Premio Nobel per la Fisica 2024

Morten Hjorth-Jensen[1]

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

UniTN, Dipartimento di Fisica, 9 dicembre 2024

Nobel per la Fisica sull'apprendimento automatico

Quali furono le ragioni e motivazioni per dare il premio Nobel per la Fisica a Hopfield e Hinton?

Tutto il materiale per questo seminario lo potete trovare sul sito
http://mhjensenseminars.github.io/MachineLearningTalk/doc/pub/UniTNDicembre2024/pdf/UniTNDicembre2024.pdf

Premio Nobel 2024

Tratto dal sito per il premio Nobel:

This year's two Nobel Laureates in Physics have used tools from physics to develop methods that are the foundation of today's powerful machine learning. John Hopfield created an associative memory that can store and reconstruct images and other types of patterns in data. Geoffrey Hinton invented a method that can autonomously find properties in data, and so perform tasks such as identifying specific elements in pictures.

Geoffrey Hinton

Dal sito per il premio Nobel.

Geoffrey Hinton used the Hopfield network as the foundation for a new network that uses a different method: the Boltzmann machine. This can learn to recognise characteristic elements in a given type of data. Hinton used tools from statistical physics, the science of systems built from many similar components. The machine is trained by feeding it examples that are very likely to arise when the machine is run. The Boltzmann machine can be used to classify images or create new examples of the type of pattern on which it was trained. Hinton has built upon this work, helping initiate the current explosive development of machine learning.

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

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

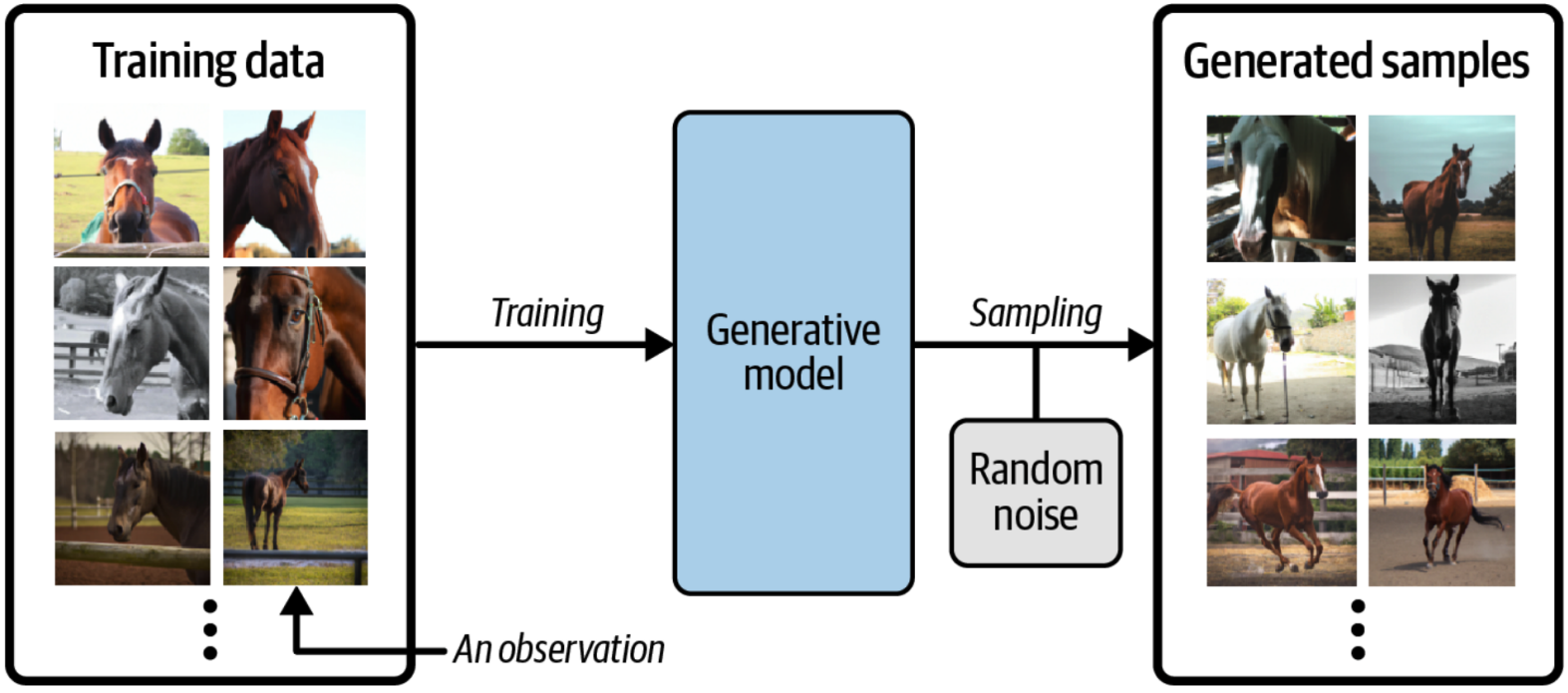
Another way to categorize machine learning tasks is to consider the desired output of a system. Some of the most common tasks are:

- 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.

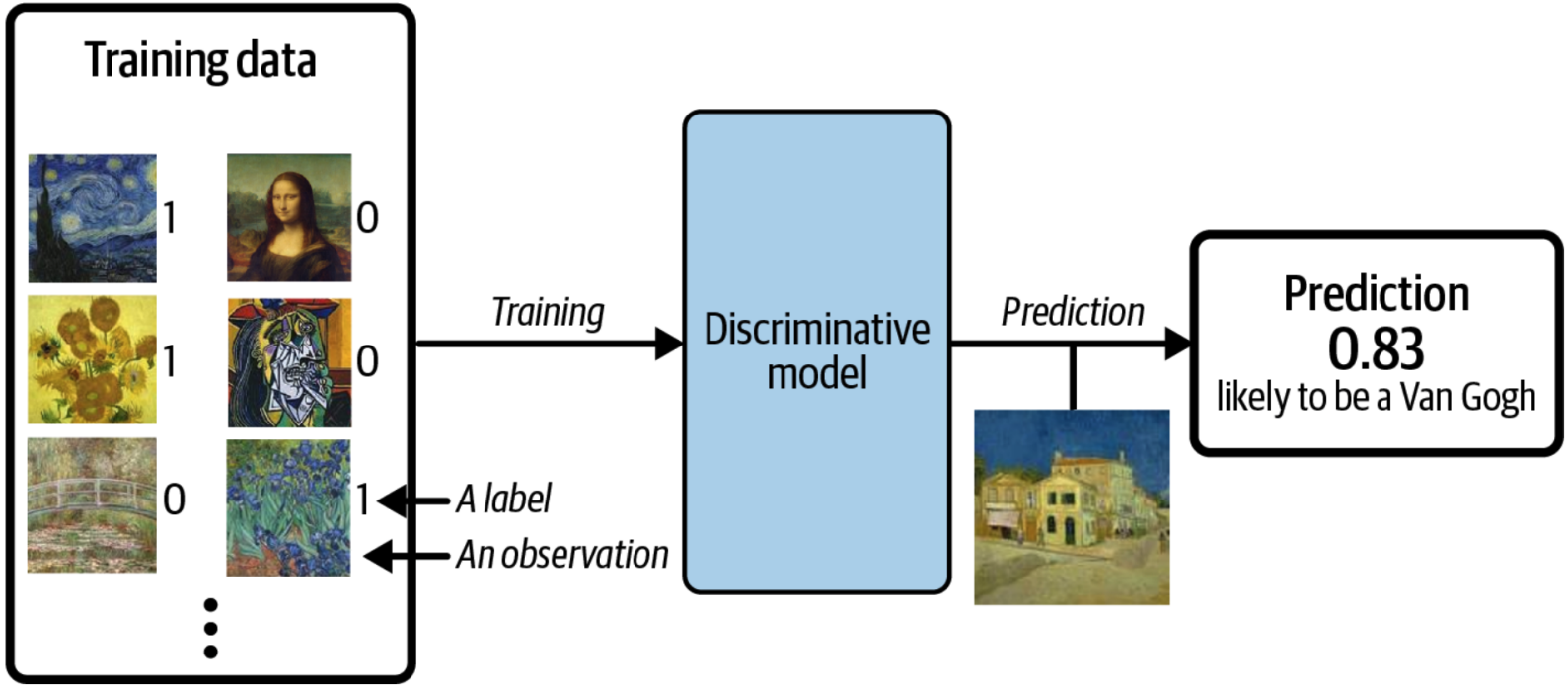
The plethora of machine learning algorithms/methods

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

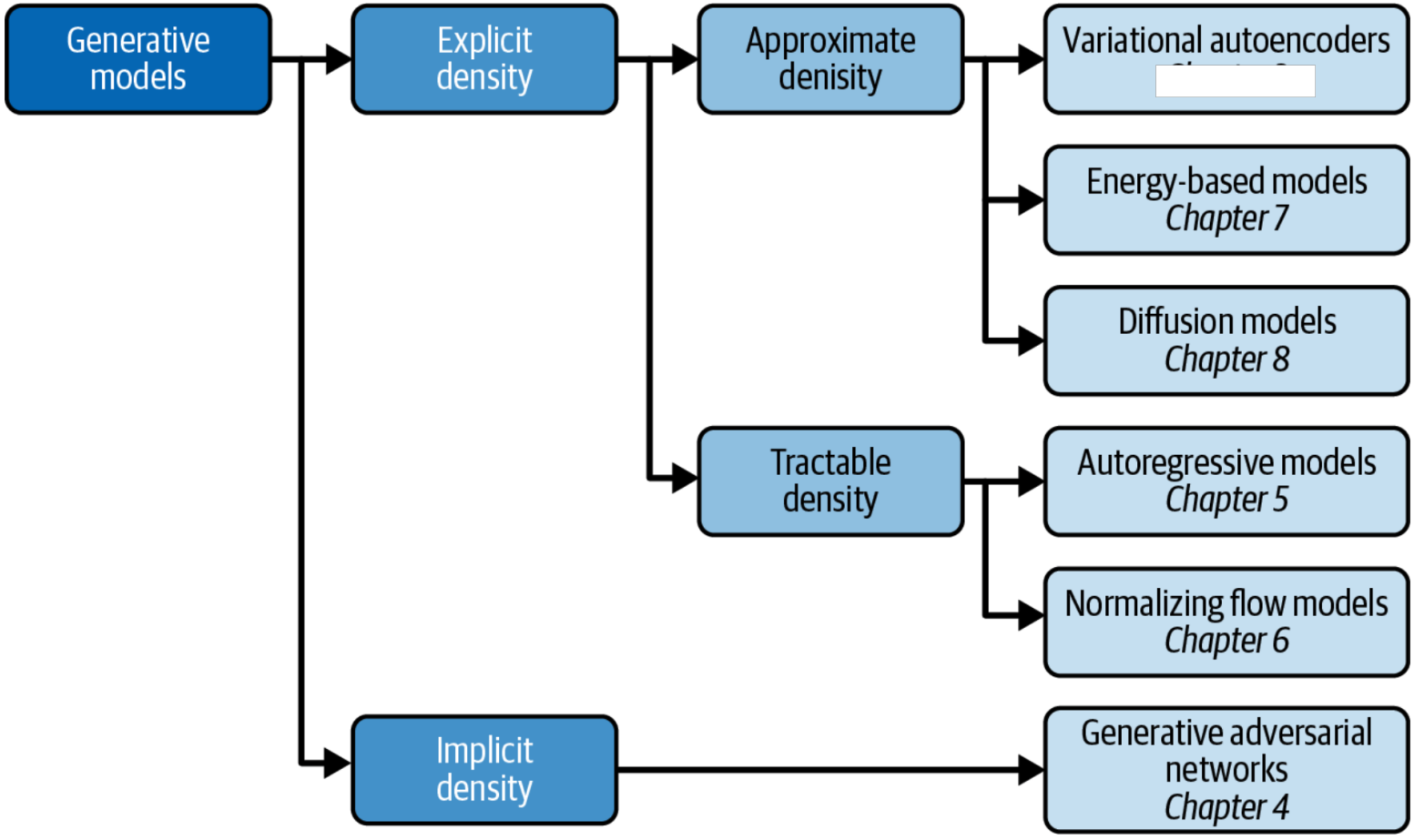
Example of generative modeling, taken from Generative Deep Learning by David Foster



Example of discriminative modeling, taken from Generative Deeep 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
- Bali and Gavras, Generative AI with Python and TensorFlow 2

All three books have GitHub addresses from where one can download all codes. We will borrow most of the material from these three texts as well as from Goodfellow, Bengio and Courville's text Deep Learning

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

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

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 $\boldsymbol{\theta}^T = [\theta_0, \theta_1, \theta_2, \ldots, \theta_{p-1}]$.
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

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

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

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

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.

Lots of room for creativity

Not all the algorithms and methods can be given a rigorous mathematical justification, opening up thereby for experimenting and trial and error and thereby exciting new developments.

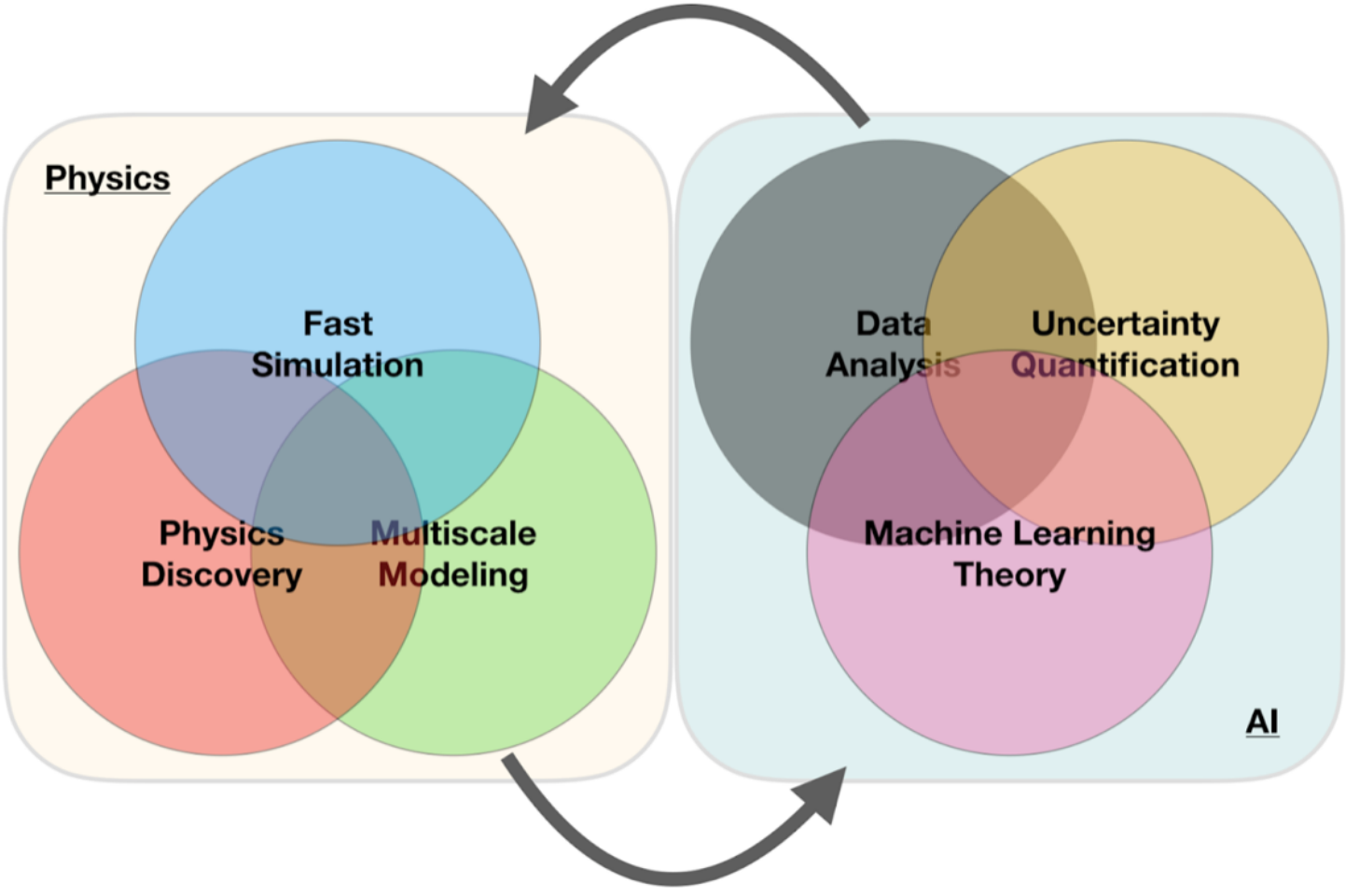
A solid command of linear algebra, multivariate theory, probability theory, statistical data analysis, optimization algorithms, understanding errors and Monte Carlo methods is important in order to understand many of the various algorithms and methods.

Job market, a personal statement: A familiarity with ML is almost becoming a prerequisite for many of the most exciting employment opportunities. And add quantum computing and there you are!

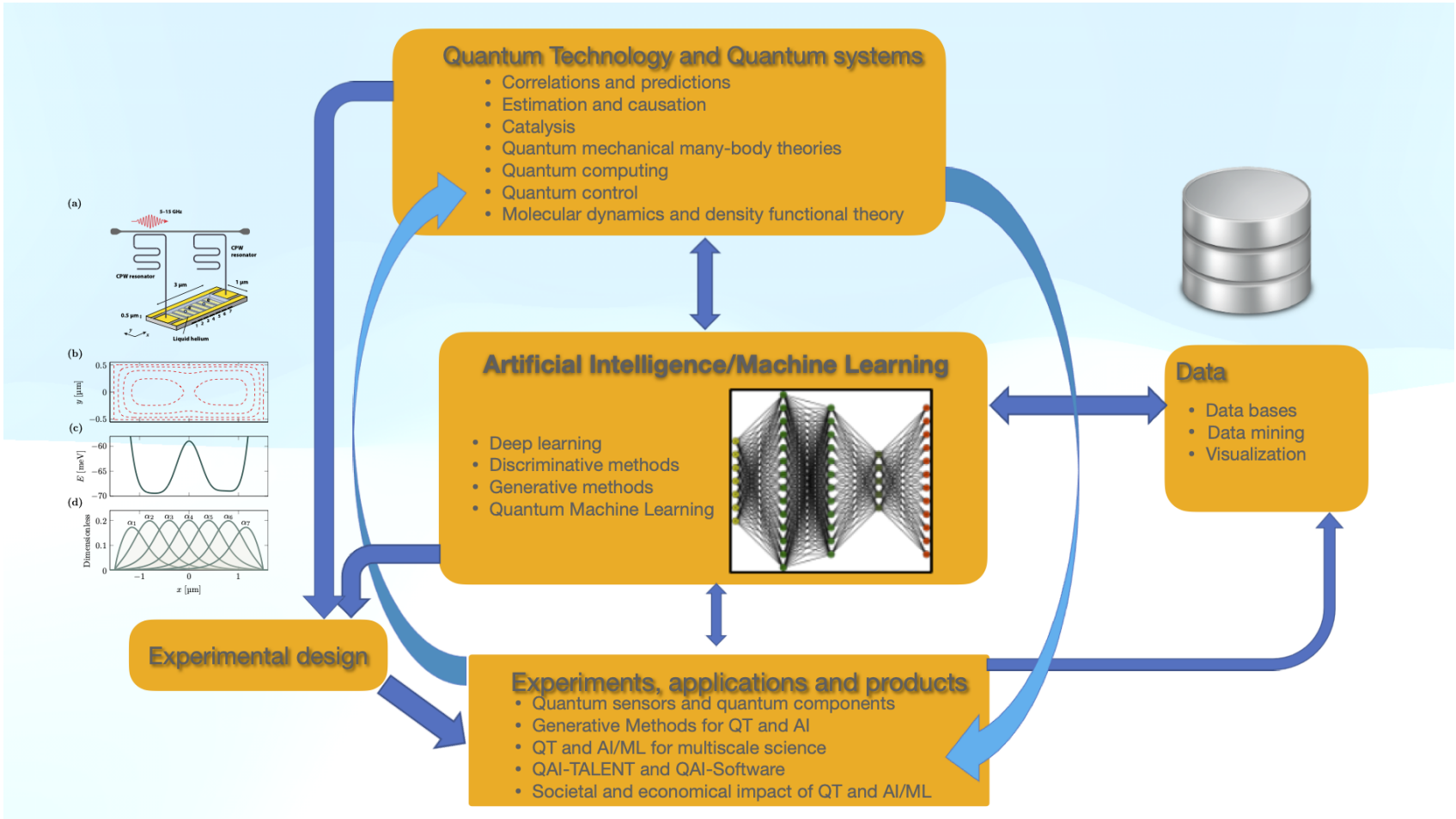
Selected references

- 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, Nature Physics Communcication, submitted
- Message-Passing Neural Quantum States for the Homogeneous Electron Gas, 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
- Particle Data Group summary on ML methods

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



AI/ML and Quantum Computing



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.

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

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

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

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

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.

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:

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

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

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

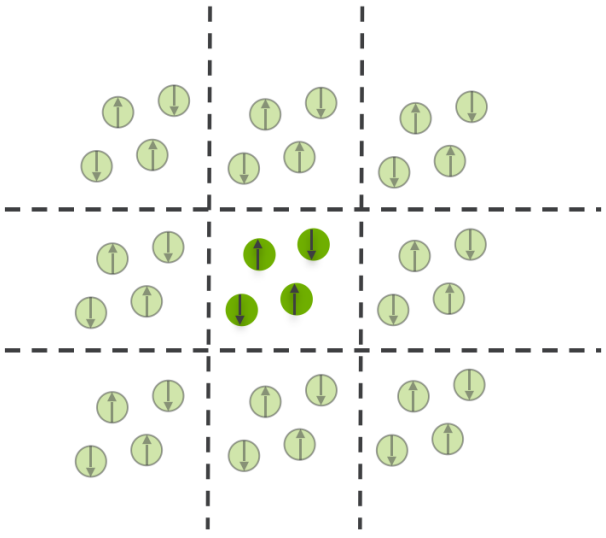
which is commonly used for nuclear Hamiltonians that do not contain tensor and spin-orbit terms. The above function is replaced by a four-layer Neural Network.

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

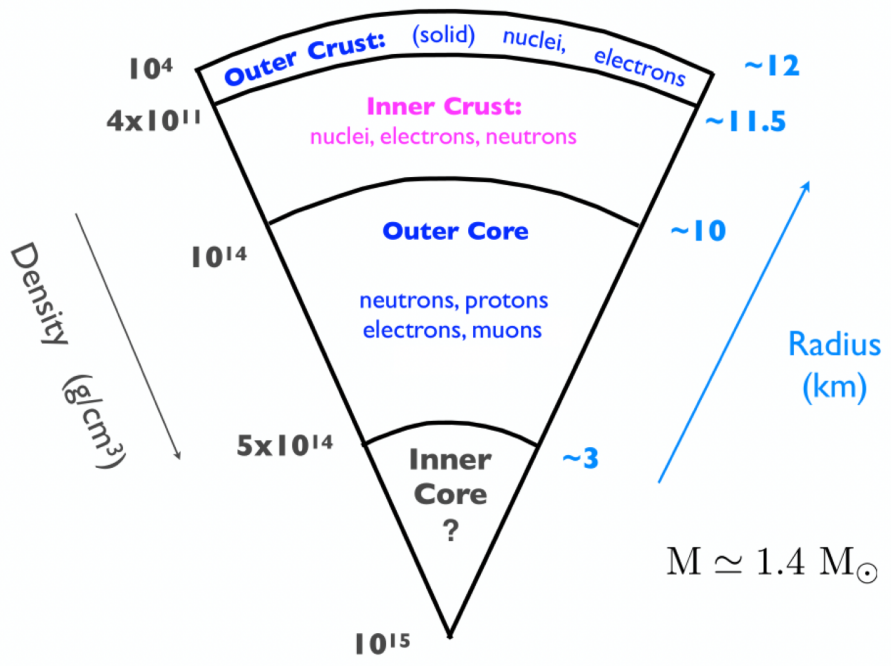


- Periodic boundary conditions and coordinate system

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

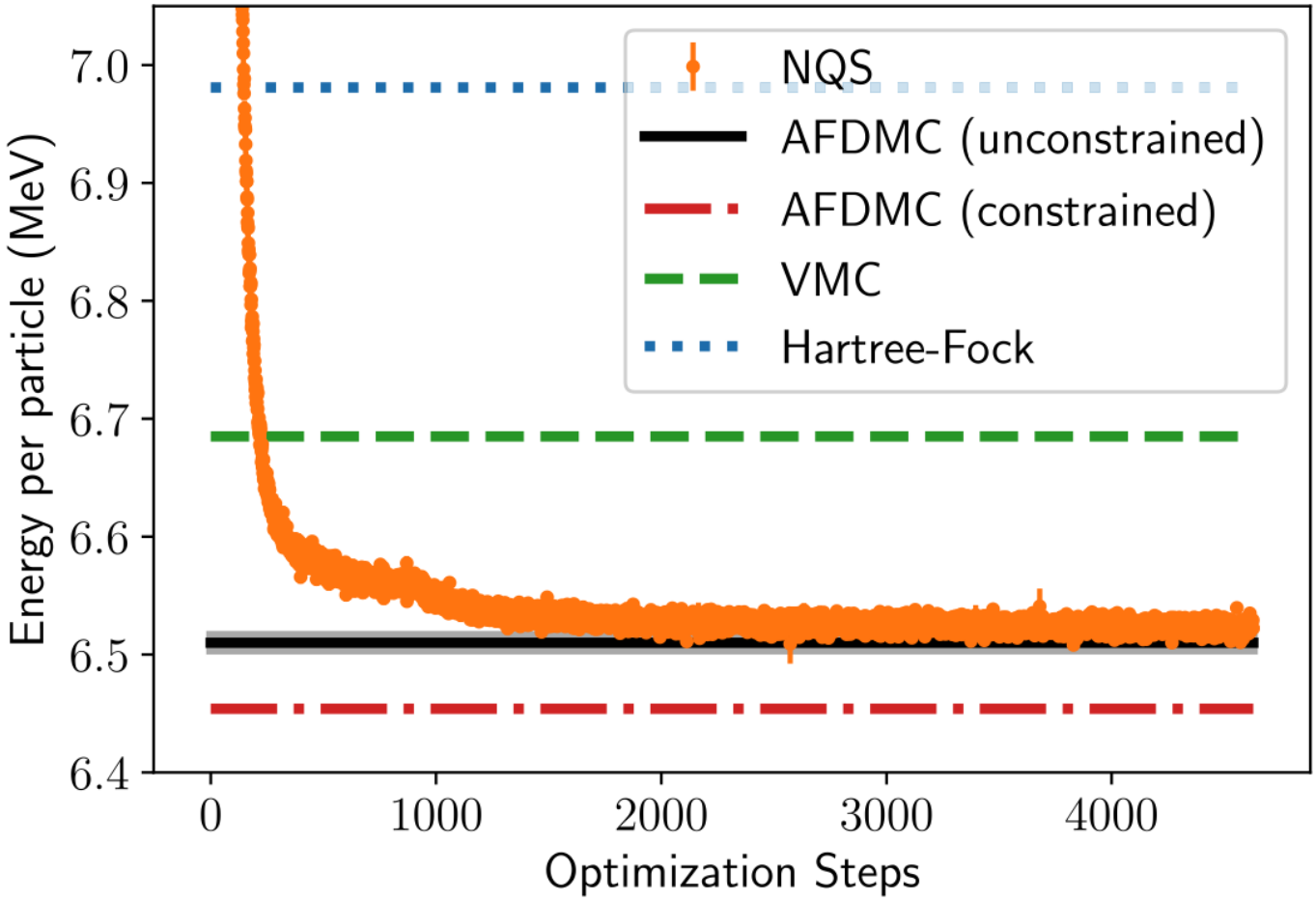
- Potential energy contribution from particle images
- Remove Coulomb potential

Neutron star structure

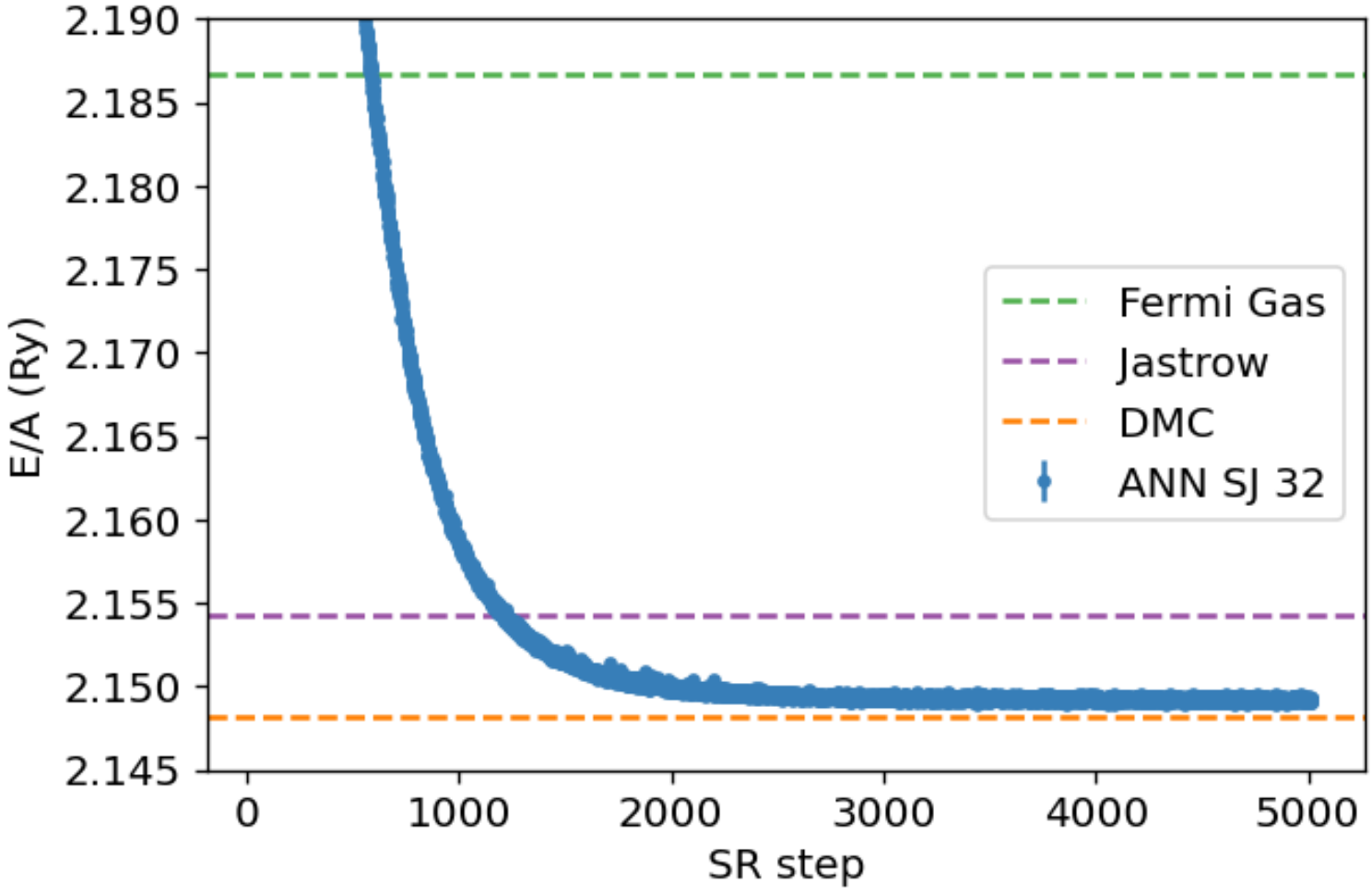


- Mostly neutrons but composition varies with density
- Nuclei in crust are squeezed into uniform matter in core
- Likely neutron superfluid in inner crust and outer core
- Calculations currently focus on inner crust

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



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.

Physics based statistical learning and data analysis

The above concepts are in some sense the difference between **old-fashioned** machine learning and statistics and Bayesian learning. In machine learning and prediction based tasks, we are often interested in developing algorithms that are capable of learning patterns from given data in an automated fashion, and then using these learned patterns to make predictions or assessments of newly given data. In many cases, our primary concern is the quality of the predictions or assessments, and we are less concerned about the underlying patterns that were learned in order to make these predictions.

Physics based statistical learning points however to approaches that give us both predictions and correlations as well as being able to produce error estimates and understand causations. This leads us to the very interesting field of Bayesian statistics.

Bayes' Theorem

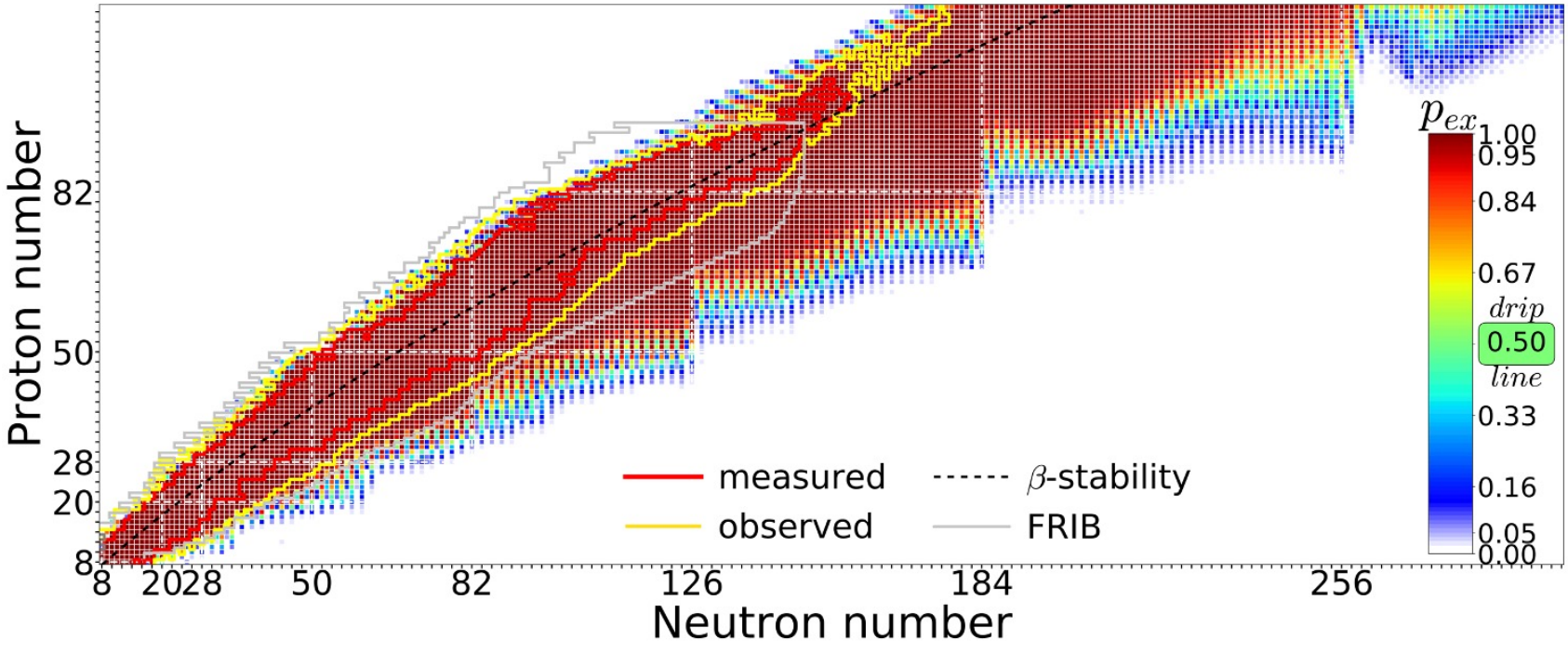
Bayes' theorem

$$p(X|Y) = \frac{p(X,Y)}{\sum_{i=0}^{n-1} p(Y|X=x_i)p(x_i)} = \frac{p(Y|X)p(X)}{\sum_{i=0}^{n-1} p(Y|X=x_i)p(x_i)}.$$

The quantity $p(Y|X)$ on the right-hand side of the theorem is evaluated for the observed data Y and can be viewed as a function of the parameter space represented by X. This function is not necessarily normalized and is normally called the likelihood function. The function $p(X)$ on the right hand side is called the prior while the function on the left hand side is the called the posterior probability. The denominator on the right hand side serves as a normalization factor for the posterior distribution.

Quantified limits of the nuclear landscape

Predictions made with eleven global mass model and Bayesian model averaging



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, accelerator 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 your 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

Possible questions for discussions

1. How do we incorporate these topics in our education?
2. More difficult: what are the consequences for universities and our educational mission?

Education

1. Incorporate elements of statistical data analysis and Machine Learning in undergraduate programs
2. Develop courses on Machine Learning and statistical data analysis
3. Build up a series of courses in Quantum Information Technologies (QIT)
4. Modifying contents of present Physics programs or new programs on Computational Physics and Quantum Technologies
   - 4.1 study direction/option in **quantum technologies**
   - 4.2 study direction/option in **Artificial Intelligence and Machine Learning**
   - 4.3 and more
5. Master of Science/PhD programs in Computational and Data Science
   - 5.1 UiO has already MSc programs in CS and DS
   - 5.2 MSU has own graduate programs plus dual degree programs in CS and DS
   - 5.3 Many other universities are developing or have similar programs

Possible courses quantum courses

Topics in a Bachelor of Science/Master of Science

1. General university course on quantum mech and quantum technologies
2. Information Systems
3. From Classical Information theory to Quantum Information theory
4. Classical vs. Quantum Logic
5. Classical and Quantum Laboratory
6. Discipline-Based Quantum Mechanics
7. Quantum Software
8. Quantum Hardware
9. more

Important Issues to think of

1. Lots of conceptual learning: superposition, entanglement, QIT applications, etc.
2. Coding is indispensable.
3. Teamwork, project management, and communication are important and highly valued
4. Engagement with industry: guest lectures, virtual tours, co-ops, and/or internships.

Observations

1. Students do not really know what QIT is.
2. ML/AI seen as black boxes/magic!
3. Students perceive that a graduate degree is necessary to work in QIS. A BSc will help.

Future Needs/Problems

1. There are already great needs for specialized people (Ph. D. s, postdocs), but also needs of people with a broad overview of what is possible in ML/AI and/or QIT.
2. There are not enough potential employees in AI/ML and QIT . It is a supply gap, not a skills gap.
3. A BSc with specialization is a good place to start
4. It is tremendously important to get everyone speaking the same language. Facility with the vernacular of quantum mechanics is a big plus.
5. There is a huge list of areas where technical expertise may be important. But employers are often more concerned with attributes like project management, working well in a team, interest in the field, and adaptability than in specific technical skills.

Illustration of a single perceptron model and an FFNN

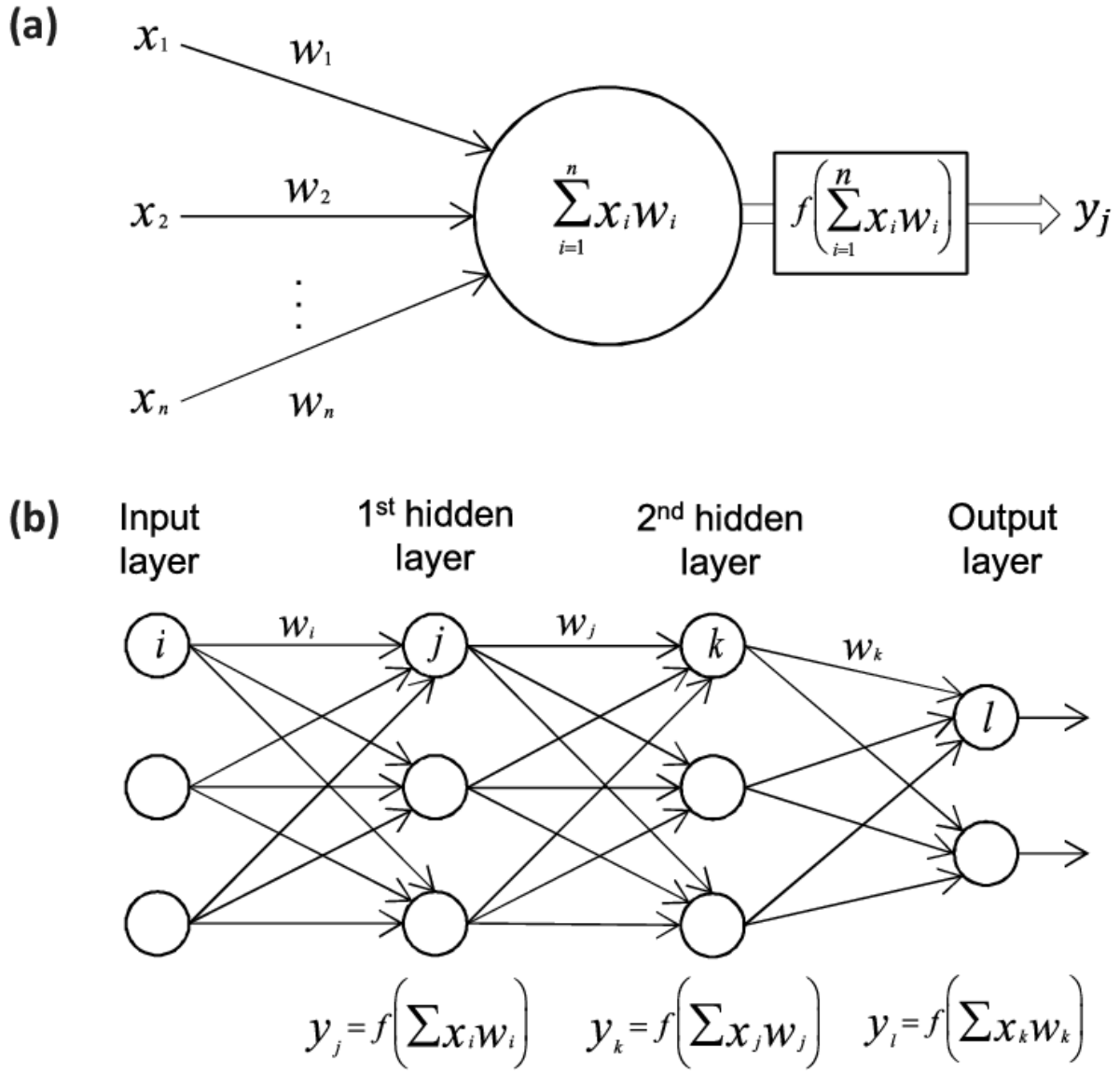


Figure: In a) we show a single perceptron model while in b) we dispay a network with two hidden layers, an input layer and an output layer.

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 differentian

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?

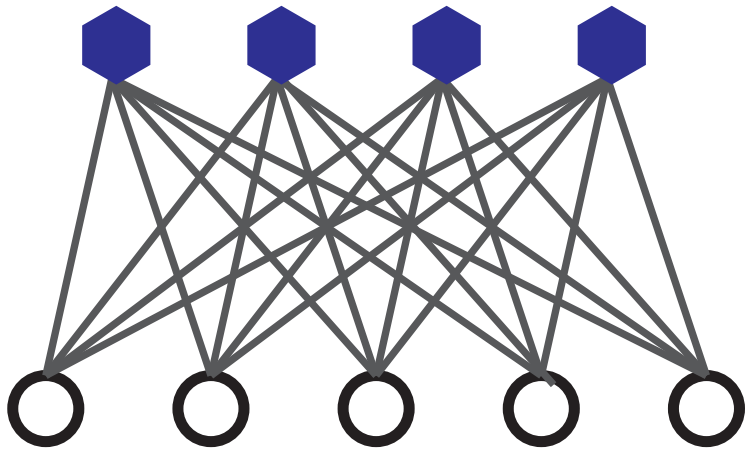
What is known as restricted Boltzmann Machines (RMB) have received a lot of attention lately. One of the major reasons is that they can be stacked layer-wise to build deep neural networks that capture complicated statistics.
The original RBMs had just one visible layer and a hidden layer, but recently 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

Hidden Layer

Interactions

Visible Layer



$b_\mu(h_\mu)$

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

$a_i(v_i)$

The network

The network layers:

1. A function $\boldsymbol{x}$ 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

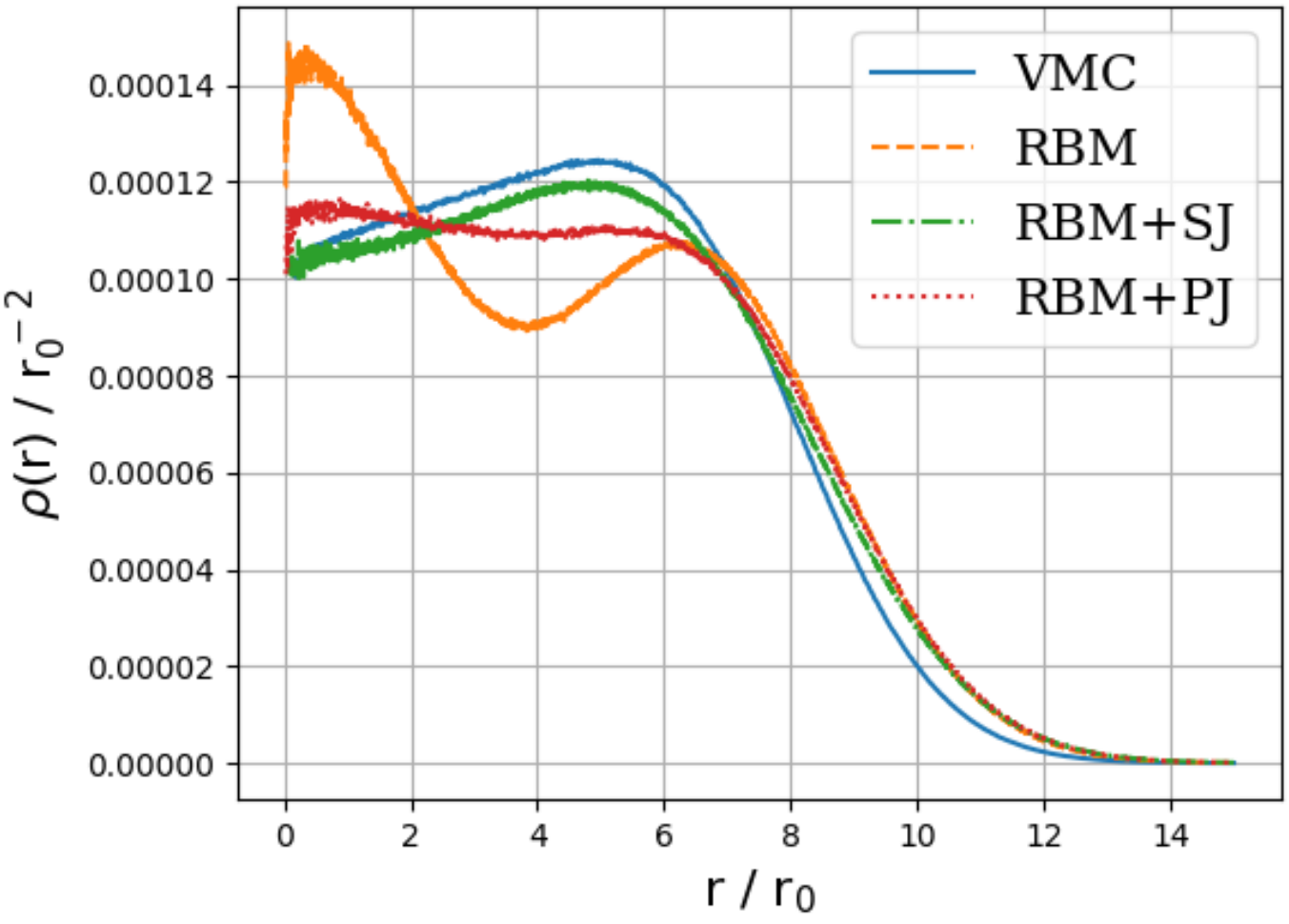
$$C_i = \frac{\partial \langle E_L \rangle}{\partial \theta_i} = 2(\langle E_L \frac{1}{\Psi} \frac{\partial \Psi}{\partial \theta_i} \rangle - \langle E_L \rangle \langle \frac{1}{\Psi} \frac{\partial \Psi}{\partial \theta_i} \rangle),$$

where the local energy is given by

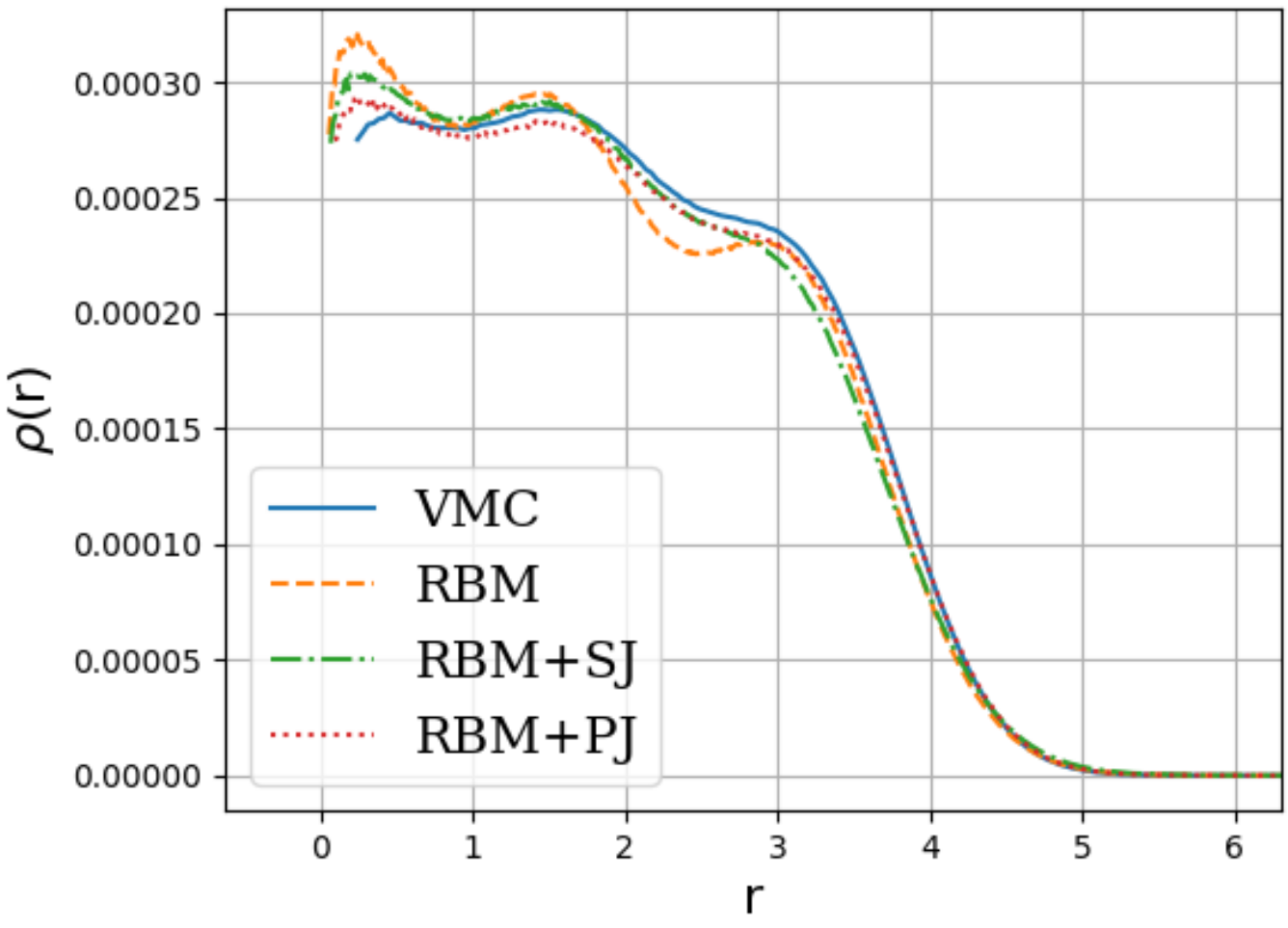
$$E_L = \frac{1}{\Psi} \hat{\boldsymbol{H}} \Psi.$$

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