Machine Learning and Boltzmann machines

Morten Hjorth-Jensen

of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University and Department of Physics, University of C

Nov 20, 2018

Types of Machine Learning, a repetition

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. Some authours also operate with a third category, namely *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.

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.
- Other unsupervised learning algortihms, here Boltzmann machines

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.

Boltzmann Machines

Why use a generative model rather than the more well known discriminative deep neural networks (DNN)?

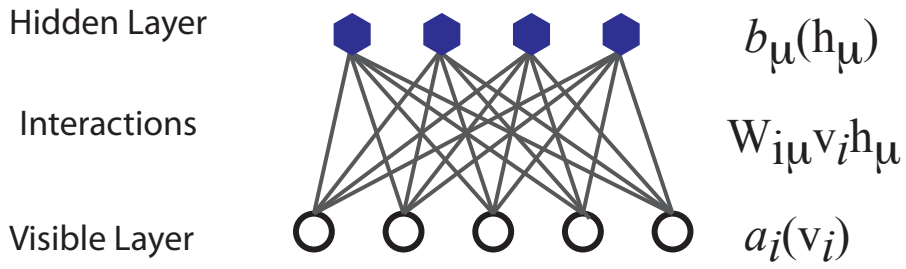
- Discriminitave methods have several limitations: They are mainly supervised learning methods, thus requiring labeled data. And there are tasks they cannot accomplish, like drawing new examples from an unknown probability distribution.
- A generative model can learn to represent and sample from a probability distribution. The core idea is to learn a parametric model of the probability distribution from which the training data was drawn. As an example
    1. A model for images could learn to draw new examples of cats and dogs, given a training dataset of images of cats and dogs.
    2. Generate a sample of an ordered or disordered Ising model phase, having been given samples of such phases.
    3. Model the trial wave function for Monte Carlo calculations

Some similarities and differences from DNNs

1. Both use gradient-descent based learning procedures for minimizing cost functions
2. Energy based models don't use backpropagation and automatic differentiation for computing gradients, instead turning to Markov Chain Monte Carlo methods.
3. DNNs often have several hidden layers. A restricted Boltzmann machine has only one hidden layer, however several RBMs can be stacked to make up Deep Belief Networks, of which they constitute the building blocks.

History: The RBM was developed by amongst others Geoffrey Hinton, called by some the "Godfather of Deep Learning", working with the University of Toronto and Google.

The structure of the RBM network



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).

Examples of this trick being employed in physics:

1. The Hubbard-Stratonovich transformation
2. The introduction of ghost fields in gauge theory
3. Shadow wave functions in Quantum Monte Carlo simulations

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 lenght as **h**.
3. W represents the interaction weights, a matrix of size $M \times N$.

Joint distribution and the Energy function

The restricted Boltzmann machine is described by a Bolztmann distribution

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

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

$$Z = \int \int e^{-\frac{1}{T_0} E(\mathbf{x}, \mathbf{h})} d\mathbf{x} d\mathbf{h}. \tag{2}$$

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

Network Elements

The function $E(\mathbf{x}, \mathbf{h})$ gives the **energy** of a configuration (pair of vectors) $(\mathbf{x}, \mathbf{h})$. The lower the energy of a configuration, the higher the probability of it. This function also depends on the parameters $\mathbf{a}$, $\mathbf{b}$ and W. Thus, when we adjust them during the learning procedure, we are adjusting the energy function to best fit our problem.

Defining different types of RBMs

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 $E(\mathbf{x}, \mathbf{h})$.

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:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_{i}^{M} x_i a_i - \sum_{j}^{N} b_j h_j - \sum_{i,j}^{M,N} x_i w_{ij} h_j, \tag{3}$$

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

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

$$E(\mathbf{x}, \mathbf{h}) = \sum_{i}^{M} \frac{(x_i - a_i)^2}{2\sigma_i^2} - \sum_{j}^{N} b_j h_j - \sum_{i,j}^{M,N} \frac{x_i w_{ij} h_j}{\sigma_i^2}. \tag{4}$$

More about RBMs

1. Useful when we model continuous data (i.e., we wish $\mathbf{x}$ to be continuous)

2. Requires a smaller learning rate, since there's no upper bound to the value a component might take in the reconstruction

Other types of units include:

1. Softmax and multinomial units
2. Gaussian visible and hidden units
3. Binomial units
4. Rectified linear units

Sampling: Metropolis sampling

In order to sample from the RBM probability distribution it is common to use Markov Chain Monte Carlo (MCMC) algorithms such as Metropolis-Hastings or Gibbs sampling.

Metropolis sampling starts by suggesting a new configuration $\boldsymbol{x}^{k+1}$. In the brute force method this is done by some random change of the visible units. The new configuration is then accepted with the acceptance probability

$$A(\boldsymbol{x}^k \rightarrow \boldsymbol{x}^{k+1}) = \min(1, \frac{P(\boldsymbol{x}^{k+1})}{P(\boldsymbol{x}^k)}), \tag{5}$$

where we need the marginalized probability

$$P(\boldsymbol{x}) = \sum_{\mathbf{h}} P_{rbm}(\mathbf{x}, \mathbf{h}) \tag{6}$$

$$= \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}. \tag{7}$$

Sampling: Gibbs sampling

In this method we sample from the joint probability $P_{rbm}(\mathbf{x}, \mathbf{h})$ by way of a two step sampling process. We alternately update the visible and hidden units. New samples are generated according to the conditional probabilities $P(x_i|\mathbf{h})$ and $P(h_j|\mathbf{x})$ respectively and accepted with the probability of 1. While the the visible nodes are dependent on the hidden nodes and vice versa, the nodes are independent of other nodes within the same layer. This is due to there being no intra layer interactions in the restricted Boltzmann machine.

The conditional probabilities are often referred to as the activitation functions in the neural networks context due to their role in determining the node outputs. For the binary-binary RBM they are

$$P(h_j = 1|\boldsymbol{x}) = \frac{1}{1 + e^{-b_j - \sum_i x_i w_{ij}}} \tag{8}$$

$$P(x_i = 1|\boldsymbol{h}) = \frac{1}{1 + e^{-a_j - \sum_j h_j w_{ij}}}, \tag{9}$$

where we recognize the logistic sigmoid function $\sigma(x) = 1/(1 + exp(-x))$.

Gaussian RBM

For the Gaussian-Binary RBM the conditional probabilities are

$$P(x_i|\mathbf{h}) = \mathcal{N}(x_i; a_i + \sum_j h_j w_{ij}, \sigma^2) \tag{10}$$

$$P(h_j = 1|\mathbf{x}) = \frac{1}{1 + e^{-b_j - \frac{1}{\sigma^2}\sum_i x_i w_{ij}}}, \tag{11}$$

while the visible units now follow a normal distribution, we see the hidden units again follow the logistic sigmoid function.

Cost function

When working with a training dataset, the most common training approach is maximizing the log-likelihood of the training data. The log likelihood characterizes the log-probability of generating the observed data using our generative model. Using this method our cost function is chosen as the negative log-likelihood. The learning then consists of trying to find parameters that maximize the probability of the dataset, and is known as Maximum Likelihood Estimation (MLE). Denoting the parameters as $\boldsymbol{\theta} = a_1, ..., a_M, b_1, ..., b_N, w_{11}, ..., w_{MN}$, the log-likelihood is given by

$$\mathcal{L}(\{\theta_i\}) = \langle \log P_\theta(\boldsymbol{x}) \rangle_{data} \tag{12}$$

$$= -\langle E(\boldsymbol{x}; \{\theta_i\}) \rangle_{data} - \log Z(\{\theta_i\}), \tag{13}$$

where we used that the normalization constant does not depend on the data, $\langle \log Z(\{\theta_i\}) \rangle = \log Z(\{\theta_i\})$ Our cost function is the negative log-likelihood, $\mathcal{C}(\{\theta_i\}) = -\mathcal{L}(\{\theta_i\})$

Optimization / Training

The training procedure of choice often is Stochastic Gradient Descent (SGD). It consists of a series of iterations where we update the parameters according to the equation

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla \mathcal{C}(\boldsymbol{\theta}_k) \tag{14}$$

at each k-th iteration. There are a range of variants of the algorithm which aim at making the learning rate η more adaptive so the method might be more efficient while remaining stable.

We now need the gradient of the cost function in order to minimize it. We find that

$$\frac{\partial \mathcal{C}(\{\theta_i\})}{\partial \theta_i} = \langle \frac{\partial E(\boldsymbol{x}; \theta_i)}{\partial \theta_i} \rangle_{data} + \frac{\partial \log Z(\{\theta_i\})}{\partial \theta_i} \tag{15}$$

$$= \langle O_i(\boldsymbol{x}) \rangle_{data} - \langle O_i(\boldsymbol{x}) \rangle_{model}, \tag{16}$$

where in order to simplify notation we defined the "operator"

$$O_i(\boldsymbol{x}) = \frac{\partial E(\boldsymbol{x};\theta_i)}{\partial \theta_i}, \tag{17}$$

and used the statistical mechanics relationship between expectation values and the log-partition function:

$$\langle O_i(\boldsymbol{x})\rangle_{model} = \mathrm{Tr} P_\theta(\boldsymbol{x}) O_i(\boldsymbol{x}) = -\frac{\partial \log Z(\{\theta_i\})}{\partial \theta_i}. \tag{18}$$

More on RBMs

The data-dependent term in the gradient is known as the positive phase of the gradient, while the model-dependent term is known as the negative phase of the gradient. The aim of the training is to lower the energy of configurations that are near observed data points (increasing their probability), and raising the energy of configurations that are far from observed data points (decreasing their probability).

The gradient of the negative log-likelihood cost function of a Binary-Binary RBM is then

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial w_{ij}} = \langle x_i h_j\rangle_{data} - \langle x_i h_j\rangle_{model} \tag{19}$$

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial a_{ij}} = \langle x_i\rangle_{data} - \langle x_i\rangle_{model} \tag{20}$$

$$\frac{\partial \mathcal{C}(w_{ij}, a_i, b_j)}{\partial b_{ij}} = \langle h_i\rangle_{data} - \langle h_i\rangle_{model}. \tag{21}$$

$$\tag{22}$$

To get the expecation values with respect to the *data*, we set the visible units to each of the observed samples in the training data, then update the hidden units according to the conditional probability found before. We then average over all samples in the training data to calculate expectation values with respect to the data.

Which sampling to use

To get the expectation values with respect to the *model*, we use Gibbs sampling. We can either initialize the $\boldsymbol{x}$ randomly or with a training sample. While we ideally want a large number of Gibbs iterations $n \to n$, one might decide to truncate it earlier for efficiency. Doing this while having intialized $\boldsymbol{x}$ with a training data vector is referred to as contrastive divergence (CD), because one is then closer to approximating the gradient of this function than the negative log-likelihood. The contrastive divergence function is the difference between two Kullback-Leibler divergences (also called relative entropy), which measure how one probability distribution diverges from a second, expected probability distribution (in this case the estimated one from the ground truth one).

RBMs for the quantum many body problem

The idea of applying RBMs to quantum many body problems was presented by G. Carleo and M. Troyer, working with ETH Zurich and Microsoft Research.
Some of their motivation included

- "The wave function Ψ is a monolithic mathematical quantity that contains all the information on a quantum state, be it a single particle or a complex molecule. In principle, an exponential amount of information is needed to fully encode a generic many-body quantum state."
- There are still interesting open problems, including fundamental questions ranging from the dynamical properties of high-dimensional systems to the exact ground-state properties of strongly interacting fermions.
- The difficulty lies in finding a general strategy to reduce the exponential complexity of the full many-body wave function down to its most essential features. That is
    1. $\rightarrow$ Dimensional reduction
    2. $\rightarrow$ Feature extraction
- Among the most successful techniques to attack these challenges, artifical neural networks play a prominent role.
- Want to understand whether an artifical neural network may adapt to describe a quantum system.

Choose the right RBM

Carleo and Troyer applied the RBM to the quantum mechanical spin lattice systems of the Ising model and Heisenberg model, with encouraging results. Our goal is to test the method on systems of moving particles. For the spin lattice systems it was natural to use a binary-binary RBM, with the nodes taking values of 1 and -1. For moving particles, on the other hand, we want the visible nodes to be continuous, representing position coordinates. Thus, we start by choosing a Gaussian-binary RBM, where the visible nodes are continuous and hidden nodes take on values of 0 or 1. If eventually we would like the hidden nodes to be continuous as well the rectified linear units seem like the most relevant choice.

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 $\boldsymbol{h}$

$$F_{rbm}(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-\frac{1}{T_0} E(\mathbf{x}, \mathbf{h})}. \tag{23}$$

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

$$F_{rbm}(\mathbf{x}) = \sum_{\mathbf{h}} F_{rbm}(\mathbf{x},\mathbf{h}) \tag{24}$$

$$= \frac{1}{Z}\sum_{\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}. \tag{25}$$

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

$$\Psi(\mathbf{X}) = F_{rbm}(\mathbf{x}) \tag{26}$$

$$= \frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(\mathbf{x},\mathbf{h})} \tag{27}$$

$$= \frac{1}{Z}\sum_{\{h_j\}} e^{-\sum_i^M \frac{(x_i-a_i)^2}{2\sigma^2}+\sum_j^N b_j h_j+\sum_{i,j}^{M,N}\frac{x_i w_{ij} h_j}{\sigma^2}} \tag{28}$$

$$= \frac{1}{Z}e^{-\sum_i^M \frac{(x_i-a_i)^2}{2\sigma^2}}\prod_j^N (1+e^{b_j+\sum_i^M \frac{x_i w_{ij}}{\sigma^2}}). \tag{29}$$

$$\tag{30}$$

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

$$G_i = \frac{\partial\langle E_L\rangle}{\partial\theta_i} = 2(\langle E_L\frac{1}{\Psi}\frac{\partial\Psi}{\partial\theta_i}\rangle - \langle E_L\rangle\langle\frac{1}{\Psi}\frac{\partial\Psi}{\partial\theta_i}\rangle), \tag{31}$$

where the local energy is given by

$$E_L = \frac{1}{\Psi}\hat{\mathbf{H}}\Psi. \tag{32}$$

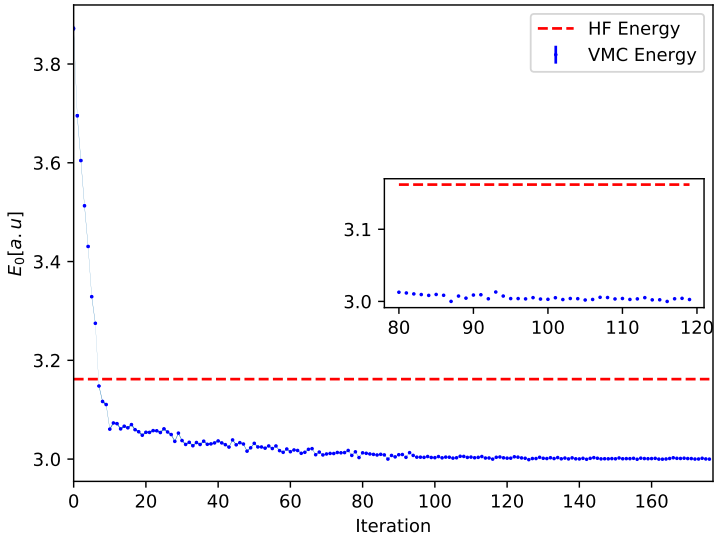
Running the codes

You can find the codes for the simple two-electron case at the Github repository https://github.com/mhjensenseminars/MachineLearningTalk/tree/master/doc/Programs/MLcpp/src. Python codes to come, only c++ as of now.

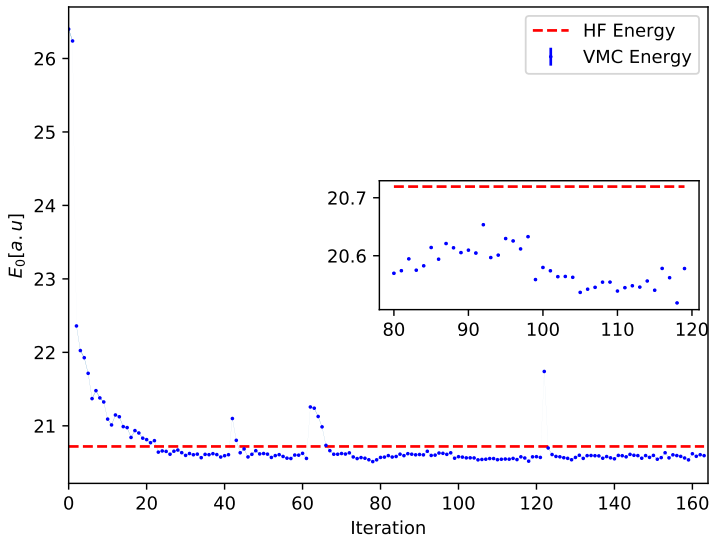
The trial wave function is based on the product of a Slater determinant with Gaussian orbitals, a simple Jastrow factor $\exp(r_{ij})$ and the reduced Boltzmann machines.

The Broyden-Fletcher-Goldfarb-Shanno algorithm was used to perform the minimization. We used 14 hidden nodes in the calculations below.

Energy as function of iterations, $N = 2$ electrons



Energy as function of iterations, $N = 6$ electrons



Conclusions and where do we stand

- A simple extension of the work of G. Carleo and M. Troyer, Science **355**, Issue 6325, pp. 602-606 (2017) gives excellent results for two-electron systems as well as good agreement with standard VMC calculations for $N = 6$ and $N = 12$ electrons.
- Minimization problem can be tricky.
- Anti-symmetry dealt with multiplying the trail wave function with an optimized Slater determinant.
- To come: Analysis of wave function from ML and compare with diffusion and Variational Monte Carlo calculations as well as the analytical results of Taut for the two-electron case.
- Extend to more fermions. How do we deal with the antisymmetry of the multi-fermion wave function?

1. Here we used standard Hartree-Fock theory to define an optimal Slater determinant. Takes care of the antisymmetry. What about constructing an anti-symmetrized network function?
2. Use thereafter ML to determine the correlated part of the wafe function (including a standard Jastrow factor).
3. Test this for multi-fermion systems and compare with other many-body methods.

- Can we use ML to find out which correlations are relevant and thereby diminish the dimensionality problem in say CC or SRG theories?