Artificial intelligence and quantum many-body problems

Morten Hjorth-Jensen[1]

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

QFC2024- Quantum gases, fundamental interactions and cosmology, Pisa, October 23-25, 2024

What is this talk about?

The main emphasis is to give you a short and pedestrian introduction to the whys and hows we can use (with several examples) machine learning methods to solve quantum mechanical many-body problemss. And why this could (or should) be of interest.

These slides and more at
`https://github.com/mhjensenseminars/MachineLearningTalk/tree/master/doc/pub/QFC2024`

Thanks to many

Jane Kim (MSU), Julie Butler (MSU), Patrick Cook (MSU), Danny Jammooa (MSU), Daniel Bazin (MSU), Dean Lee (MSU), Witek Nazarewicz (MSU), Michelle Kuchera (Davidson College), Even Nordhagen (UiO), Robert Solli (UiO, Expert Analytics), Bryce Fore (ANL), Alessandro Lovato (ANL), Stefano Gandolfi (LANL), Francesco Pederiva (UniTN), and Giuseppe Carleo (EPFL). Niyaz Beysengulov and Johannes Pollanen (experiment, MSU); Zachary Stewart, Jared Weidman, and Angela Wilson (quantum chemistry, MSU) Jonas Flaten, Oskar, Leinonen, Øyvind Sigmundson Schøyen, Stian Dysthe Bilek, and Håkon Emil Kristiansen (UiO). Marianne Bathen and Lasse Vines (experiments (UiO). Excuses to those I have omitted/forgotten.

And sponsors

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

Perhaps I should have talked about this instead....

How to use many-body theory to design quantum circuits (Quantum engineering)

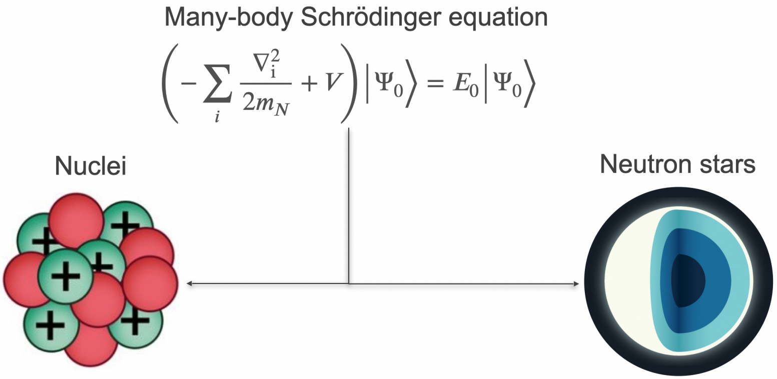
1. Many-body methods like F(ull)C(onfiguration)I(nteraction) theory, Coupled-Cluster theory and other with
   - Adaptive basis sets
   - Time dependence
   - Optimization of experimental parameters
   - Feedback from experiment
2. Finding optimal parameters for tuning of entanglement, see PRX Quantum **5**, 030324 (2024)
3. Numerical experiments to mimick real systems
4. Constructing quantum circuits to simulate specific systems
5. Quantum machine learning to optimize quantum circuits, see `https://arxiv.org/abs/2403.14406` and more

AI/ML and some statements you may have heard (and what do they mean?)

1. Fei-Fei Li on ImageNet: **map out the entire world of objects** (The data that transformed AI research)
2. Russell and Norvig in their popular textbook: **relevant to any intellectual task; it is truly a universal field** (Artificial Intelligence, A modern approach)
3. Woody Bledsoe puts it more bluntly: **in the long run, AI is the only science** (quoted in Pamilla McCorduck, Machines who think)

If you wish to have a critical read on AI/ML from a societal point of view, see Kate Crawford's recent text Atlas of AI.
Here: with AI/ML we intend a collection of machine learning methods with an emphasis on statistical learning and data analysis

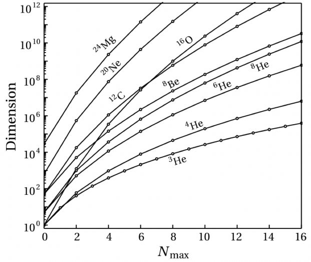
Qauntum mechanical many-body problems (nuclear example here)


Many-body Schrödinger equation
$\left(-\sum_i \frac{\nabla_i^2}{2m_N} + V\right)\left|\Psi_0\right\rangle = E_0\left|\Psi_0\right\rangle$
Nuclei
Neutron stars

Curse of dimensionality

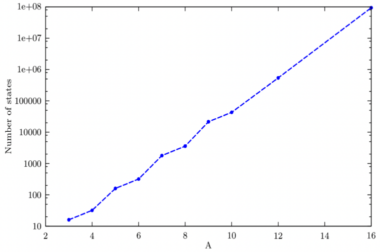
Configuration-interaction

$$\binom{N}{A} = \frac{N!}{(N-A)!A!}$$



Green's function Monte Carlo

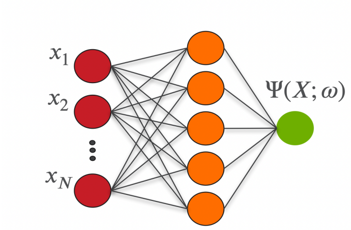
$$\lim_{\tau\to\infty} e^{-(H-E_0)\tau}\left|\Psi_T\right\rangle = c_o\left|\Psi_0\right\rangle$$

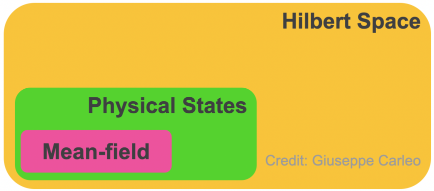


Neural network quantum states

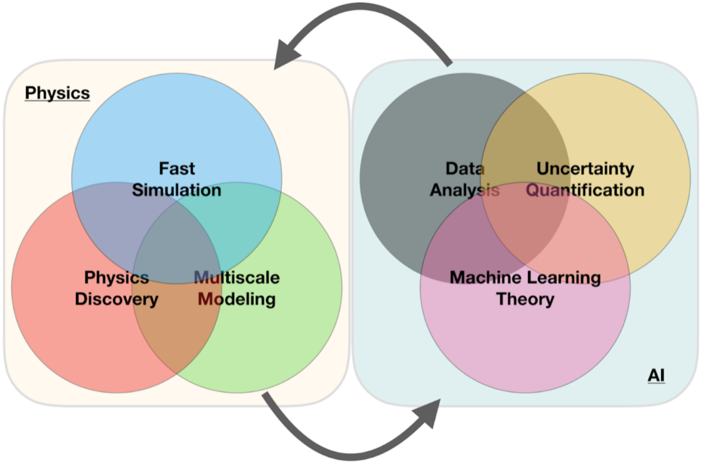
Neural networks compactly represent complex high-dimensional functions

Most quantum states of interest have distinctive features and intrinsic structures

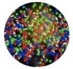




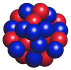
Machine learning. A simple perspective on the interface between ML and Physics

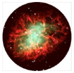


ML in Nuclear Physics (or any field, almost)

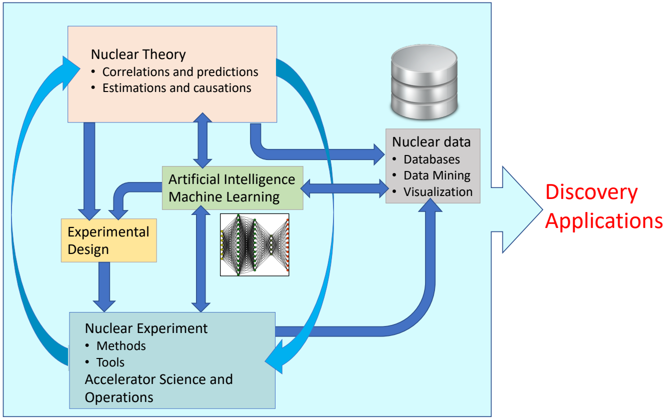

Hot and Dense Nuclear Matter

Hadrons


Atomic Nucleus


Nuclei in the Cosmos

Fundamental interactions



Types of machine learning

The approaches to machine learning are many, but are often split into two main categories. In *supervised learning* we know the answer to a problem, and let the computer deduce the logic behind it. On the other hand, *unsupervised learning* is a method for finding patterns and relationship in data sets without any prior knowledge of the system.
An important third category is *reinforcement learning*. This is a paradigm of learning inspired by behavioural psychology, where learning is achieved by trial-and-error, solely from rewards and punishment.

Main categories

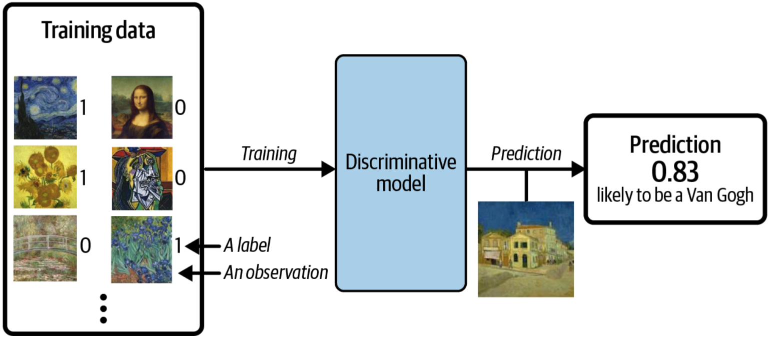
Another way to categorize machine learning tasks is to consider the desired output of a system. Some of the most common tasks are:

- Classification: Outputs are divided into two or more classes. The goal is to produce a model that assigns inputs into one of these classes. An example is to identify digits based on pictures of hand-written ones. Classification is typically supervised learning.
- Regression: Finding a functional relationship between an input data set and a reference data set. The goal is to construct a function that maps input data to continuous output values.
- Clustering: Data are divided into groups with certain common traits, without knowing the different groups beforehand. It is thus a form of unsupervised learning.

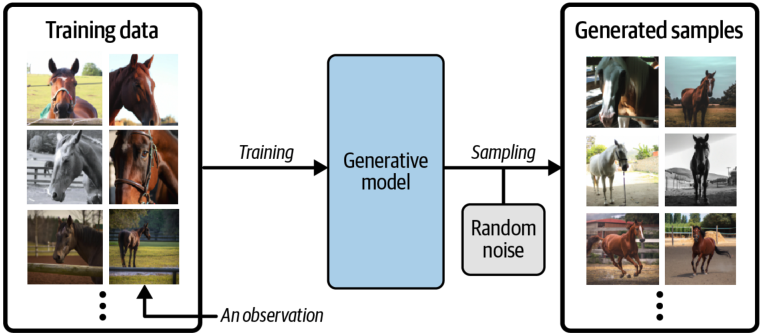
The plethora of machine learning algorithms/methods

1. Deep learning: Neural Networks (NN), Convolutional NN, Recurrent NN, Boltzmann machines, autoencoders and variational autoencoders and generative adversarial networks, stable diffusion and many more generative models
2. Bayesian statistics and Bayesian Machine Learning, Bayesian experimental design, Bayesian Regression models, Bayesian neural networks, Gaussian processes and much more
3. Dimensionality reduction (Principal component analysis), Clustering Methods and more
4. Ensemble Methods, Random forests, bagging and voting methods, gradient boosting approaches
5. Linear and logistic regression, Kernel methods, support vector machines and more
6. Reinforcement Learning; Transfer Learning and more

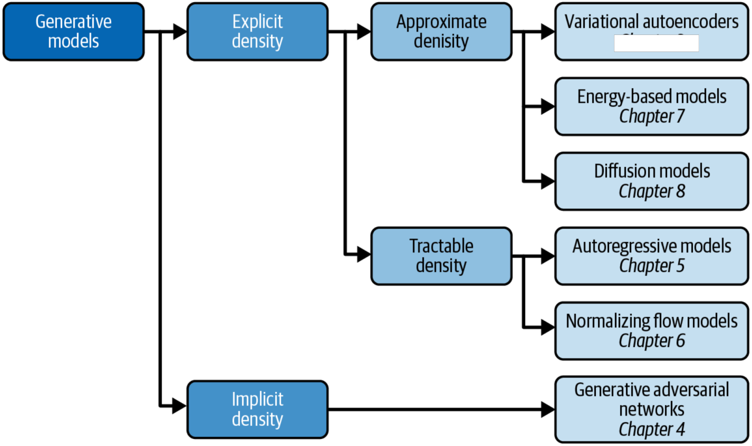
Example of discriminative modeling, taken from Generative Deep Learning by David Foster



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



Taxonomy of generative deep learning, taken from Generative Deep Learning by David Foster



Good books with hands-on material and codes

- Sebastian Rashcka et al, Machine learning with Sickit-Learn and PyTorch
- David Foster, Generative Deep Learning with TensorFlow
- Babcock and Gavras, Generative AI with Python and TensorFlow 2

All three books have GitHub sites from where one can download all codes. A good and more general text (2016) is Goodfellow, Bengio and Courville, Deep Learning

More references

Reading on diffusion models

1. A central paper is the one by Sohl-Dickstein et al, Deep Unsupervised Learning using Nonequilibrium Thermodynamics, https://arxiv.org/abs/1503.03585
2. See also Diederik P. Kingma, Tim Salimans, Ben Poole, Jonathan Ho, Variational Diffusion Models, https://arxiv.org/abs/2107.00630

and VAEs

1. An Introduction to Variational Autoencoders, by Kingma and Welling, see https://arxiv.org/abs/1906.02691

And two Nobel prizes this year!

What are the basic Machine Learning ingredients?

Almost every problem in ML and data science starts with the same ingredients:

- The dataset $\boldsymbol{x}$ (could be some observable quantity of the system we are studying)
- A model which is a function of a set of parameters $\boldsymbol{\alpha}$ that relates to the dataset, say a likelihood function $p(\boldsymbol{x}|\boldsymbol{\alpha})$ or just a simple model $f(\boldsymbol{\alpha})$
- A so-called **loss/cost/risk** function $\mathcal{C}(\boldsymbol{x}, f(\boldsymbol{\alpha}))$ which allows us to decide how well our model represents the dataset.

We seek to minimize the function $\mathcal{C}(\boldsymbol{x}, f(\boldsymbol{\alpha}))$ by finding the parameter values which minimize $\mathcal{C}$. This leads to various minimization algorithms. It may surprise many, but at the heart of all machine learning algortihms there is an optimization problem.

Low-level machine learning, the family of ordinary least squares methods

Our data which we want to apply a machine learning method on, consist of a set of inputs $\boldsymbol{x}^T = [x_0, x_1, x_2, \dots, x_{n-1}]$ and the outputs we want to model $\boldsymbol{y}^T = [y_0, y_1, y_2, \dots, y_{n-1}]$. We assume that the output data can be represented (for a regression case) by a continuous function f through

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon.$$

Setting up the equations

In linear regression we approximate the unknown function with another continuous function $\tilde{\boldsymbol{y}}(\boldsymbol{x})$ which depends linearly on some unknown parameters $\boldsymbol{\theta}^T = [\theta_0, \theta_1, \theta_2, \dots, \theta_{p-1}]$.
The input data can be organized in terms of a so-called design matrix with an approximating function $\tilde{\boldsymbol{y}}$

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\theta},$$

The objective/cost/loss function

The simplest approach is the mean squared error

$$C(\boldsymbol{\Theta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \frac{1}{n}\left\{\left(\boldsymbol{y}-\boldsymbol{\tilde{y}}\right)^T\left(\boldsymbol{y}-\boldsymbol{\tilde{y}}\right)\right\},$$

or using the matrix $\boldsymbol{X}$ and in a more compact matrix-vector notation as

$$C(\boldsymbol{\Theta}) = \frac{1}{n}\left\{\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}\right)^T\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}\right)\right\}.$$

This function represents one of many possible ways to define the so-called cost function.

Training solution

Optimizing with respect to the unknown parameters θ_j we get

$$\boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta},$$

and if the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is invertible we have the optimal values

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

We say we 'learn' the unknown parameters $\boldsymbol{\theta}$ from the last equation.

Selected references

- Mehta et al. and Physics Reports (2019).
- Machine Learning and the Physical Sciences by Carleo et al
- Artificial Intelligence and Machine Learning in Nuclear Physics, Amber Boehnlein et al., Reviews Modern of Physics 94, 031003 (2022)
- Dilute neutron star matter from neural-network quantum states by Fore et al, Physical Review Research 5, 033062 (2023)
- Neural-network quantum states for ultra-cold Fermi gases, Jane Kim et al, Commun Phys 7, 148 (2024), see `https://doi.org/10.48550/arXiv.2305.08831`
- Particle Data Group summary on ML methods

Scientific Machine Learning

An important and emerging field is what has been dubbed as scientific ML, see the article by Deiana et al "Applications and Techniques for Fast Machine Learning in Science, Big Data **5**, 787421 (2022):https://doi.org/10.3389/fdata.2022.787421"

The authors discuss applications and techniques for fast machine learning (ML) in science – the concept of integrating power ML methods into the real-time experimental data processing loop to accelerate scientific discovery. The report covers three main areas

1. applications for fast ML across a number of scientific domains;
2. techniques for training and implementing performant and resource-efficient ML algorithms;
3. and computing architectures, platforms, and technologies for deploying these algorithms.

And more

- An important application of AI/ML methods is to improve the estimation of bias or uncertainty due to the introduction of or lack of physical constraints in various theoretical models.
- In theory, we expect to use AI/ML algorithms and methods to improve our knowledge about correlations of physical model parameters in data for quantum many-body systems. Deep learning methods show great promise in circumventing the exploding dimensionalities encountered in quantum mechanical many-body studies.
- Merging a frequentist approach (the standard path in ML theory) with a Bayesian approach, has the potential to infer better probabilitity distributions and error estimates.
- Machine Learning and Quantum Computing is a very interesting avenue

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

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

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

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

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

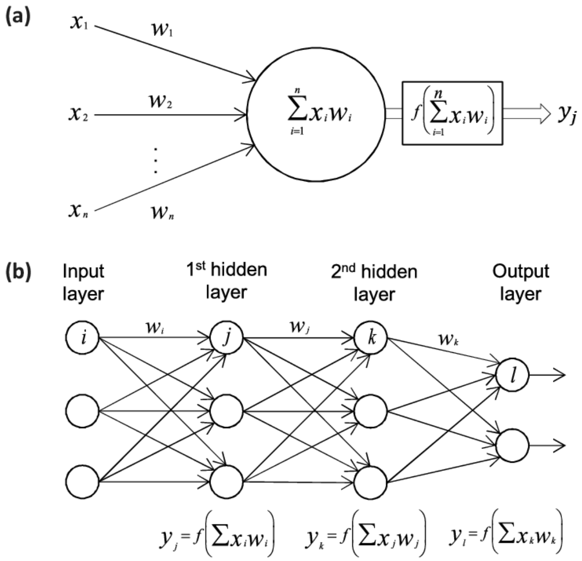


Figure: In a) we show a single perceptron model while in b) we dispay a network with two hidden layers, an input layer and an output layer.

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:

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

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

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

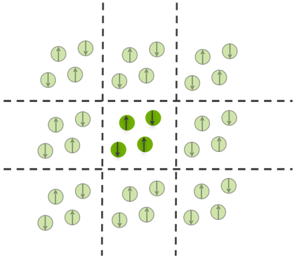
which is commonly used for nuclear Hamiltonians that do not contain tensor and spin-orbit terms. The above function is replaced by a four-layer Neural Network.

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

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

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

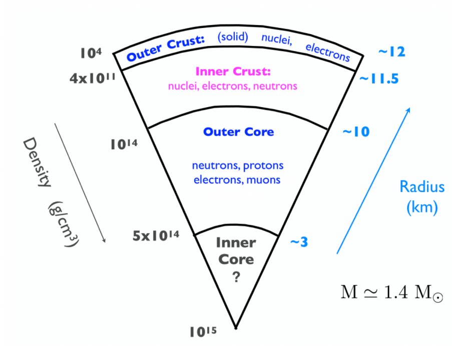


- Periodic boundary conditions and coordinate system

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

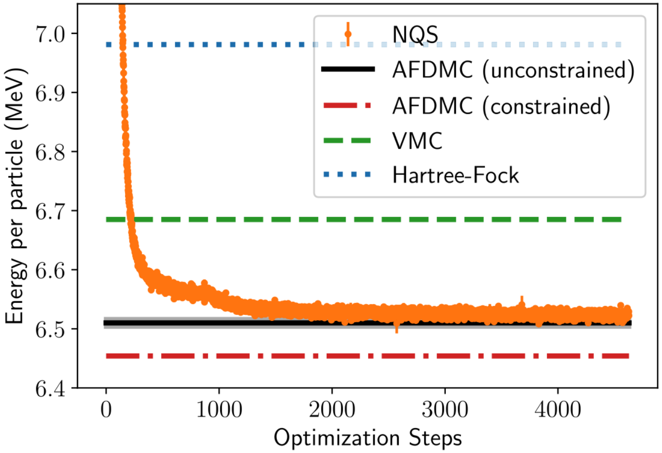
- Potential energy contribution from particle images
- Remove Coulomb potential

Neutron star structure

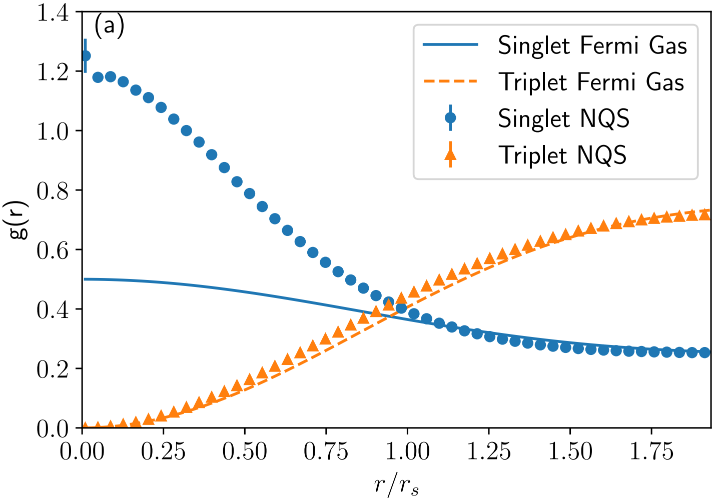


- Mostly neutrons but composition varies with density
- Nuclei in crust are squeezed into uniform matter in core
- Likely neutron superfluid in inner crust and outer core
- Calculations currently focus on inner crust

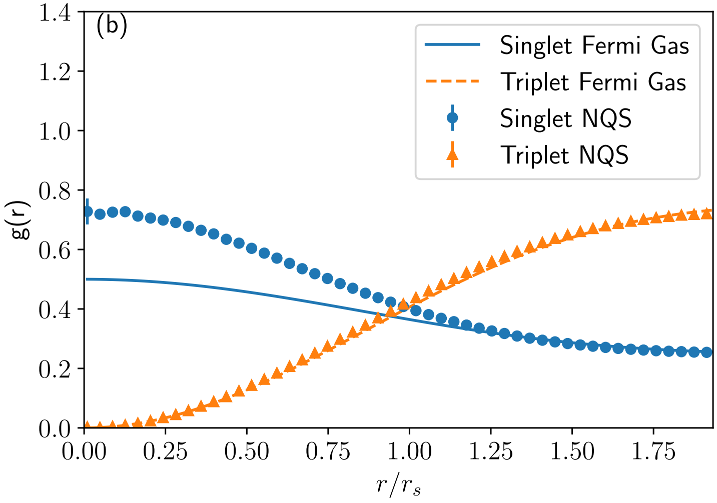
Dilute neutron star matter from neural-network quantum states by Fore et al, Physical Review Research 5, 033062 (2023) at density $\rho = 0.04\ \mathrm{fm}^{-3}$



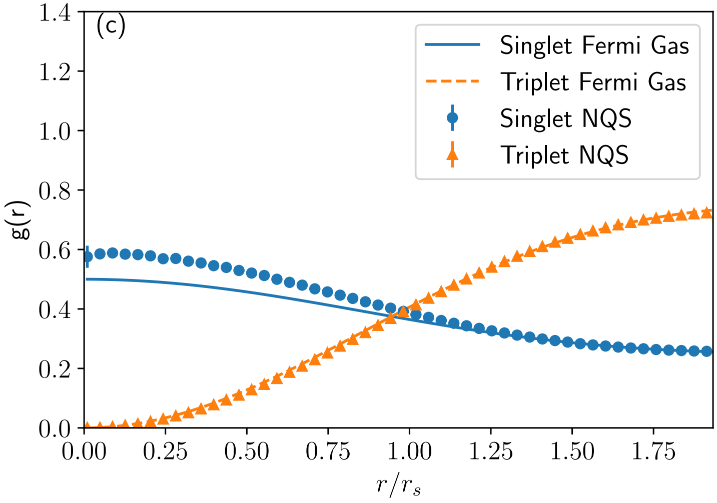
Pairing and Spin-singlet and triplet two-body distribution functions at $\rho = 0.01$ fm^{-3}



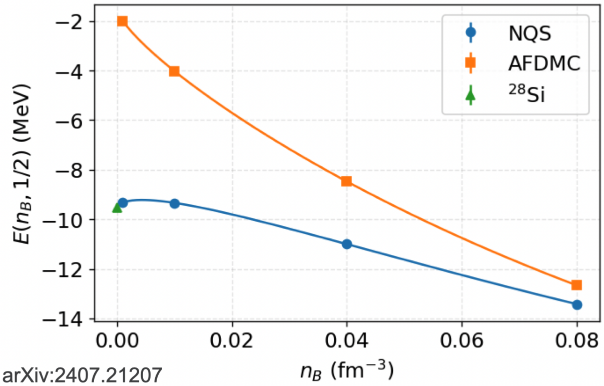
Pairing and Spin-singlet and triplet two-body distribution functions at $\rho = 0.04$ fm^{-3}



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

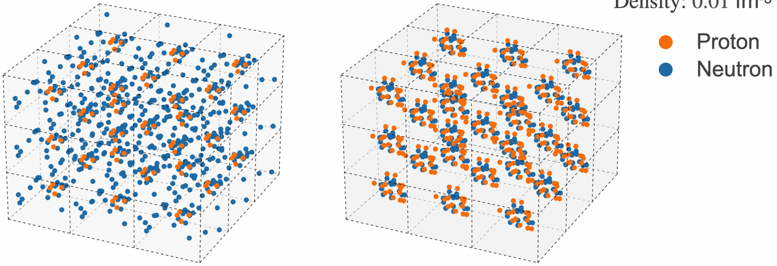


Symmetric nuclear matter



B. Fore, arXiv:2407.21207

Self-emerging clustering

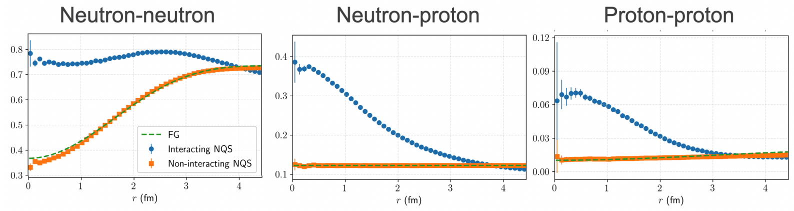


4 protons; 24 neutrons

14 protons; 14 neutrons

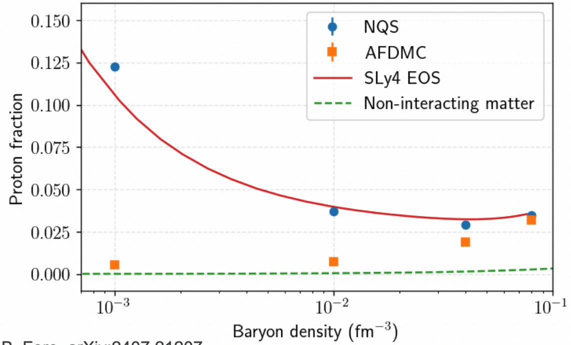
B. Fore, arXiv:2407.21207

Clustering: Two-body pair distributions



B. Fore, arXiv:2407.21207

Nuclear matter proton fraction

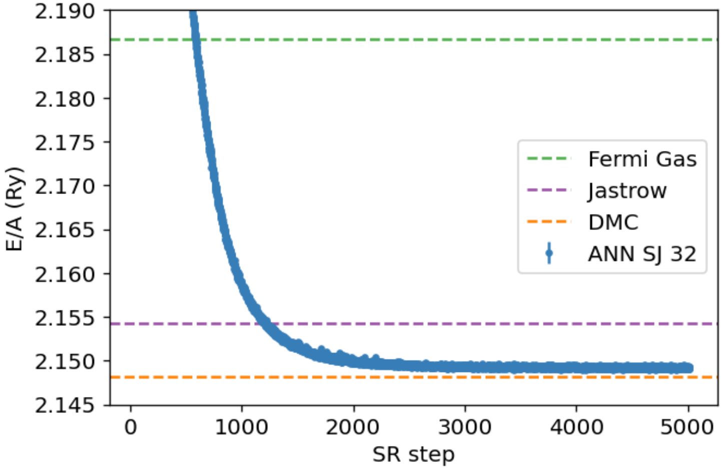


B. Fore, arXiv:2407.21207

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,



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 ($r_{ij} = r$)

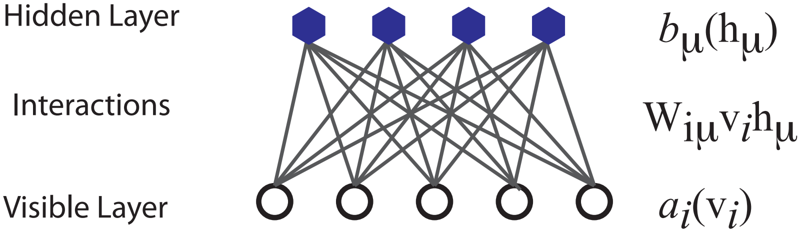
$$\hat{H}_r = -\nabla_r^2 + \frac{1}{4}\omega^2 r^2 + \frac{1}{r},$$

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

Generative models: Why Boltzmann machines?

What is known as restricted Boltzmann Machines (RMB) 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



The network

The network layers:

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

Joint distribution

The restricted Boltzmann machine is described by a Bolztmann distribution

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

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

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

Note the absence of the inverse temperature in these equations.

Network Elements, the energy function

The function $E(\boldsymbol{x}, \boldsymbol{h})$ 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 (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 $E(\boldsymbol{x}, \boldsymbol{h})$. 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:

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

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

Gaussian binary

Gaussian-Binary RBM:

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

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

Representing the wave function

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

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

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

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

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

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

Define the cost function

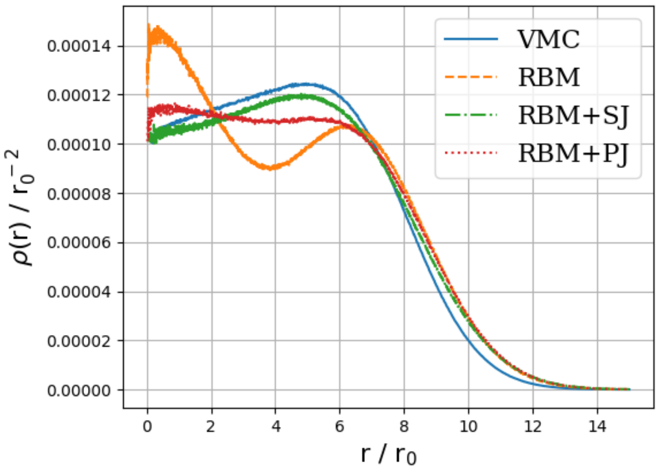
Now we don't necessarily have training data (unless we generate it by using some other method). However, what we do have is the variational principle which allows us to obtain the ground state wave function by minimizing the expectation value of the energy of a trial wavefunction (corresponding to the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

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

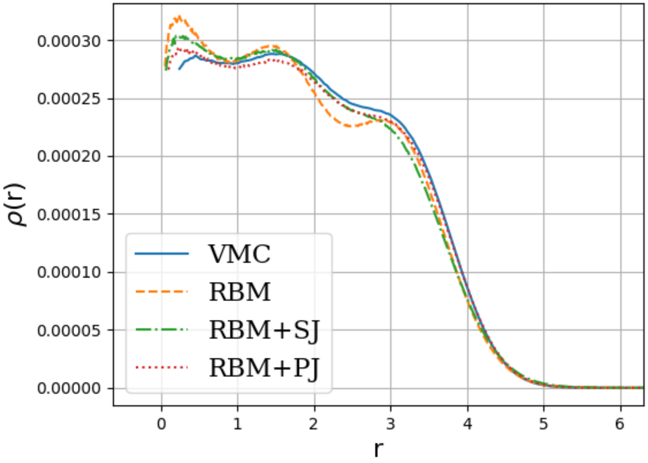
where the local energy is given by

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

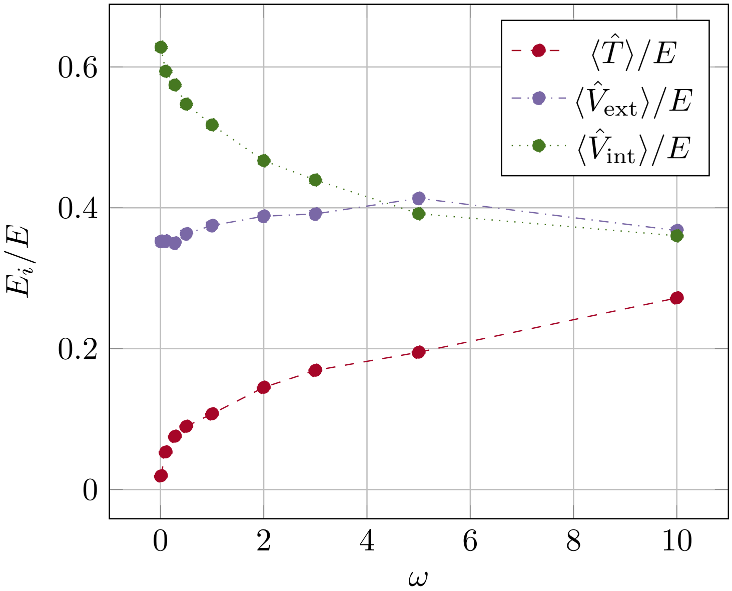
Quantum dots and Boltzmann machines, onebody densities $N = 6$, $\hbar\omega = 0.1$ a.u.



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 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, experimental design, analysis of experimental data and more?
  - Can we use AL/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

Our network example, simple percepetron with one input

As as simple example we define now a simple perceptron model with all quantities given by scalars. We consider only one input variable x and one target value y. We define an activation function σ_1 which takes as input

$$z_1 = w_1 x + b_1,$$

where w_1 is the weight and b_1 is the bias. These are the parameters we want to optimize. This output is then fed into the **cost/loss** function, which we here for the sake of simplicity just define as the squared error

$$C(x; w_1, b_1) = \frac{1}{2}(a_1 - y)^2.$$

Optimizing the parameters

In setting up the feed forward and back propagation parts of the algorithm, we need now the derivative of the various variables we want to train.

We need

$$\frac{\partial C}{\partial w_1} \text{ and } \frac{\partial C}{\partial b_1}.$$

Using the chain rule we find

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} = (a_1 - y)\sigma_1' x,$$

and

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial b_1} = (a_1 - y)\sigma_1',$$

which we later will just define as

$$\frac{\partial C}{\partial a_1}\frac{\partial a_1}{\partial z_1} = \delta_1.$$

Implementing the simple perceptron model

In the example code here we implement the above equations (with explict expressions for the derivatives) with just one input variable x and one output variable. The target value $y = 2x + 1$ is a simple linear function in x. Since this is a regression problem, we define the cost function to be proportional to the least squares error

$$C(y, w_1, b_1) = \frac{1}{2}(a_1 - y)^2,$$

with a_1 the output from the network.

```
# import necessary packages
import numpy as np
import matplotlib.pyplot as plt

def feed_forward(x):
    # weighted sum of inputs to the output layer
    z_1 = x*output_weights + output_bias
    # Output from output node (one node only)
    # Here the output is equal to the input
    a_1 = z_1
    return a_1

def backpropagation(x, y):
    a_1 = feed_forward(x)
    # derivative of cost function
    derivative_cost = a_1 - y
```

Central magic

Automatic differentiation

Essential elements of generative models

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

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

Energy models

We define a domain $\boldsymbol{X}$ of stochastic variables $\boldsymbol{X} = \{x_0, x_1, \dots, x_{n-1}\}$ with a pertinent probability distribution

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

where we have assumed that the random varaibles x_i are all independent and identically distributed (iid).
We will now assume that we can defined this function in terms of optimization parameters $\boldsymbol{\Theta}$, 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 $\boldsymbol{H} = \{h_0, h_1, \dots, h_{m-1}\}$.

Probability model

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

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

Further simplifications

We simplify further by rewriting it as

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

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

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

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.

The derivative of the partition function

The partition function, defined above as

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

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^2).

Explicit expression for the derivative

We can rewrite

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

which reads in more detail

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

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

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

Final expression

Taking the derivative gives us

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

which is the expectation value of $\log f$

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

that is

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

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

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

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.

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

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

and

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

and

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