

January 15-19: Advanced machine learning and data analysis for the physical sciences

Morten Hjorth-Jensen^{1,2}

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

Department of Physics and Astronomy and Facility for Rare Isotope Beams,
Michigan State University, East Lansing, Michigan, USA²

FYS5429/9429, Spring 2024

© 1999-2024, Morten Hjorth-Jensen. Released under CC Attribution-NonCommercial 4.0 license

Overview of first week, January 15-21, 2024

1. Presentation of course with general overview of methods
2. Discussion of possible projects and presentation of participants
3. Deep learning methods, review of neural networks

Practicalities and possible projects

1. Although the course is defined as a self-study course, we will have weekly lectures with small weekly exercise assignments
2. We plan to work on two projects which will define the content of the course, the format can be agreed upon by the participants but the following topics could define an outline for possible projects and machine learning topics
 - ▶ Deep learning with the aim to develop a code for CNNs and/or RNNs and study data of relevance for own research ([Higgs challenge for example](#))
 - ▶ Study autoencoders and variational autoencoders with application to own data
 - ▶ GANs and applications to own data
 - ▶ Solve quantum/or classical many-body problems with deep learning methods (overlaps with FYS4411)
 - ▶ Physics informed Machine Learning, applications to for example solution of Navier-Stokes equations
 - ▶ Bayesian Machine Learning and Gaussian processes
 - ▶ and many other research paths and topics
3. No exam, only two projects. Each projects counts 1/2 of the final grade.
4. All info at the GitHub address [http://github.com/](#)

Deep learning methods covered, tentative

1. Deep learning, classics

- 1.1 Feed forward neural networks and its mathematics (NNs)
- 1.2 Convolutional neural networks (CNNs)
- 1.3 Recurrent neural networks (RNNs)
- 1.4 Autoencoders and principal component analysis
- 1.5 Physical Sciences informed machine learning

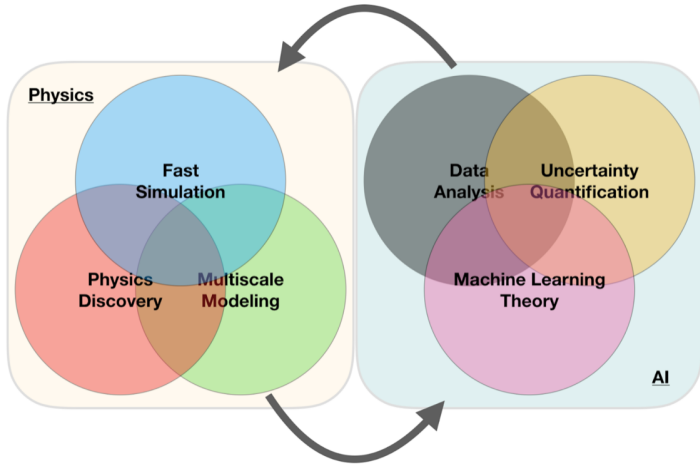
2. Deep learning, generative methods

- 2.1 Basics of generative models
- 2.2 Boltzmann machines and energy based methods
- 2.3 Diffusion models (tentative)
- 2.4 Variational autoencoders (VAEs)
- 2.5 Generative Adversarial Networks (GANs)
- 2.6 Autoregressive methods (tentative)

Additionally Kernel regression (Gaussian processes) and Bayesian statistics

Kernel machine regression (KMR), also called Gaussian process regression, is a popular tool in the machine learning literature. The main idea behind KMR is to flexibly model the relationship between a large number of variables and a particular outcome (dependent variable).

Machine learning. A simple perspective on the interface between ML and Physics, HPC is essential



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](#). See also <https://www.nationaldefensemagazine.org/articles/2023/3/24/ukraine-a-living-lab-for-ai-warfare>

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 emerging 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 (NNs), Convolutional NNs, Recurrent NNs, Transformers, Boltzmann machines, autoencoders and variational autoencoders and generative adversarial networks and other 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

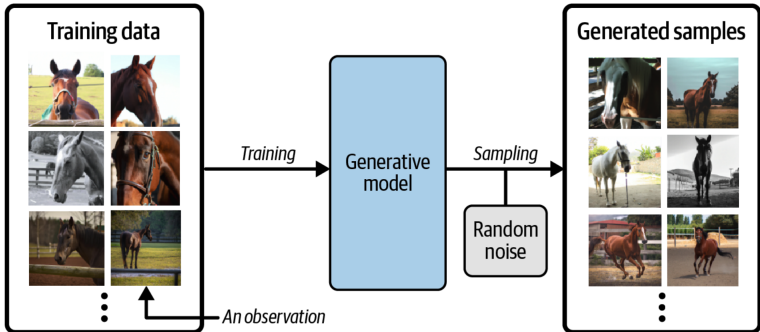
What Is Generative Modeling?

Generative modeling can be broadly defined as follows:

Generative modeling is a branch of machine learning that involves training a model to produce new data that is similar to a given dataset.

What does this mean in practice? Suppose we have a dataset containing photos of horses. We can train a generative model on this dataset to capture the rules that govern the complex relationships between pixels in images of horses. Then we can sample from this model to create novel, realistic images of horses that did not exist in the original dataset.

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



Generative Modeling

In order to build a generative model, we require a dataset consisting of many examples of the entity we are trying to generate. This is known as the training data, and one such data point is called an observation.

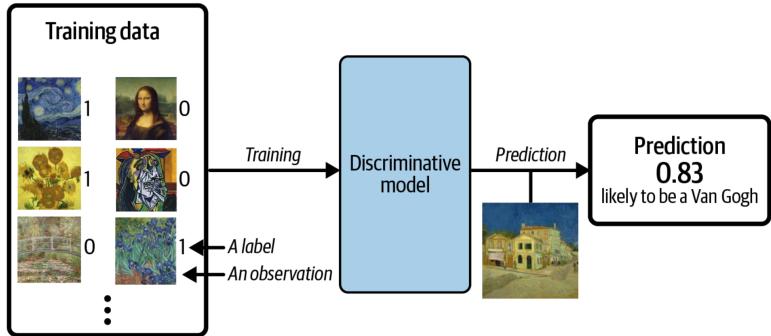
Each observation consists of many features. For an image generation problem, the features are usually the individual pixel values; for a text generation problem, the features could be individual words or groups of letters. It is our goal to build a model that can generate new sets of features that look as if they have been created using the same rules as the original data.

Conceptually, for image generation this is an incredibly difficult task, considering the vast number of ways that individual pixel values can be assigned and the relatively tiny number of such arrangements that constitute an image of the entity we are trying to generate.

Generative Versus Discriminative Modeling

In order to truly understand what generative modeling aims to achieve and why this is important, it is useful to compare it to its counterpart, discriminative modeling. If you have studied machine learning, most problems you will have faced will have most likely been discriminative in nature.

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

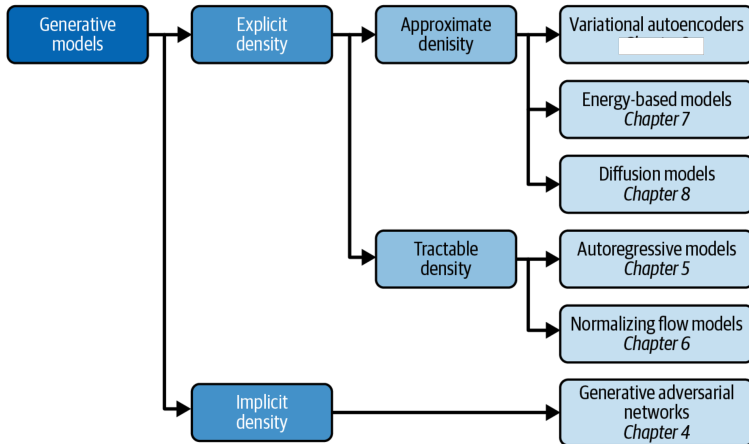


Discriminative Modeling

When performing discriminative modeling, each observation in the training data has a label. For a binary classification problem such as our data could be labeled as ones and zeros. Our model then learns how to discriminate between these two groups and outputs the probability that a new observation has label 1 or 0

In contrast, generative modeling doesn't require the dataset to be labeled because it concerns itself with generating entirely new data (for example an image), rather than trying to predict a label for say a given image.

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



What are the basic ingredients?

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

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

We seek to minimize the function $\mathcal{C}(x, f(\alpha))$ 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 algorithms 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 $\mathbf{x}^T = [x_0, x_1, x_2, \dots, x_{n-1}]$ and the outputs we want to model $\mathbf{y}^T = [y_0, y_1, y_2, \dots, y_{n-1}]$. We assume that the output data can be represented (for a regression case) by a continuous function f through

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

Setting up the equations

In linear regression we approximate the unknown function with another continuous function $\tilde{\mathbf{y}}(\mathbf{x})$ 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 $\tilde{\mathbf{y}}$

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

The objective/cost/loss function

The simplest approach is the mean squared error

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

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

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

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

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

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

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

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

Ridge and LASSO Regression

Our optimization problem is

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n} \left\{ (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \right\}.$$

or we can state it as

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = \frac{1}{n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2,$$

where we have used the definition of a norm-2 vector, that is

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}.$$

From OLS to Ridge and Lasso

By minimizing the above equation with respect to the parameters θ we could then obtain an analytical expression for the parameters θ . We can add a regularization parameter λ by defining a new cost function to be optimized, that is

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\theta\|_2^2 + \lambda \|\theta\|_2^2$$

which leads to the Ridge regression minimization problem where we require that $\|\theta\|_2^2 \leq t$, where t is a finite number larger than zero. We do not include such a constraints in the discussions here.

Lasso regression

Defining

$$C(\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1,$$

we have a new optimization equation

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1$$

which leads to Lasso regression. Lasso stands for least absolute shrinkage and selection operator. Here we have defined the norm-1 as

$$\|\mathbf{x}\|_1 = \sum_i |x_i|.$$

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 which one download all codes.

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)

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.

Physics driven Machine Learning

Another hot topic is what has loosely been dubbed **Physics-driven deep learning**. See the recent work on [Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators](#), Nature Machine Learning, vol 3, 218 (2021).

From their abstract

A less known but powerful result is that an NN with a single hidden layer can accurately approximate any nonlinear continuous operator. This universal approximation theorem of operators is suggestive of the structure and potential of deep neural networks (DNNs) in learning continuous operators or complex systems from streams of scattered data. ... We demonstrate that DeepONet can learn various explicit operators, such as integrals and fractional Laplacians, as well as implicit operators that represent deterministic and stochastic differential equations.

Many-body physics, Quantum Monte Carlo and deep learning

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

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

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

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

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

Quantum Monte Carlo Motivation

Basic steps

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

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

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

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

Quantum Monte Carlo Motivation

Define a new quantity

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

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

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

with N being the number of Monte Carlo samples.

Energy derivatives

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

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

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

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

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

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

Derivatives of the local energy

The elements of the gradient of the local energy are

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

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

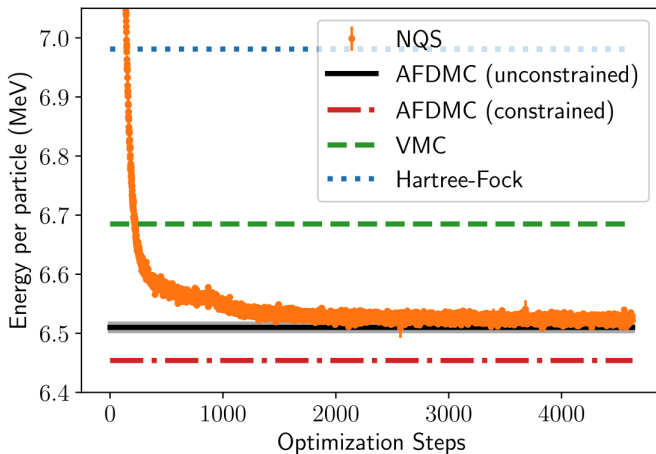
$$\left\langle \frac{\bar{\Psi}_i}{\Psi} \right\rangle \langle E_L \rangle$$

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.

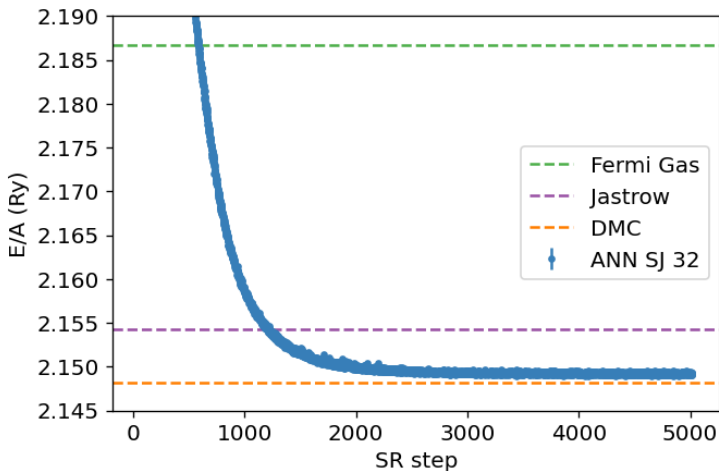
Why Feed Forward Neural Networks (FFNN)?

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**.

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



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,



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.

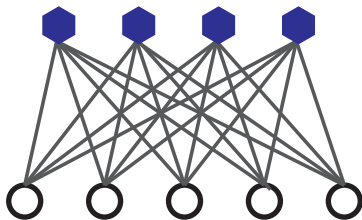
Furthermore, they have been used to solve complicated quantum mechanical many-particle problems or classical statistical physics problems like the Ising and Potts classes of models.

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 x that represents the visible layer, a vector of M elements (nodes). This layer represents both what the RBM might be given as training input, and what we want it to be able to reconstruct. This might for example be the pixels of an image, the spin values of the Ising model, or coefficients representing speech.
2. The function h represents the hidden, or latent, layer. A vector of N elements (nodes). Also called "feature detectors".

Goals

The goal of the hidden layer is to increase the model's expressive power. We encode complex interactions between visible variables by introducing additional, hidden variables that interact with visible degrees of freedom in a simple manner, yet still reproduce the complex correlations between visible degrees in the data once marginalized over (integrated out).

The network parameters, to be optimized/learned:

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

Joint distribution

The restricted Boltzmann machine is described by a Boltzmann distribution

$$P_{rbm}(x, h) = \frac{1}{Z} e^{-\frac{1}{T_0} E(x, h)},$$

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

$$Z = \int \int e^{-\frac{1}{T_0} E(x, h)} dx dh.$$

It is common to ignore T_0 by setting it to one.

Network Elements, the energy function

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

$$F_{rbm}(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-\frac{1}{T_0} E(\mathbf{x}, \mathbf{h})}.$$

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

$$F_{rbm}(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})}.$$

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

$$\Psi(\mathbf{X}) = F_{rbm}(\mathbf{x}),$$

or we could square the wave function.

Choose 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

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

Extrapolations and model interpretability

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

