O_i(\mathbf{x}) \rangle_{data} - \langle O_i(\mathbf{x}) \rangle_{model}.\end{aligned}$$

Simplifications

In order to simplify notation we defined the "operator"

$$O_i(\mathbf{x}) = \frac{\partial E(\mathbf{x}; \Theta_i)}{\partial \Theta_i},$$

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

$$\langle O_i(\mathbf{x}) \rangle_{model} = \text{Tr} P_\Theta(\mathbf{x}) O_i(\mathbf{x}) = -\frac{\partial \log Z(\{\Theta_i\})}{\partial \Theta_i}.$$

Positive and negative phases

As discussed earlier, 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).

Gradient examples

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

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

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

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

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 unknown probability which we approximate with q , we can measure the difference by

$$\text{KL}(p||q) = \int_{-\infty}^{\infty} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x}.$$

Kullback-Leibler divergence

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

$$\begin{aligned}\text{KL}(f(\mathbf{x})||p(\mathbf{x}|\Theta)) &= \int_{-\infty}^{\infty} f(\mathbf{x}) \log \frac{f(\mathbf{x})}{p(\mathbf{x}|\Theta)} d\mathbf{x} \\ &= \int_{-\infty}^{\infty} f(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x} - \int_{-\infty}^{\infty} f(\mathbf{x}) \log p(\mathbf{x}|\Theta) d\mathbf{x} \\ &= \langle \log f(\mathbf{x}) \rangle_{f(\mathbf{x})} - \langle \log p(\mathbf{x}|\Theta) \rangle_{f(\mathbf{x})} \\ &= \langle \log f(\mathbf{x}) \rangle_{data} + \langle E(\mathbf{x}) \rangle_{data} + \log Z \\ &= \langle \log f(\mathbf{x}) \rangle_{data} + \mathcal{C}_{LL}.\end{aligned}$$

Maximizing log-likelihood

The first term is constant with respect to Θ since $f(x)$ is independent of Θ . 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.

More on the partition function

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

$$\left\langle \frac{\partial E(\mathbf{x}; \Theta_i)}{\partial \Theta_i} \right\rangle_{model} = \int p(\mathbf{x}|\Theta) \frac{\partial E(\mathbf{x}; \Theta_i)}{\partial \Theta_i} d\mathbf{x} = -\frac{\partial \log Z(\Theta_i)}{\partial \Theta_i}.$$

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

Setting up for gradient descent calculations

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

$$\begin{aligned}\frac{\partial \mathcal{C}_{LL}}{\partial \Theta_i} &= \left\langle \frac{\partial E(\mathbf{x}; \Theta_i)}{\partial \Theta_i} \right\rangle_{data} + \frac{\partial \log Z(\Theta_i)}{\partial \Theta_i} \\ &= \left\langle \frac{\partial E(\mathbf{x}; \Theta_i)}{\partial \Theta_i} \right\rangle_{data} - \left\langle \frac{\partial E(\mathbf{x}; \Theta_i)}{\partial \Theta_i} \right\rangle_{model}\end{aligned}$$

Difference of moments

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 \mathbf{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.

More observations

The gradient of the cost function also demonstrates why gradients of unsupervised, generative models must be computed differently from those of for example FNNs. While the data-dependent expectation value is easily calculated based on the samples x_i in the training data, we must sample from the model in order to generate samples from which to calculate 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.

Adding hyperparameters

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.

The original RBM had binary visible and hidden nodes. They were shown to be universal approximators of discrete distributions. It was also shown that adding hidden units yields strictly improved modelling power.

Binary-binary (BB) RBMs

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. We have the energy function

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

Marginal probability

We have the binary-binary marginal probability defined as

$$\begin{aligned} p_{BB}(\mathbf{x}, \mathbf{h}, \Theta) &= \frac{1}{Z_{BB}(\Theta)} e^{\sum_i^M a_i x_i + \sum_j^N b_j h_j + \sum_{ij}^{M,N} x_i w_{ij} h_j} \\ &= \frac{1}{Z_{BB}(\Theta)} e^{\mathbf{a}^T \mathbf{x} + \mathbf{b}^T \mathbf{h} + \mathbf{x}^T \mathbf{W} \mathbf{h}} \end{aligned}$$

with the partition function

$$Z_{BB}(\Theta) = \sum_{\mathbf{x}, \mathbf{h}} e^{\mathbf{a}^T \mathbf{x} + \mathbf{b}^T \mathbf{h} + \mathbf{x}^T \mathbf{W} \mathbf{h}}.$$

Marginal Probability Density Function for the visible units

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

$$\begin{aligned} p_{BB}(\mathbf{x}, \Theta) &= \sum_{\mathbf{h}} p_{BB}(\mathbf{x}, \mathbf{h}, \Theta) \\ &= \frac{1}{Z_{BB}} \sum_{\mathbf{h}} e^{\mathbf{a}^T \mathbf{x} + \mathbf{b}^T \mathbf{h} + \mathbf{x}^T \mathbf{W} \mathbf{h}} \\ &= \frac{1}{Z_{BB}} e^{\mathbf{a}^T \mathbf{x}} \sum_{\mathbf{h}} e^{\sum_j^N (b_j + \mathbf{x}^T \mathbf{w}_{*j}) h_j} \\ &= \frac{1}{Z_{BB}} e^{\mathbf{a}^T \mathbf{x}} \sum_{\mathbf{h}} \prod_j^N e^{(b_j + \mathbf{x}^T \mathbf{w}_{*j}) h_j} \\ &= \frac{1}{Z_{BB}} e^{\mathbf{a}^T \mathbf{x}} \left(\sum_{h_1} e^{(b_1 + \mathbf{x}^T \mathbf{w}_{*1}) h_1} \times \sum_{h_2} e^{(b_2 + \mathbf{x}^T \mathbf{w}_{*2}) h_2} \times \right. \\ &\quad \left. \dots \times \sum_{h_N} e^{(b_N + \mathbf{x}^T \mathbf{w}_{*N}) h_N} \right) \end{aligned}$$

Marginal probability for hidden units

A similar derivation yields the marginal probability of the hidden units

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

Conditional Probability Density Functions

We derive the probability of the hidden units given the visible units using Bayes' rule (we drop the explicit Θ dependence)

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

On and off probabilities

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

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

and

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

Conditional probability for visible units

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

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

We have

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

Gaussian-Binary Restricted Boltzmann Machines

Inserting into the expression for $E_{RBM}(\mathbf{x}, \mathbf{h}, \Theta)$ in equation results in the energy

$$\begin{aligned} E_{GB}(\mathbf{x}, \mathbf{h}, \Theta) &= \sum_i^M \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_j^N b_j h_j - \sum_{ij}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2} \\ &= \left\| \frac{\mathbf{x} - \mathbf{a}}{2\sigma} \right\|^2 - \mathbf{b}^T \mathbf{h} - \left(\frac{\mathbf{x}}{\sigma^2} \right)^T \mathbf{W} \mathbf{h}. \end{aligned}$$

Joint Probability Density Function

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

Partition function

The partition function is given by

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

Marginal Probability Density Functions

We proceed to find the marginal probability densities of the Gaussian-binary RBM. We first marginalize over the binary hidden units to find $p_{GB}(\mathbf{x})$

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

Then the visible units

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

$$\begin{aligned} p_{GB}(\mathbf{h}) &= \int p_{GB}(\tilde{\mathbf{x}}, \mathbf{h}) d\tilde{\mathbf{x}} \\ &= \frac{1}{Z_{GB}} \int e^{-||\frac{\tilde{\mathbf{x}} - \mathbf{a}}{2\sigma}||^2 + \mathbf{b}^T \mathbf{h} + (\frac{\tilde{\mathbf{x}}}{\sigma^2})^T \mathbf{W} \mathbf{h}} d\tilde{\mathbf{x}} \\ &= \frac{1}{Z_{GB}} e^{\mathbf{b}^T \mathbf{h}} \int \prod_i^M e^{-\frac{(\tilde{x}_i - a_i)^2}{2\sigma_i^2} + \frac{\tilde{x}_i w_{i*}^T \mathbf{h}}{\sigma_i^2}} d\tilde{\mathbf{x}} \\ &= \frac{1}{Z_{GB}} e^{\mathbf{b}^T \mathbf{h}} \left(\int e^{-\frac{(\tilde{x}_1 - a_1)^2}{2\sigma_1^2} + \frac{\tilde{x}_1 w_{1*}^T \mathbf{h}}{\sigma_1^2}} d\tilde{x}_1 \right. \\ &\quad \times \left. \int e^{-\frac{(\tilde{x}_2 - a_2)^2}{2\sigma_2^2} + \frac{\tilde{x}_2 w_{2*}^T \mathbf{h}}{\sigma_2^2}} d\tilde{x}_2 \right. \\ &\quad \times \dots \end{aligned}$$

Conditional Probability Density Functions

We finish by deriving the conditional probabilities.

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

Hidden units

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

$$\begin{aligned} p_{GB}(h_j = 1 | \mathbf{x}) &= \frac{e^{b_j + (\frac{\mathbf{x}}{\sigma^2})^T \mathbf{w}_{*j}}}{1 + e^{b_j + (\frac{\mathbf{x}}{\sigma^2})^T \mathbf{w}_{*j}}} \\ &= \frac{1}{1 + e^{-b_j - (\frac{\mathbf{x}}{\sigma^2})^T \mathbf{w}_{*j}}} \\ p_{GB}(h_j = 0 | \mathbf{x}) &= \frac{1}{1 + e^{b_j + (\frac{\mathbf{x}}{\sigma^2})^T \mathbf{w}_{*j}}}. \end{aligned}$$

Visible units

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

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

Comments

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 \mathbf{h} is a normal distribution with mean $b_i + \mathbf{w}_{i*}^T \mathbf{h}$ and variance σ_i^2 .