

# Machine learning approaches for quantum mechanical problems

Morten Hjorth-Jensen<sup>1,2</sup>

Department of Physics and Center for Computing in Science Education,  
University of Oslo, Norway<sup>1</sup>

Department of Physics and Astronomy and Facility for Rare Isotope Beams,  
Michigan State University, East Lansing, Michigan, USA<sup>2</sup>

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

# What is this talk about?

The main emphasis 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](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), Daniel Bazin (MSU), Dean Lee (MSU), Daniel Lee (Cornell), Even Nordhagen (UiO), Robert Solli (UiO, Expert Analytics), 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

## 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>
- ▶ Neural-network quantum states for ultra-cold Fermi gases, Jane Kim et al. Nature Communications Physics **7**, 148

## Extrapolations and model interpretability

When you hear phrases like **predictions and estimations** and **correlations and causations**, what do you think of?

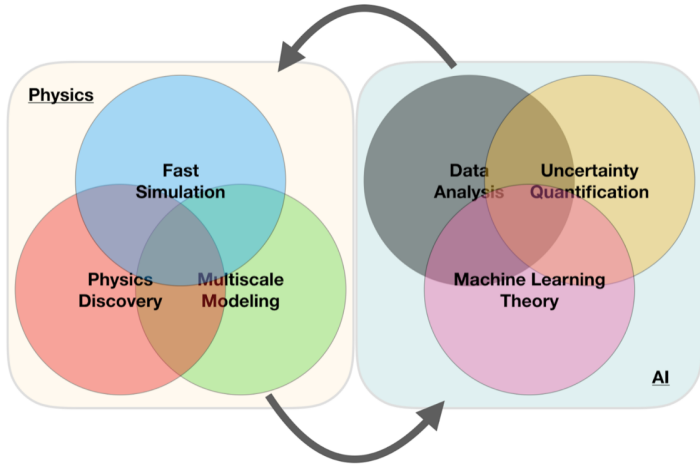
May be you think of the difference between classifying new data points and generating new data points.

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?

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





# 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) ones on many-body physics, computational physics, machine learning and quantum computing, see <https://github.com/mhjensen>

# Many-body physics, Quantum Monte Carlo and deep learning

Given a hamiltonian  $H$  and a trial wave function  $\Psi_T$ , 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  $\alpha$ . 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  $\alpha_i$ . 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

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

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

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

and

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

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.

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

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



## The approximation theorem in words

**Any continuous function  $y = F(\mathbf{x})$  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

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

Then we have

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

## More on the general approximation theorem

None of the proofs give any insight into the relation between the number of hidden layers and nodes and the approximation error  $\epsilon$ , nor the magnitudes of  $\mathbf{W}$  and  $\mathbf{b}$ .

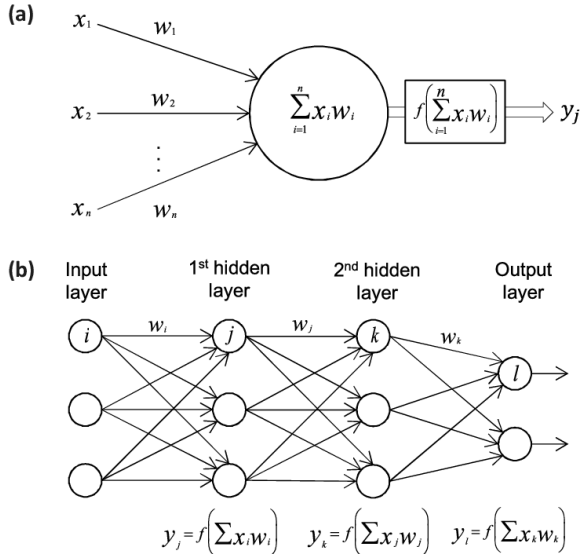
Neural networks (NNs) have what we may call a kind of universality no matter what function we want to compute.

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  $F(\mathbf{x})$  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 display 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:**

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

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

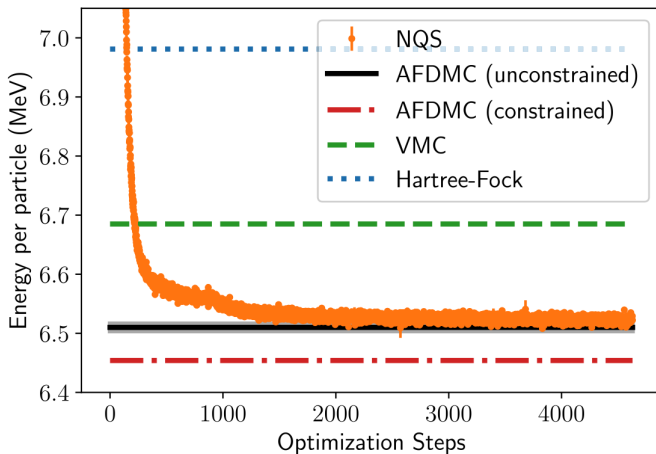
## Deep learning neural networks, Variational Monte Carlo calculations of $A \leq 4$ nuclei with an artificial neural-network correlator ansatz by Adams et al.

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

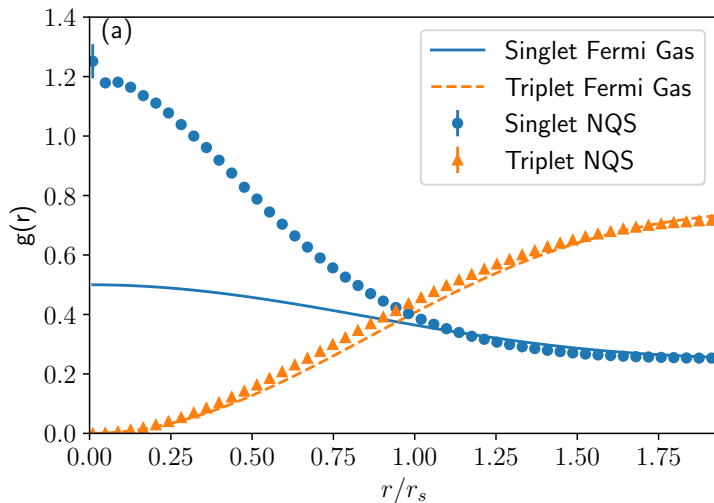
$$|\Psi_V^J\rangle = \prod_{i < j < k} \left( 1 - \sum_{\text{cyc}} u(r_{ij})u(r_{jk}) \right) \prod_{i < j} f(r_{ij}) |\Phi\rangle, \quad (2)$$

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.

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

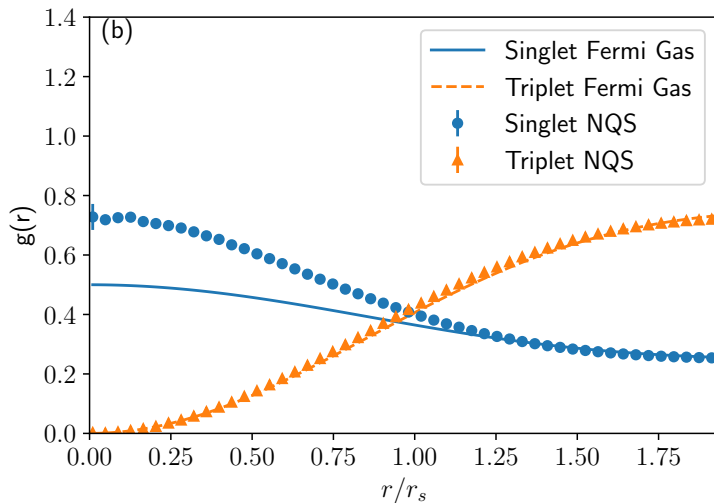


# Pairing and Spin-singlet and triplet two-body distribution functions at $\rho = 0.01 \text{ fm}^{-3}$

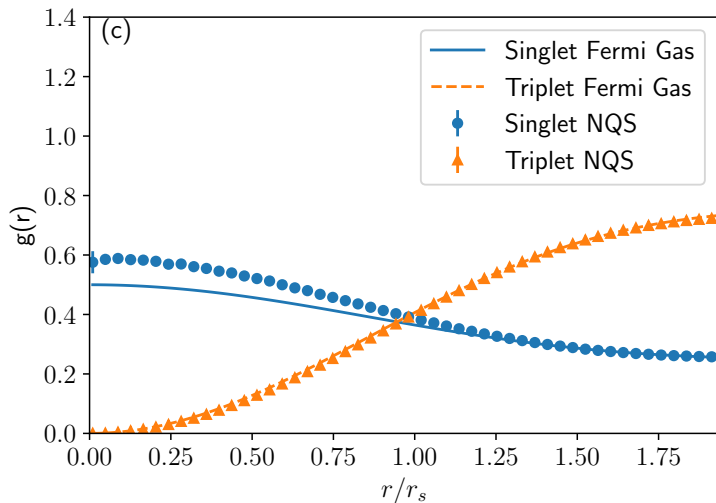




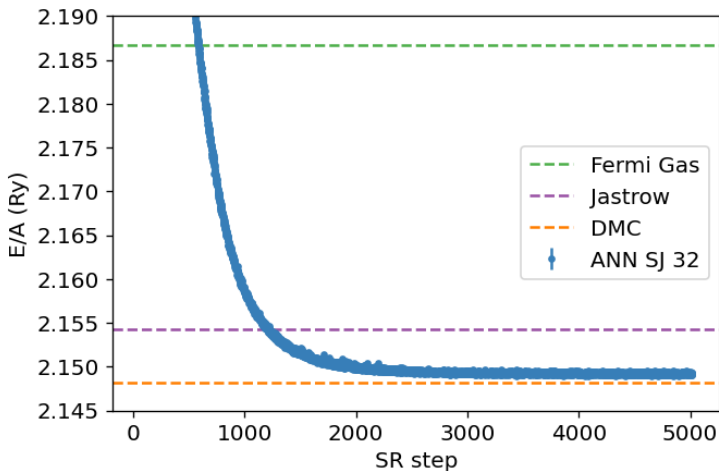
# Pairing and Spin-singlet and triplet two-body distribution functions at $\rho = 0.04 \text{ fm}^{-3}$



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



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,



# 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).

## Generative models: Why Boltzmann machines?

What is known as restricted Boltzmann Machines (RBM) have received a lot of attention lately. One of the major reasons is that they can be stacked layer-wise to build deep neural networks that capture complicated statistics.

The original RBMs had just one visible layer and a hidden layer, but recently so-called Gaussian-binary RBMs have gained quite some popularity in imaging since they are capable of modeling continuous data that are common to natural images.

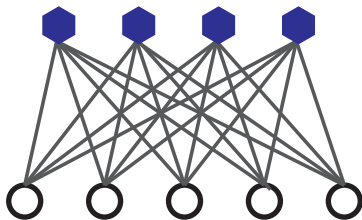
Furthermore, they have been used to solve complicated quantum mechanical many-particle problems or classical statistical physics problems like the Ising and Potts classes of models.

# The structure of the RBM network

Hidden Layer

Interactions

Visible Layer



$$b_{\mu}(h_{\mu})$$

$$W_{i\mu}v_ih_{\mu}$$

$$a_i(v_i)$$

# The network

## The network layers:

1. A function  $\mathbf{x}$  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  $\mathbf{h}$  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 length as  $\mathbf{h}$ .
3.  $\mathbf{W}$  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:

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

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

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

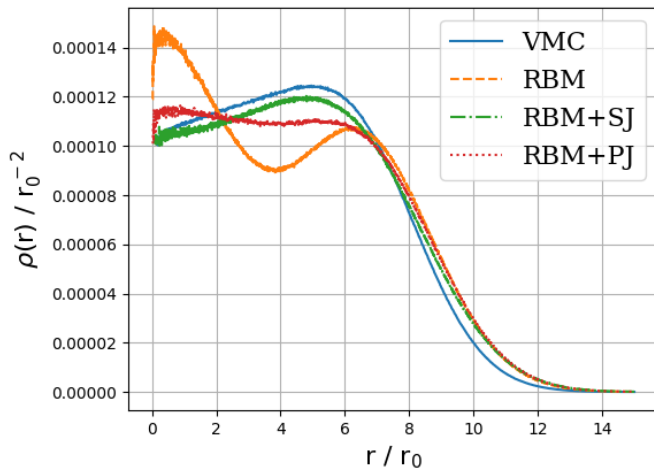
$$C_i = \frac{\partial \langle E_L \rangle}{\partial \theta_i} = 2(\langle E_L \frac{1}{\Psi} \frac{\partial \Psi}{\partial \theta_i} \rangle - \langle E_L \rangle \langle \frac{1}{\Psi} \frac{\partial \Psi}{\partial \theta_i} \rangle),$$

where the local energy is given by

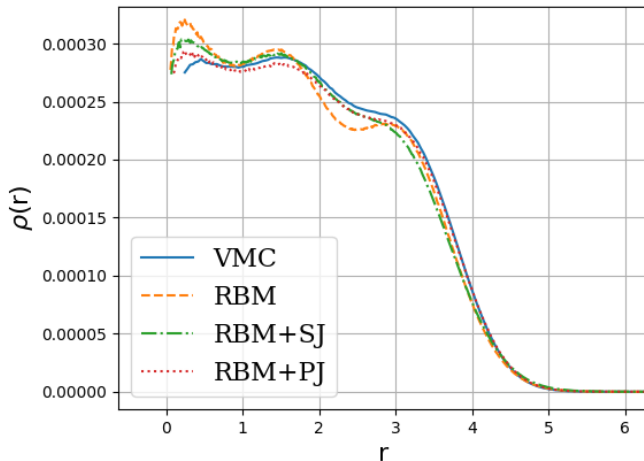
$$E_L = \frac{1}{\Psi} \hat{H} \Psi.$$

# Quantum dots and Boltzmann machines, onebody densities

$N = 6$ ,  $\hbar\omega = 0.1$  a.u.

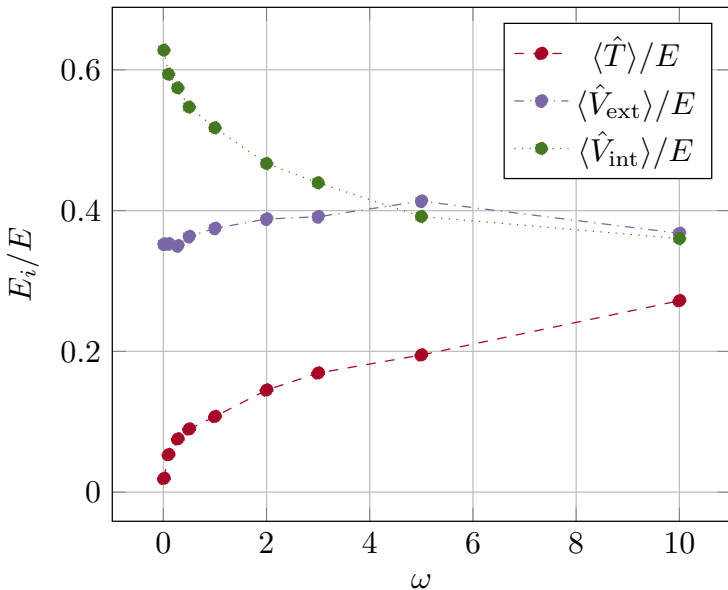


Onebody densities  $N = 30$ ,  $\hbar\omega = 1.0$  a.u.





## Expectation values as functions of the oscillator frequency



## Observations (or conclusions if you prefer)

- ▶ Need for AI/Machine Learning in physics, lots of ongoing activities
- ▶ To solve many complex problems and facilitate discoveries, multidisciplinary efforts are required involving scientists in physics, statistics, computational science, applied math and other fields.
- ▶ There is a need for focused AI/ML learning efforts that will benefit accelerator science and experimental and theoretical programs

## More observations

- ▶ How do we develop insights, competences, knowledge in statistical learning that can advance a given field?
  - ▶ For example: Can we use ML to find out which correlations are relevant and thereby diminish the dimensionality problem in standard many-body theories?
  - ▶ Can we use AI/ML in detector analysis, accelerator design, analysis of experimental data and more?
  - ▶ Can we use AI/ML to carry out reliable extrapolations by using current experimental knowledge and current theoretical models?
- ▶ The community needs to invest in relevant educational efforts and training of scientists with knowledge in AI/ML. These are great challenges to the CS and DS communities
- ▶ Quantum computing and quantum machine learning not discussed here
- ▶ Most likely tons of things I have forgotten

# Possible start to raise awareness about ML in our own field

- ▶ Make an ML challenge in your own field a la [Learning to discover: the Higgs boson machine learning challenge](#).  
Alternatively go to kaggle.com at <https://www.kaggle.com/c/higgs-boson>
- ▶ HEP@CERN and HEP in general have made significant impacts in the field of machine learning and AI. Something to learn from

# Essential elements of generative models

The aim of generative methods is to train a probability distribution  $p$ . The methods we will focus on are:

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

We 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 defined this function in terms of optimization parameters  $\Theta$ , which could be the biases and weights of a deep network, and a set of hidden variables we also assume to be random variables which also 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)

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

## Marginal and conditional probabilities

We can in turn define the marginal probabilities

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

and

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



## 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

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

which we rewrite as

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

## 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  $p(x_i; \Theta) = \sum_{h_j \in \mathbf{H}} f(x_i, h_j; \Theta)$ . 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  $\Theta$  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

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

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 MC<sup>2</sup>).

## 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

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

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

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

## Final expression

Taking the derivative gives us

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

which is the expectation value of  $\log f$

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

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.

## Binary-binary model

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

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