Computational Statistical Physics
Exercise sheet 6

FS 2020
Oded Zilberberg

Exercise 1. Generating Ising configurations with the Restricted Boltzmann Machine

Goal: In this exercise we are going to learn a) what Restricted Boltzmann Machines are, b) how they can be trained and c) how they can be used to generate Ising configurations at a certain temperature.

Task 1: *Read carefully through chapter 1.9 of the lecture notes and familiarize yourself with the concepts of a neuron, the Hopfield Network and the Boltzmann Machine.*

A Restricted Boltzmann Machine (RBM) is a neural network consisting of two layers of neurons where every neuron of one layer is connected with every neuron of the other layer (inter-layer connections between every two neurons). Within the same layer neurons are not connected (no intra-layer connections). A schematic is presented in figure 1.

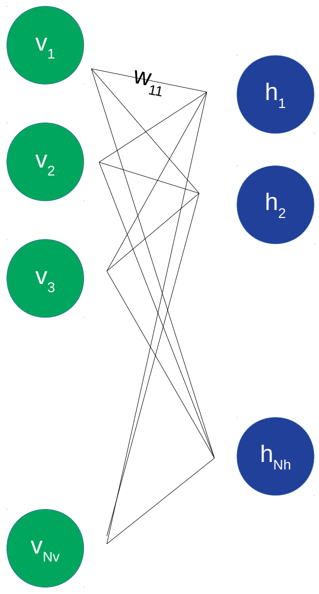

Figure 1: Schematic of a RBM. Visible layer (green). Hidden layer (blue).

One of the two layers is called *visible* layer while the other one is called *hidden* layer. Interacting with the machine (input and output) can only occur over the visible layer. The hidden layer is not directly accessible. Moreover, the neurons are *binary*, i.e., they can only take one of two possible values - either 0 or 1.

Let's call the number of visible nodes N_v and the number of hidden nodes N_h. Furthermore, call the current value of the j-th node in the visible layer v_j and the i-th node in the hidden layer h_i. With these definitions we are able to have a closer look at the dynamics of the system. Given $\boldsymbol{v} = (v_1, .., v_{N_v})$ the value of the i-th node in the hidden layer is set to 1 with probability

$$p(h_i = 1|\boldsymbol{v}) = \sigma\left(\sum_{j=1}^{N_v} w_{ij} v_j + b_i\right)$$

else it is set to 0. The coefficients w_{ij} are called *weights* and the coefficients b_i are called *biases* (of the hidden layer). $\sigma(x)$ is the *sigmoid function*

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

which maps any real number to the interval $(0,1)$. Similarly, given the values $\boldsymbol{h} = (h_1, .., h_{N_h})$ of the hidden layer the value of the j-th visible node is determined by

$$p(v_j = 1|\boldsymbol{h}) = \sigma\left(\sum_{i=1}^{N_h} w_{ji} h_i + a_j\right)$$

where a_j are the biases of the visible layer. Note that the weights are symmetric, i.e., $w_{ij} = w_{ji}$. Due to these update rules the RBM is classified as a *stochastic* model.

Task 2: *State and explain the differences between a Hopfield Network, a Boltzmann Machine and a Restricted Boltzmann Machine.*

In the following we are going to use the RBM to generate $2D$ Ising configurations with $L = 32$ at a certain temperature T. Therefore, we choose the number of visible nodes to be $N_v = 32 \times 32$.

Before samples can be drawn the machine has to be *trained.* By training we mean updating the weights and biases according to our training data.[1] This is done via *contrastive divergence.* The update rule for the weights is given by

$$w_{ij} \to w_{ij} - \epsilon\left(\langle v_j h_i\rangle_{data} - \langle v_j h_i\rangle^k_{model}\right)$$

where ϵ is a so-called *learning rate.* The expectation values are understood to be averages over the whole set of training data.[2] The quantity $(v_j h_i)_{data}$ is calculated by taking a vector $\boldsymbol{v}$ from the training data and computing the corresponding vector $\boldsymbol{h}$ as described above. For the quantity $(v_j h_i)^k_{model}$ one has to take a vector from the training data, compute the corresponding vector $\boldsymbol{h}$, compute the new $\boldsymbol{v}$ and perform k more back-and-forth operations. More information about the contrastive divergence can be found here: `https://arxiv.org/pdf/1803.08823.pdf` (p. 90 ff.).
For completeness, the update rules for the biases are given by

$$a_j \to a_j - \epsilon\left(\langle v_j\rangle_{data} - \langle v_j\rangle^k_{model}\right)$$
$$b_i \to b_i - \epsilon\left(\langle h_i\rangle_{data} - \langle h_i\rangle^k_{model}\right).$$

For the following tasks we provide you with a python project which is missing some functionality that you have to implement. At first, the training data is extracted and brought into the right shape (implementation found in "*ising_main.py*"). Then, the RBM is set up and trained for one fixed temperature T (implementation found in "*my_RBM_tf2.py*"). Finally, new Ising configurations are generated and the magnetization and energy are plotted in "*plotting.py*".

Task 3: *Implement the function "contr_divergence" in the class "RBM" in the file "my_RBM_tf2.py" as described above.*

Hint: You might find the following functions helpful:

[1] We provide you with 1000 Ising configurations for three different temperatures. They are found in the file "*ising_data_L32.h5*".

[2] Note that this procedure is in general very slow because the averages are always computed over the whole training data. Instead of performing this kind of optimization for the weights and biases it is beneficial to divide the whole set of training data into a set of mini-batches and compute the averages only over these mini-batches.

- *tensorflow.sigmoid*
- *tensorflow.add*
- *tensorflow.tensordot*
- *tensorflow.transpose*
- *tensorflow.reshape*

Check out the tensorflow documentation (`https://www.tensorflow.org/api_docs/python`) for more information.

Task 4: *Use the training data stored in "data/ising/ising_data_L32.h5" to find the optimal weights and biases for your RBM (you can also generate additional data using the file "wolff.jl"). (Disclaimer: Training the machine may take quite a while depending on your computer.)*

Hint: You can use the file ising_main.py to train the RBM. The weigths will be stored in results/models/... .

Once the machine is trained it can be used to generate new samples. This can be done in the following way:

1. Set the nodes in the visible layer to random values (either 0 or 1).
2. Let the machine evolve i.e. go several times back and forth between the visible and hidden layer.
3. Read out the nodes in the visible layer. This is the desired sample.

Task 5: *Use the RBM to obtain new Ising configurations. Plot the energy and the magnetization as a function of sampling time.*

Hint: You can use the file plotting.py .

Task 6: *Repeat Task 4 and Task 5 for at least two more temperatures.*

Exercise 2. Classifying temperatures of Ising configurations with a feed-forward network

> *Goal: Here, we are going to learn about another kind of neural network. This time we are not going to generate new Ising configurations but instead determine the temperatures of given Ising configurations.*

Consider a neural network made up of 4 layers as displayed in figure 2.

The two outer layers (used for input and output) are visible while the two inner layers are hidden. In contrast to the RBM there is no back-and-forth flow of information between the visible and hidden layers. Instead, information flows from the input to the output layer. That is why this network is called a *feed-forward* network. Furthermore, we assume that the nodes are not binary anymore but can take continuous values between 0 and 1 and that the dynamics

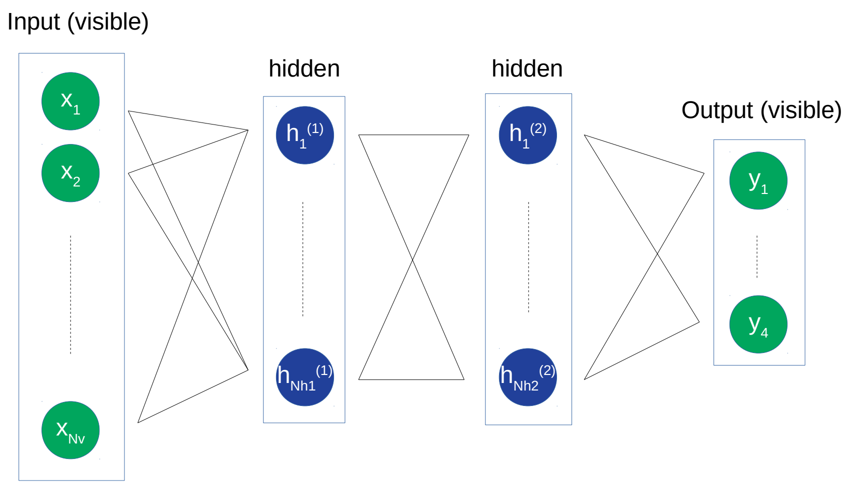

Figure 2: Schematic of a feed-forward network with two hidden layers. Visible layers (green). Hidden layers (blue).

of the system is given by

$$h_k^{(1)} = \sigma\left(\sum_l w_{kl}^{(1)} x_l + b_k^{(1)}\right)$$

$$h_j^{(2)} = \sigma\left(\sum_k w_{jk}^{(2)} h_k^{(1)} + b_j^{(2)}\right)$$

$$y_i = \sigma\left(\sum_j w_{ij}^{(3)} h_j^{(2)} + b_i^{(3)}\right)$$

where $\sigma(x)$ is again the sigmoid function. Due to the fact that the values of the nodes are uniquely defined (not set with a certain probability as in the RBM) the network is called *deterministic*.

Since the goal is to map an Ising configuration to its corresponding temperature the input layer is chosen to have 32×32 nodes while the ouput layer consists of 3 nodes (one for every possible temperature we would like to detect). Thus, the values of the nodes in the output layer can be interpreted as probabilities that the system has a certain temperature.

Training the machine means again that the weights and biases have to be adjusted. This is done in a way such that the so-called *cost function* (also *loss function*) is minimized. Given some input $\boldsymbol{i}^{(d)}$ with expected output $\boldsymbol{o}^{(d)}$ the *(mean-squared) cost* of this single training example is defined as

$$C^{(d)} = \sum_{i=1}^{4} \left(y_i^{(d)} - o_i^{(d)}\right)^2 .$$

With this the *total cost function* C is defined as the average of all costs over the whole training data set[3]

$$C\left(w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, w^{(3)}, b^{(3)}\right) = \frac{1}{N_{data}} \sum_{d=1}^{N_{data}} C^{(d)}.$$

[3] Note that in our case $N_{data} = 1000$.

The most straightforward way to do the updates of the weights and biases is by using a *steepest descent* method. However, such a method is usually slow because one has to average over all data of the training set in every step. Therefore, similar to Exercise 1, we randomly divide the set of training data into mini-batches and compute the gradient only for one of these mini-batches in one step. This procedure is known as *stochastic gradient descent.* The update rule can be stated in the following form:

$$\begin{pmatrix} w^{(1)} \\ b^{(1)} \\ w^{(2)} \\ b^{(2)} \\ w^{(3)} \\ b^{(3)} \end{pmatrix} \to \begin{pmatrix} w^{(1)} \\ b^{(1)} \\ w^{(2)} \\ b^{(2)} \\ w^{(3)} \\ b^{(3)} \end{pmatrix} - \epsilon \begin{pmatrix} \partial_{w^{(1)}} \\ \partial_{b^{(1)}} \\ \partial_{w^{(2)}} \\ \partial_{b^{(2)}} \\ \partial_{w^{(3)}} \\ \partial_{b^{(3)}} \end{pmatrix} C.$$

The gradient ∇C of the cost function can be computed via *backpropagation* which is nothing else but the chain rule.

Task 1: *State and derive the analytical expressions for* $\partial_{w^{(3)}_{i,j}} C$ *and* $\partial_{w^{(2)}_{i,j}} C$.

Task 2: *Build up the network and train your machine with the test data provided in "ising_data_L32.h5".*

Hint: You may use the python skeleton "measuring_temperature.py". It will store the weights in training_1/... .

Task 3: *Use the samples you generated in exercise 1 and determine the corresponding temperatures using the network from this exercise.*

Hint: You can use the file plotting.py and extend it to measure the temperature. To load the weights use:
model = create_model()
model.load_weights('Training_1/cp.ckpt')

Task 4 (optional): *In the end machine learning is about trial and error, i.e., finding the best model to describe and successfully predict the kind of data you consider. Therefore, modify your feed-forward network and see which modifications yield the best results. There are several things you can change. To mention only a few:*

- *Number of hidden layers*
- *Number of nodes in the hidden layers*
- *Activation function/non-linearity (instead of the sigmoid function one can use the ReLU, Softmax, etc.)*
- *Cost function (instead of the mean-squared cost function one can use the categorical cross-entropy, etc.)*
- *...*

Feel free to try anything which comes to your mind!