Machine learning approaches for quantum mechanical problems

Morten Hjorth-Jensen[1,2]

Department of Physics and Center for Computing in Science Education,
University of Oslo, Norway[1]

Department of Physics and Astronomy and Facility for Rare Isotope Beams,
Michigan State University, East Lansing, Michigan, USA[2]

PhysML workshop, Oslo, Norway, May 14-16, 2024

What is this talk about?

The main aim is to give you a short and hopefully pedestrian introduction to machine learning methods for solving quantum mechanical many-body problems.
The first part has an emphasis on both generative and discriminative methods while the second part of the talk introduces a new method dubbed **Parametric Matrix Models**, see https://arxiv.org/abs/2401.11694.

These slides and more at
https://github.com/mhjensenseminars/MachineLearningTalk/tree/master/doc/pub/sintefml

Thanks to many

Jane Kim (MSU/Ohio U), Julie Butler (MSU/Mt Union), Patrick Cook (MSU), Danny Jammooa (MSU), Dean Lee (MSU), Daniel Lee (Cornell), Even Nordhagen (UiO), Bryce Fore (ANL), Alessandro Lovato (ANL), Stefano Gandolfi (LANL), Francesco Pederiva (UniTN), and Giuseppe Carleo (EPFL).

And sponsors

1. National Science Foundation, USA (various grants)
2. Department of Energy, USA (various grants)
3. Research Council of Norway (various grants) and my employers University of Oslo and Michigan State University

Background

1. I have my training and research in traditional many-body theories and have studied and coded almost all, FCI, mean-field methods, Coupled Cluster theories, Green's function approaches, many-body perturbation theories, Monte Carlo methods and also molecular dynamics. Time independent and time-dependent theories
2. Central keywords: Effective degrees of freedom and dimensionality reduction
3. Last 6-8 years focus on quantum computing and machine learning
4. Developed many courses (regular and intensive) on many-body physics, computational physics, machine learning and quantum computing, see https://github.com/mhjensen

Selected references

- A high-bias, low-variance introduction to Machine Learning for physicists, Mehta et al., Physics Reports **810**, 1 (2019), https://www.sciencedirect.com/science/article/pii/S0370157319300766?via%3Dihub.
- Machine Learning and the Physical Sciences by Carleo et al., Reviews of Modern Physics **91**, 045002 (2019), https://link.aps.org/doi/10.1103/RevModPhys.91.045002
- Artificial Intelligence and Machine Learning in Nuclear Physics, Amber Boehnlein et al., Reviews Modern of Physics **94**, 031003 (2022), https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.94.031003
- Dilute neutron star matter from neural-network quantum states by Fore et al, Physical Review Research **5**, 033062 (2023), https://journals.aps.org/prresearch/pdf/10.1103/PhysRevResearch.5.033062

Selected references

- Neural-network quantum states for ultra-cold Fermi gases, Jane Kim et al, Nature Communications Physics **7**, 148 (2024), https://www.nature.com/articles/s42005-024-01613-w
- Message-Passing Neural Quantum States for the Homogeneous Electron Gas, Gabriel Pescia, Jane Kim et al. arXiv.2305.07240,":"https://doi.org/10.48550/arXiv.2305.07240"
- Parametric Matrix Models, Patrick Cook, Danny Jammooa, MHJ, Dean Lee and Daniel Lee, https://arxiv.org/abs/2401.11694.

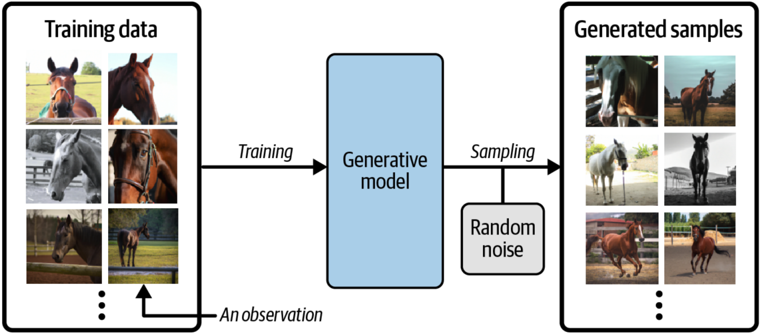
Extrapolations and model interpretability

When you hear phrases like **predictions and estimations** and **correlations and causations**, what do you think of?
May be you think of the difference between classifying new data points and generating new data points.
Or perhaps you consider that correlations represent some kind of symmetric statements like if A is correlated with B, then B is correlated with A. Causation on the other hand is directional, that is if A causes B, B does not necessarily cause A.

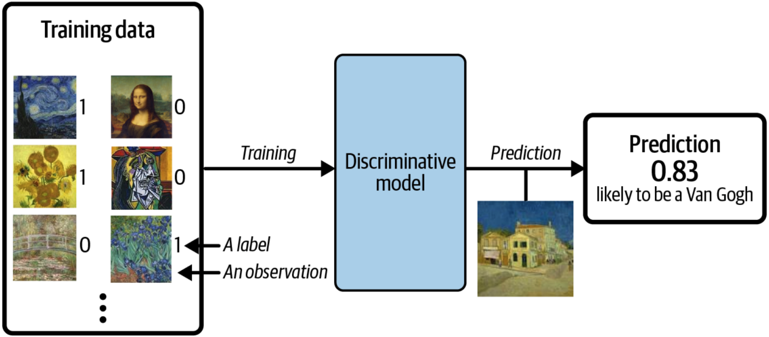
Generative and discriminative models

1. Balance between tractability and flexibility
2. We want to extract information about correlations, to make predictions, quantify uncertainties and express causality
3. How do we represent reliably our effective degrees of freedom?

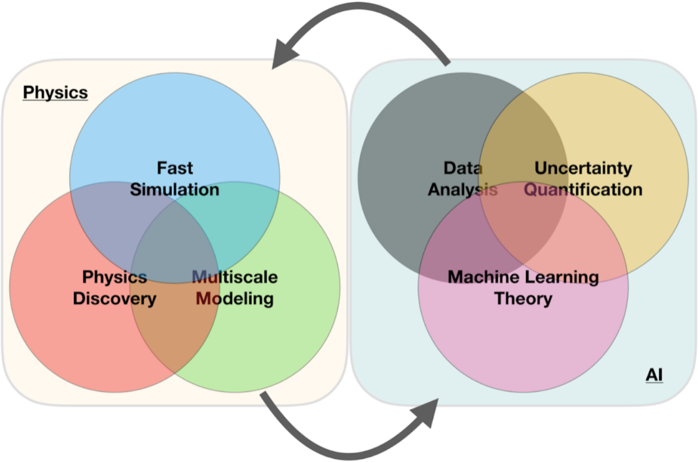
Example of generative modeling, taken from Generative Deep Learning by David Foster



Example of discriminative modeling, taken from Generative Deeep Learning by David Foster



Machine learning. A simple perspective on the interface between ML and Physics



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

$$\langle E \rangle = \frac{\int d\boldsymbol{R}\Psi^{\ast}_T(\boldsymbol{R})H(\boldsymbol{R})\Psi_T(\boldsymbol{R})}{\int d\boldsymbol{R}\Psi^{\ast}_T(\boldsymbol{R})\Psi_T(\boldsymbol{R})},$$

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

$$E_0 \le \langle E \rangle.$$

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 $\psi_T(\boldsymbol{R})$.

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

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

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

Quantum Monte Carlo Motivation

Define a new quantity

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

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

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

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 $\langle E[\boldsymbol{\alpha}]\rangle = \langle E_L\rangle$)

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

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 evaluted using MC intergration (with all its possible error sources). Use methods like stochastic gradient or other minimization methods to find the optimal parameters.

Why Feed Forward Neural Networks (FFNN)?

According to the *Universal approximation theorem*, a feed-forward neural network with just a single hidden layer containing a finite number of neurons can approximate a continuous multidimensional function to arbitrary accuracy, assuming the activation function for the hidden layer is a **non-constant, bounded and monotonically-increasing continuous function**.

Universal approximation theorem

The universal approximation theorem plays a central role in deep learning. Cybenko (1989) showed the following:

Let σ be any continuous sigmoidal function such that

$$\sigma(z) = \begin{cases} 1 & z \rightarrow \infty \\ 0 & z \rightarrow -\infty \end{cases}$$

Given a continuous and deterministic function $F(\boldsymbol{x})$ on the unit cube in d-dimensions $F \in [0,1]^d$, $x \in [0,1]^d$ and a parameter $\epsilon > 0$, there is a one-layer (hidden) neural network $f(\boldsymbol{x}; \boldsymbol{\Theta})$ with $\boldsymbol{\Theta} = (\boldsymbol{W}, \boldsymbol{b})$ and $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$, for which

$$|F(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\Theta})| < \epsilon \, \forall \boldsymbol{x} \in [0,1]^d.$$

The approximation theorem in words

Any continuous function $y = F(\boldsymbol{x})$ supported on the unit cube in d-dimensions can be approximated by a one-layer sigmoidal network to arbitrary accuracy.
Hornik (1991) extended the theorem by letting any non-constant, bounded activation function to be included using that the expectation value

$$\mathbb{E}[|F(\boldsymbol{x})|^2] = \int_{\boldsymbol{x}\in D} |F(\boldsymbol{x})|^2 p(\boldsymbol{x}) d\boldsymbol{x} < \infty.$$

Then we have

$$\mathbb{E}[|F(\boldsymbol{x}) - f(\boldsymbol{x};\boldsymbol{\Theta})|^2] = \int_{\boldsymbol{x}\in D} |F(\boldsymbol{x}) - f(\boldsymbol{x};\boldsymbol{\Theta})|^2 p(\boldsymbol{x}) d\boldsymbol{x} < \epsilon.$$

More on the general approximation theorem

None of the proofs give any insight into the relation between the number of of hidden layers and nodes and the approximation error ϵ, nor the magnitudes of $\boldsymbol{W}$ and $\boldsymbol{b}$.
Neural networks (NNs) have what we may call a kind of universality no matter what function we want to compute.

It does not mean that an NN can be used to exactly compute any function. Rather, we get an approximation that is as good as we want.

Class of functions we can approximate

The class of functions that can be approximated are the continuous ones. If the function $F(\boldsymbol{x})$ is discontinuous, it won't in general be possible to approximate it. However, an NN may still give an approximation even if we fail in some points.

Illustration of a single perceptron model and an FFNN

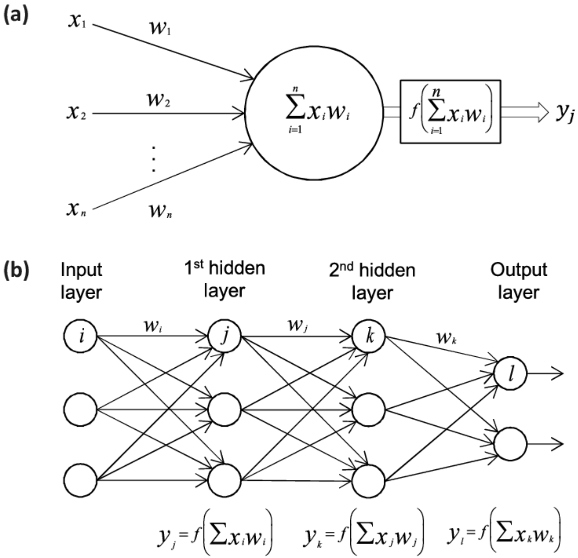


Figure: In a) we show a single perceptron model while in b) we dispay a network with two hidden layers, an input layer and an output layer.

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:

$$
\begin{aligned}
H_{LO} = & -\sum_i \frac{\vec{\nabla}_i^2}{2m_N} + \sum_{i<j} \left(C_1 + C_2\, \vec{\sigma_i} \cdot \vec{\sigma_j}\right) e^{-r_{ij}^2 \Lambda^2/4} \\
& + D_0 \sum_{i<j<k} \sum_{\text{cyc}} e^{-\left(r_{ik}^2 + r_{ij}^2\right)\Lambda^2/4}\,, \qquad (1)
\end{aligned}
$$

where m_N is the mass of the nucleon, $\vec{\sigma_i}$ is the Pauli matrix acting on nucleon i, and $\sum_{\text{cyc}}$ 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

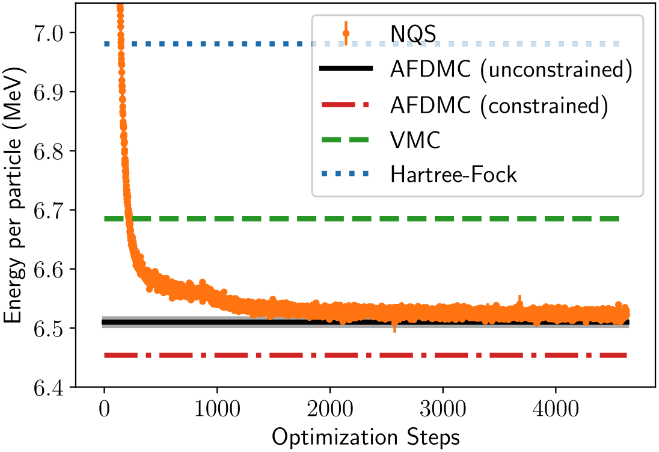
Deep learning neural networks, Variational Monte Carlo calculations of $A \leq 4$ nuclei with an artificial neural-network correlator ansatz by Adams et al.

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

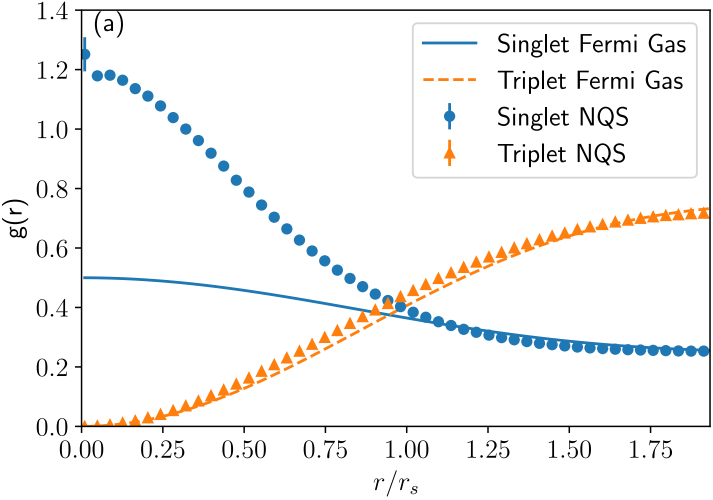
$$|\Psi_V^J\rangle = \prod_{i<j<k}\Big(1-\sum_{\text{cyc}} u(r_{ij})u(r_{jk})\Big)\prod_{i<j} f(r_{ij})|\Phi\rangle\,, \tag{2}$$

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

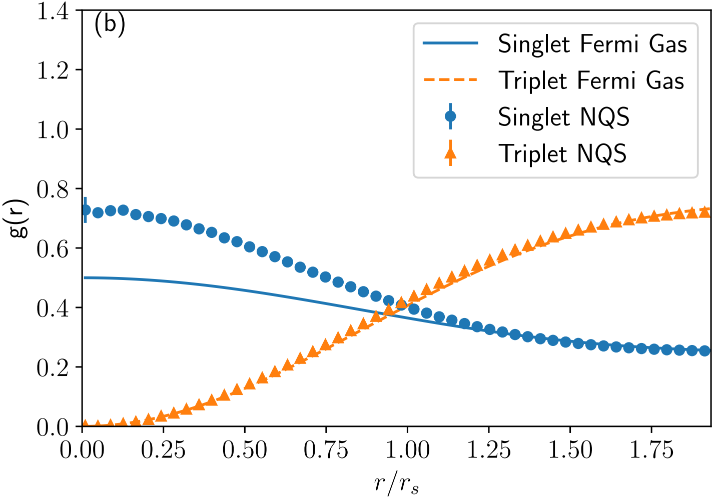
Dilute neutron star matter from neural-network quantum states by Fore et al, Physical Review Research 5, 033062 (2023) at density $\rho = 0.04\ \mathrm{fm}^{-3}$



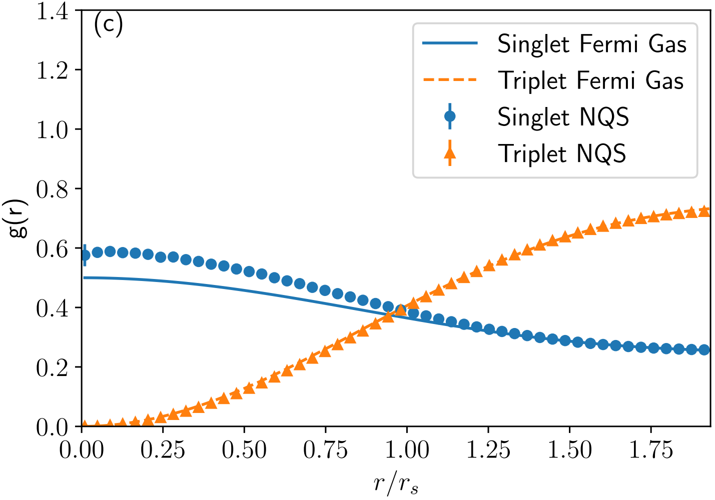
Pairing and Spin-singlet and triplet two-body distribution functions at $\rho = 0.01$ fm^{-3}



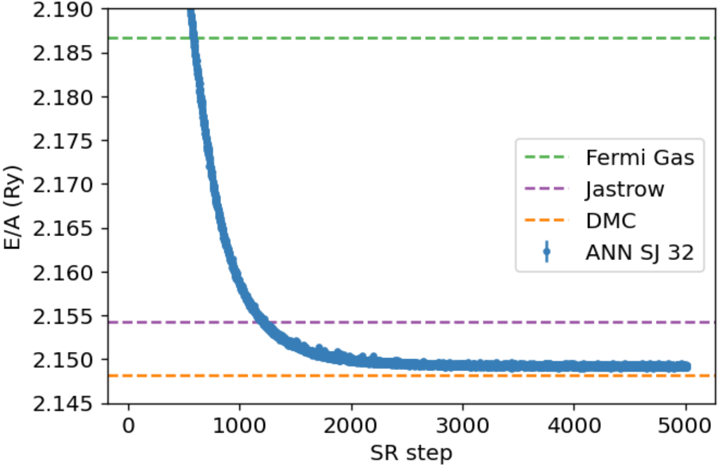
Pairing and Spin-singlet and triplet two-body distribution functions at $\rho = 0.04\ \text{fm}^{-3}$



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



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,



Generical approaches to probability models

We define a probability

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

where $f(x_i, h_j; \boldsymbol{\Theta})$ is a function which we assume is larger or equal than zero and obeys all properties required for a probability distribution and $Z(\boldsymbol{\Theta})$ 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)

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

Marginal and conditional probabilities

We can in turn define the marginal probabilities

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

and

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

Change of notation

Note the change to a vector notation. A variable like $\boldsymbol{x}$ 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

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

changes to

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

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 $\boldsymbol{x}$ and $\boldsymbol{h}$ 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

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

which we rewrite as

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

Optimizing the logarithm instead

Computing the derivatives with respect to the parameters $\boldsymbol{\Theta}$ is easier (and equivalent) with taking the logarithm of the probability. We will thus optimize

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

which leads to

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

Expression for the gradients

This leads to the following equation

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

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.

Final expression

Taking the derivative gives us

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

which is the expectation value of $\log f$

$$\nabla_{\mathbf{\Theta}} \log Z(\mathbf{\Theta}) = \sum_{x_i \sim p} p(x_i; \mathbf{\Theta}) \nabla_{\mathbf{\Theta}} \log f(x_i; \mathbf{\Theta}),$$

that is

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

This quantity is evaluated using Monte Carlo sampling, with Gibbs sampling as the standard sampling rule.

Final expression for the gradients

This leads to the following equation

$$\nabla_{\boldsymbol{\Theta}} \log p(\boldsymbol{X}; \boldsymbol{\Theta}) = \nabla_{\boldsymbol{\Theta}} \left(\sum_{x_i \in \boldsymbol{X}} \log f(x_i; \boldsymbol{\Theta}) \right) - \mathbb{E}_{x \sim p}(\log f(x_i; \boldsymbol{\Theta})) = 0.$$

Introducing the energy model

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

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

where $f(\boldsymbol{x},\boldsymbol{h};\boldsymbol{\Theta})$ 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

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

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

More compact notation

With the above definition we can write the probability as

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

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

Examples of gradient expressions

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

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

and

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

and

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

Network Elements, the energy function

The function $E(\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{\Theta})$ gives the **energy** of a configuration (pair of vectors) $(\boldsymbol{x}, \boldsymbol{h})$. The lower the energy of a configuration, the higher the probability of it. This function also depends on the parameters $\boldsymbol{a}$, $\boldsymbol{b}$ 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 $E(\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{\Theta})$. 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:

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

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

Gaussian-binary RBM

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

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

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

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,

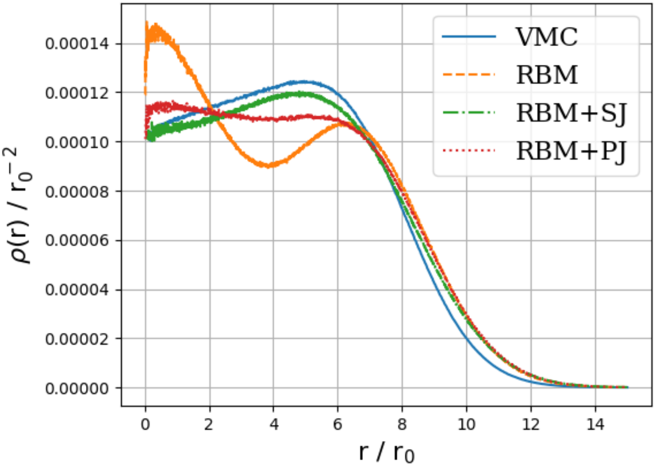
$$\hat{V} = \sum_{i<j}^{N} \frac{1}{r_{ij}},$$

with $r_{ij} = \sqrt{\mathrm{r}_i^2 - \mathrm{r}_j^2}$.
Separable Hamiltonian with the relative motion part ($r_{ij} = r$)

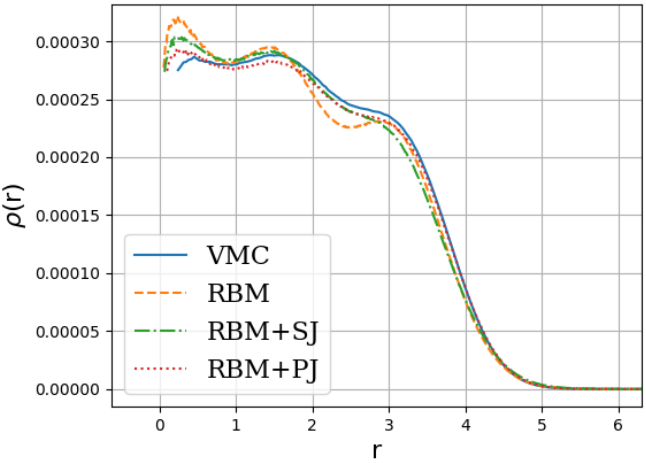
$$\hat{H}_r = -\nabla_r^2 + \frac{1}{4}\omega^2 r^2 + \frac{1}{r},$$

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

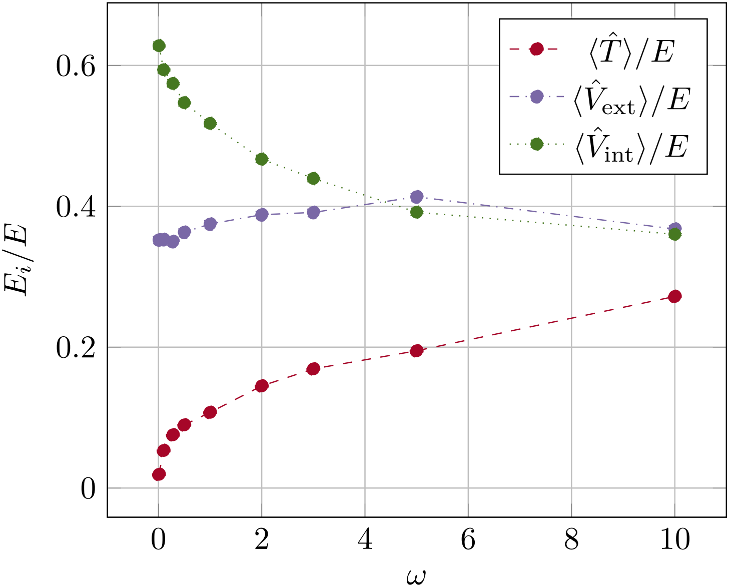
Quantum dots and Boltzmann machines, onebody densities $N = 6$, $\hbar\omega = 0.1$ a.u.



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



Expectation values as functions of the oscillator frequency



Parametric Matrix Models

Given data for k energies and k observables in the ground state of a Hamiltonian that is a function of some coupling consents

$$H(\boldsymbol{c}) = H_0 + \sum_j c_j H_j$$

$$\hat{y}(\boldsymbol{c}) = [\hat{E}_k(\boldsymbol{c}), \langle \psi_0(\boldsymbol{c}) | \hat{O}_k | \psi_0(\boldsymbol{c}) \rangle]$$

Model with same structure

We form a PMM with same structure

$$M(\boldsymbol{c}) = \underline{M_0} + \sum_j c_j \underline{M_j}$$

and calculate its k energies and k ovservables in the PMM ground state

$$y(\boldsymbol{c}) = [E_k(\boldsymbol{c}), \langle \phi_0(\boldsymbol{c}) | \underline{O}_k | \phi_0(\boldsymbol{c}) \rangle] \tag{3}$$

where $\underline{M_0}, \underline{M_j}, \underline{O} \in \mathrm{C}^{\overline{n} \times \overline{n}}$ are Hermitian matrices.

Simple labeling

We label the trainable matrices, vectors, and scalars of the PMM by an underline. Likewise denote hyperparameters of the PMM by an overline. The trainable parameters are trained by minimizing the mean squared error over all N training points.

$$\mathcal{L} = \frac{1}{N} \sum_{i}^{N} (\hat{y}_k(\boldsymbol{c}_i) - y_k(\boldsymbol{c}_i))^2,$$

where $\hat{y}_k(\boldsymbol{c})$ is the true data, and $y_k(\boldsymbol{c})$ corresponds to the output of the PMM.

Parametric matrix models for regression type problems

Given dataset $X \in \mathbb{R}^{n\times m}$, we compute a "Hamiltonian" that is linear in the input features x_i

$$H(X) = \underline{H_0} + \sum_i^m x_i \underline{H_i},$$

Where $H_i \in \mathbb{C}^{N\times N}$.

Calculating eigensystem

The eigensystem is then calculated for H

$$H_k = V \Lambda V^\dagger = \begin{bmatrix} | & | & \cdots & | \\ v_1 & v_2 & \cdots & v_l \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_l \end{bmatrix} \begin{bmatrix} | & | & \cdots & | \\ v_1 & v_2 & \cdots & v_l \\ | & | & \cdots & | \end{bmatrix}$$

Ordering eigenpairs

Ordering the eigenpairs by decreasing eigenvalue magnitude, $|\lambda_1^{(k)}| \geq |\lambda_2^{(k)}| \geq \cdots \geq |\lambda_l^{(k)}$, we select the first d-eigenvectors to "decode" into the penultimate output vector $\vec{z} \in \mathbb{R}^c$,

$$z_k = \underline{b_k} + \sum_{i \leq j}^{d} |\vec{v}_i^{\dagger} \underline{\Delta_{kij} |\vec{v}_j}|^2 - \frac{1}{2} |\underline{\Delta_{kij}}|_2^2,$$

where $\underline{\Delta_{kij}}$ are trainable Hermitian "decoder" matrices and $\underline{\vec{b}} \in \mathbb{R}^c$ is a trainable bias vector. The trainable parameters are trained by minimizing a loss function over all J training points.

$$\mathcal{L} = \sum_i |\hat{z}_i - z_i|^P$$

Where $\hat{z}$ is the true data, and z is the output correspnding to the PMM.