# Exercise class 6

Maximilian Holst
Toni Heugel

# Solution exercise sheet 5

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

**Computational Statistical Physics**
**Exercise sheet 5**

FS 2020
Oded Zilberberg

## Heisenberg model

The Heisenberg model can be seen as a generalization of the Ising model, where a more realistic model of classical magnetization is represented by the Hamiltonian

$$H = -J \sum_{\langle i,j \rangle} \vec{S}_i \cdot \vec{S}_j, \tag{1}$$

where $\vec{S}_i \in \mathbb{S}^2 \subset \mathbb{R}^3$ belongs the surface of a 3-sphere.

While for the XY model (spins in $\mathbb{S}^1$) it is advantageous to use polar coordinates, it is unclear and possibly system dependent if in the 3D spin case the added complexity and FLOP introduced by spherichal coordinates, justifies the memory savings.

Implement a MC simulation of the Heisemberg model with a Metropolis algorithm. Outside a few changes in the update probability computation and the representation of you cluster, you should not need to modify the code from previous exercises much. Remember to normalize your vectors if you use a cartesian representation.

**Taks 1:** Compute the critical temperature for $J = 1$. You can use either the binder cumulants, or the magnetic susceptibility. You should find $T_c \approx 1.443$.

**Task 2:** Compute the autocorrelation time, either for $E$ or $|\vec{M}|$, at $T_c$ and find the critical dynamical exponent given by the relation $\tau \propto L^{z_c}$.

The cluster algorithm can be extended to system with spins of arbitrary dimensions, with the trick of considering reflections around a random plane at each MC step. For each step select a random unit vector $\vec{r}$ which is chosen to be orthogonal to the reflection plane. Decide if you want to implement Swedsen-Wang or Wolff, grow a cluster with bond probability

$$p_{i,j} = 1 - \exp\left[-2\beta(\vec{S}_i \cdot \vec{r})(\vec{S}_j \cdot \vec{r})\right] \tag{2}$$

and flip the spins as

$$\vec{S}_i{}' = \vec{S}_i - 2\vec{r}(\vec{S}_i \cdot \vec{r}). \tag{3}$$

**Task 3:** Repeat the computation of $T_c$ and $z_c$ using the Swendsen-Wang or Wolff algorithm.

## Unsupervised machine learning

We can use PCA and k-means to investigate condensed matter systems. One of the paradigmatic systems is the two-dimensional ferromagnetic Ising model. It has two phases separated by a well

# Implementation - Heisenberg

We want:

- adapt the Metropolis code to the Heisenberg Hamiltonian
- Compute the autocorrelation time and the critical dynamical exponent
- Do the same for a cluster algorithm

Changes Metropolis:

- $\overrightarrow{S}_{old} \rightarrow \overrightarrow{S}_{new}$ , $\overrightarrow{S}_{new}$ is new random spin on $S^2$

Changes Wolff

- Spin is reflected
- New bond probability $p_{i,j} = 1 - \exp\left[-2\beta(\vec{S}_i \cdot \vec{r})(\vec{S}_j \cdot \vec{r})\right]$

# Implementation - Heisenberg

Critical dynamical exponent:

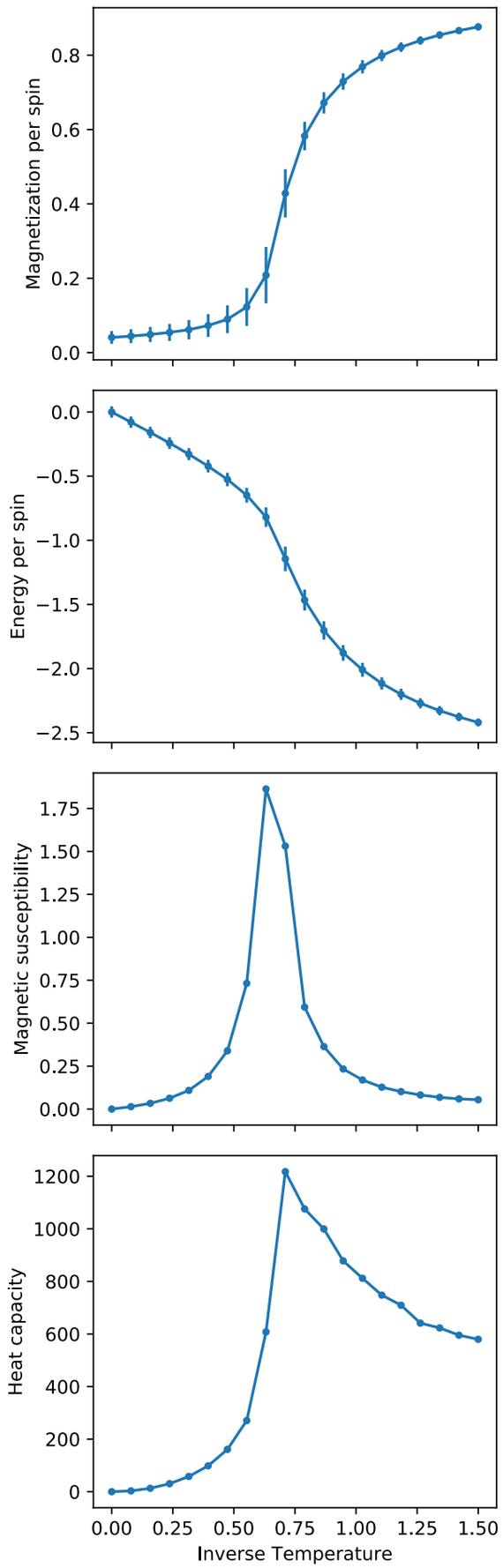
- $\tau \propto L^{z_c}$
- $\tau$ can be found like in exercise 3
- Calculate $\tau$ at $T_c$ for different $L$
- Plot $\log L$ vs. $\log \tau$
  - Slope gives $z_c$

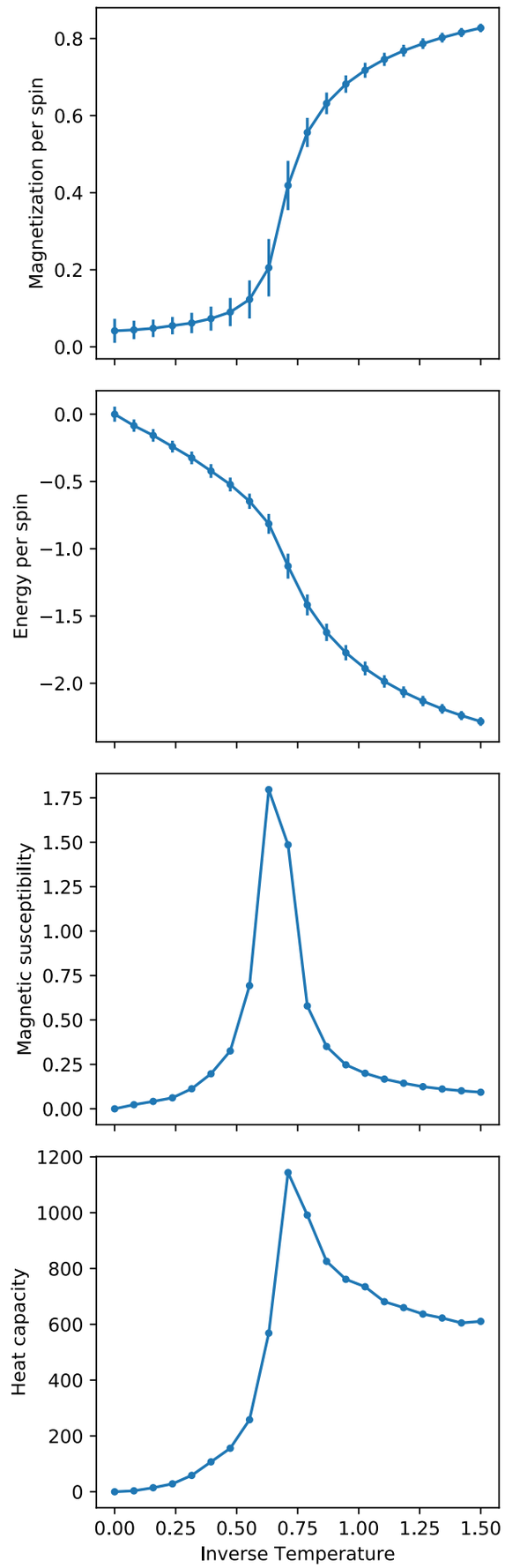
# Code

# Results

Metropolis:

- $z_c \approx 1.8$




Wolff:

- $z_c \approx 0.8$

# Implementation - Unsupervised ML

We want:

- Use PCA to find order parameter
- Use k-means to find centers of the clusters

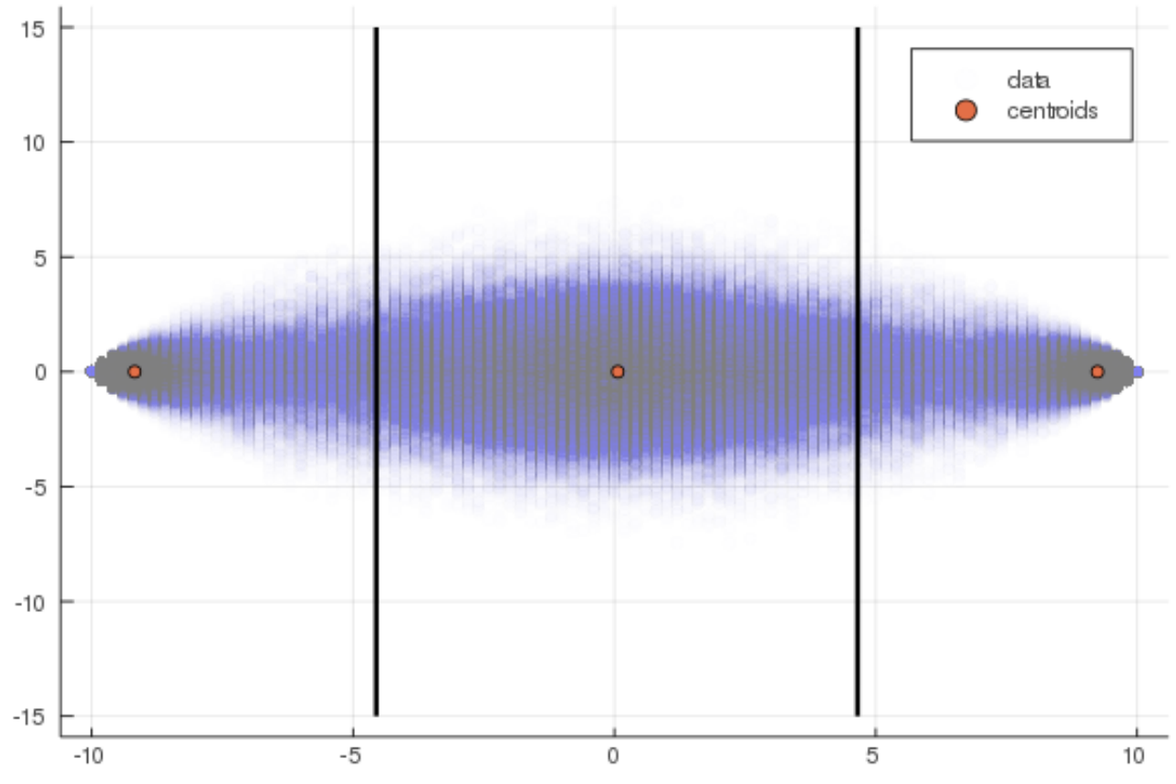
Implementation

- We perform PSA using MultivariateStats to find the 2 most important directions
- Use transformed data to find the average at different temperatures
- Use k-means from Clustering
- Divide the plane into 3 regions

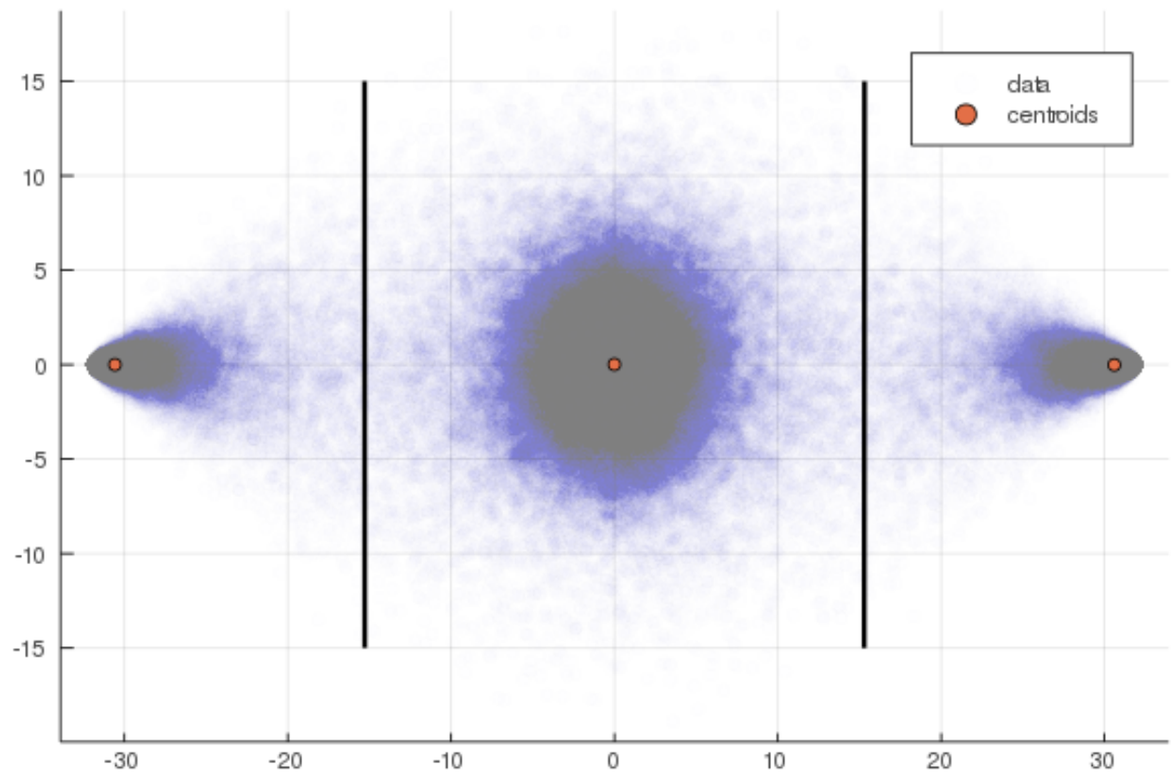
# Code

# Results

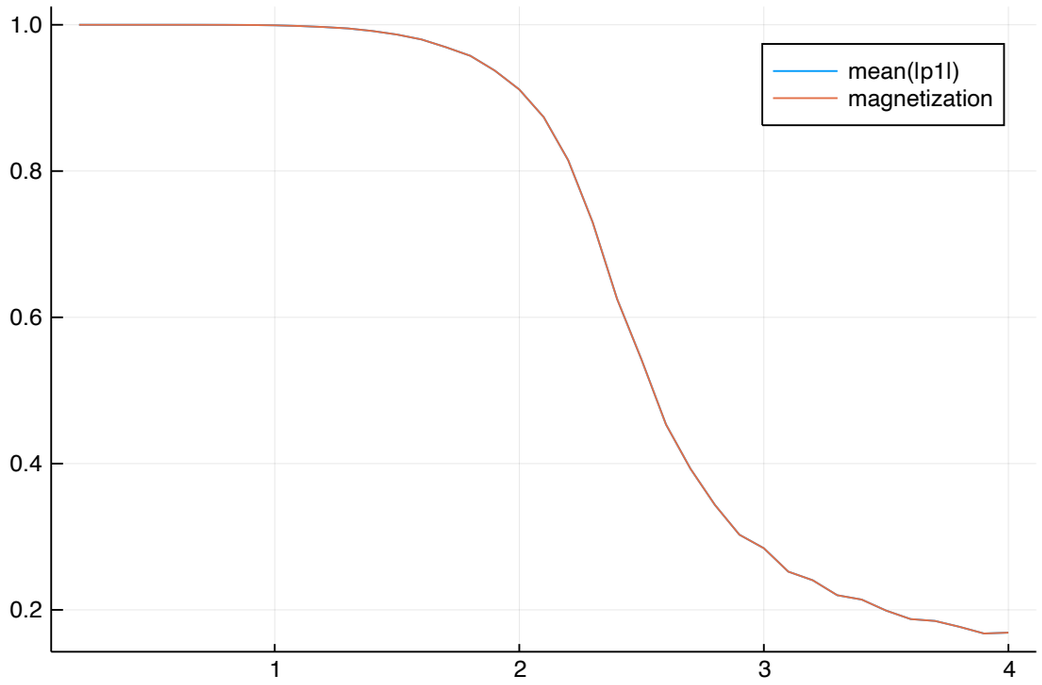
L=10

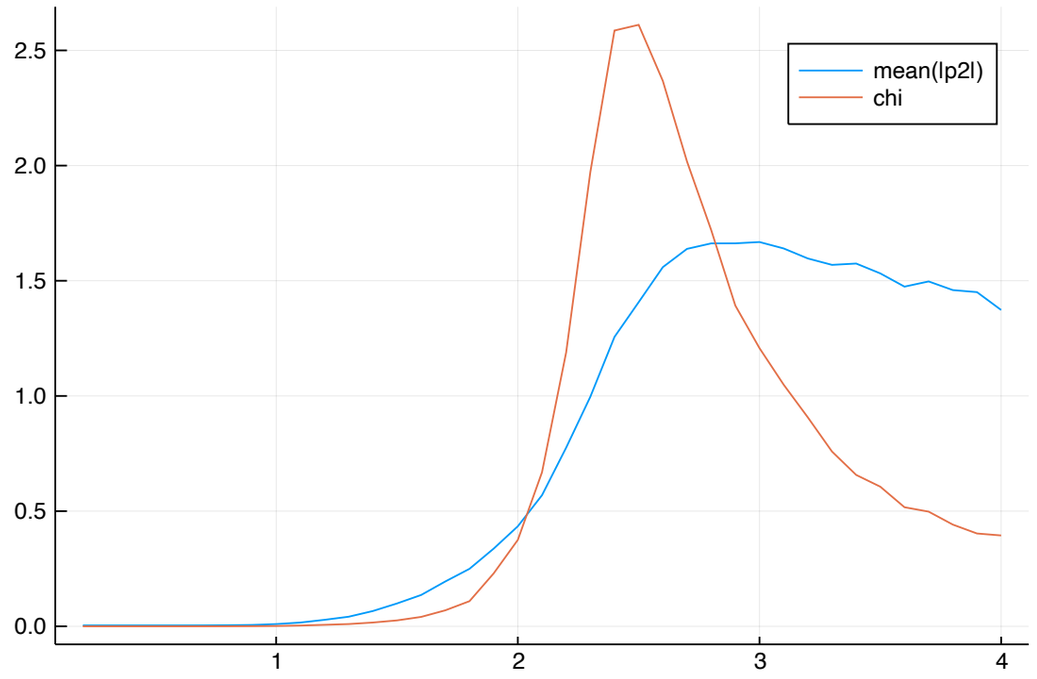


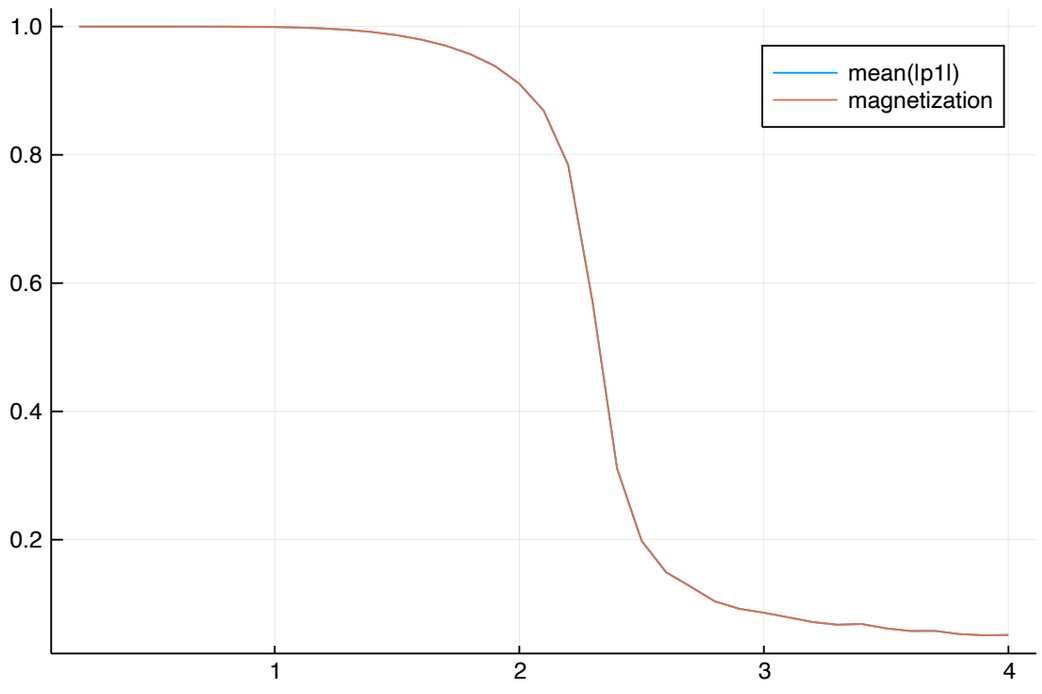

L=32

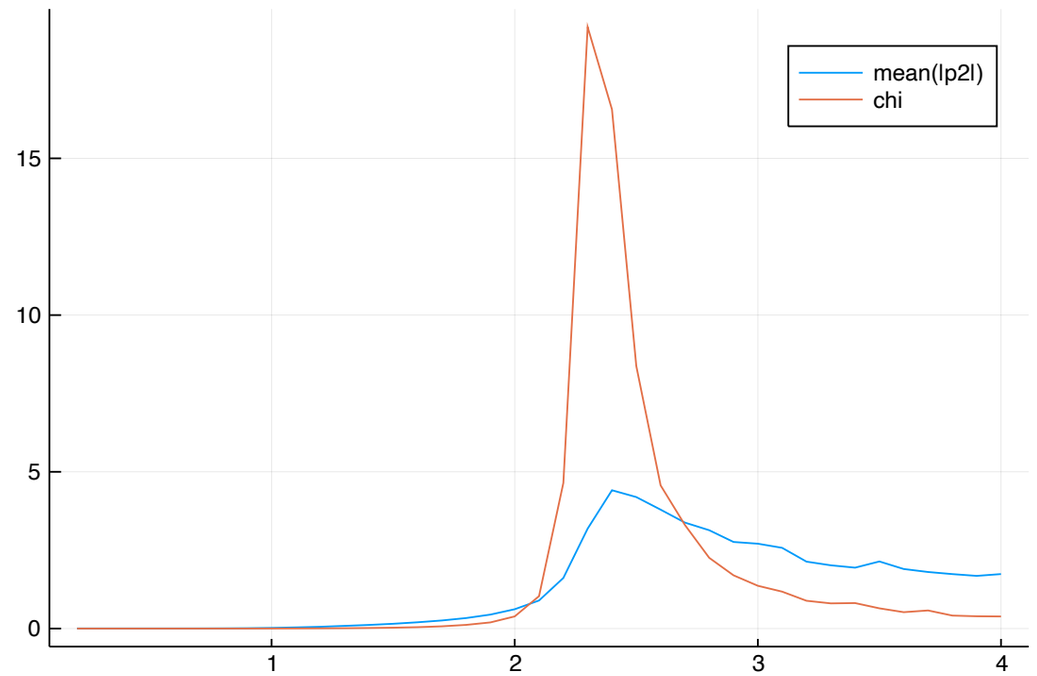


# Results


mean(lp1l)
magnetization
1.0
0.8
0.6
0.4
0.2
1
2
3
4


mean(lp2l)
chi
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4

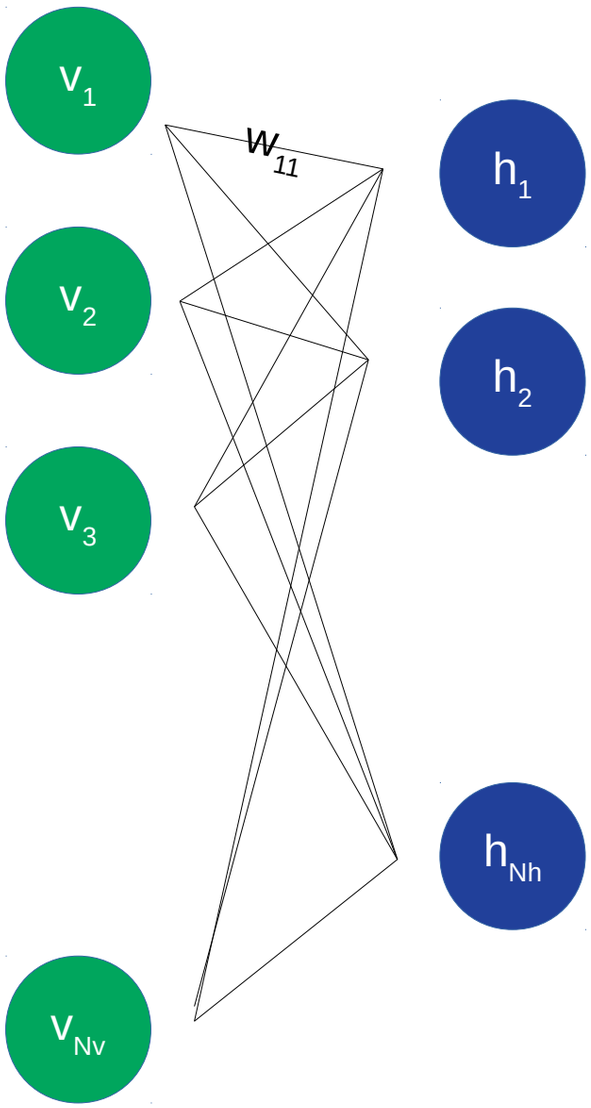

mean(lp1l)
magnetization
1.0
0.8
0.6
0.4
0.2
1
2
3
4


mean(lp2l)
chi
15
10
5
0
1
2
3
4

# Exercise sheet 6

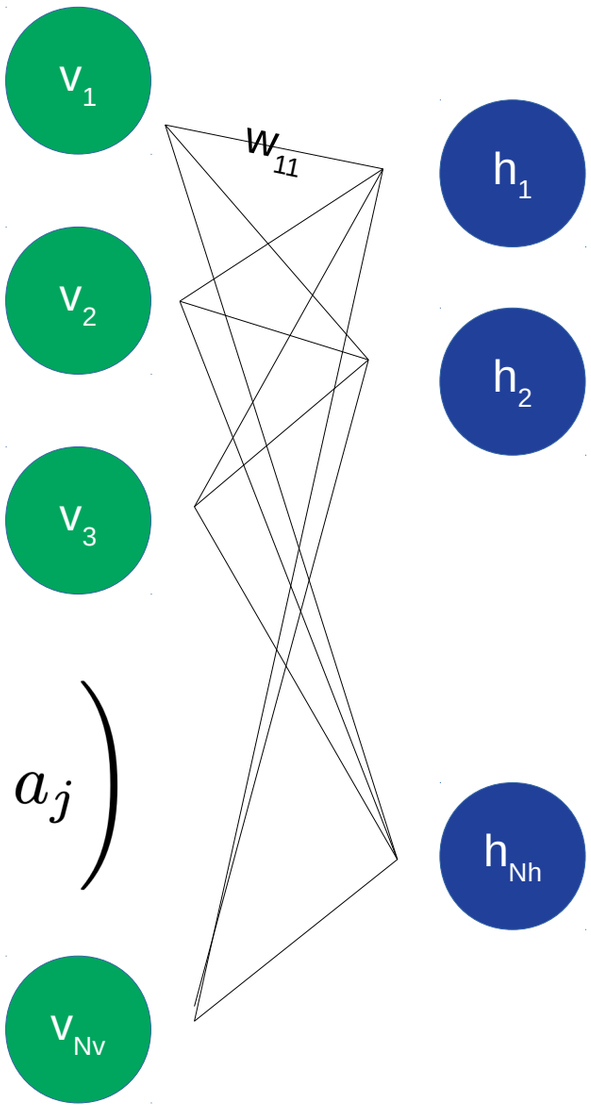
# Restricted Boltzmann Machine

- Inter-layer connections
- No intra-layer connections



# Restricted Boltzmann Machine

- Inter-layer connections
- No intra-layer connections




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

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

# Restricted Boltzmann Machine

Training:

- Update weights and biases according to training data
- Contrastive divergence:

$$w_{ij} \rightarrow w_{ij} - \epsilon \left( \langle v_j h_i \rangle_{data} - \langle v_j h_i \rangle_{model}^{k} \right)$$

vector from training data

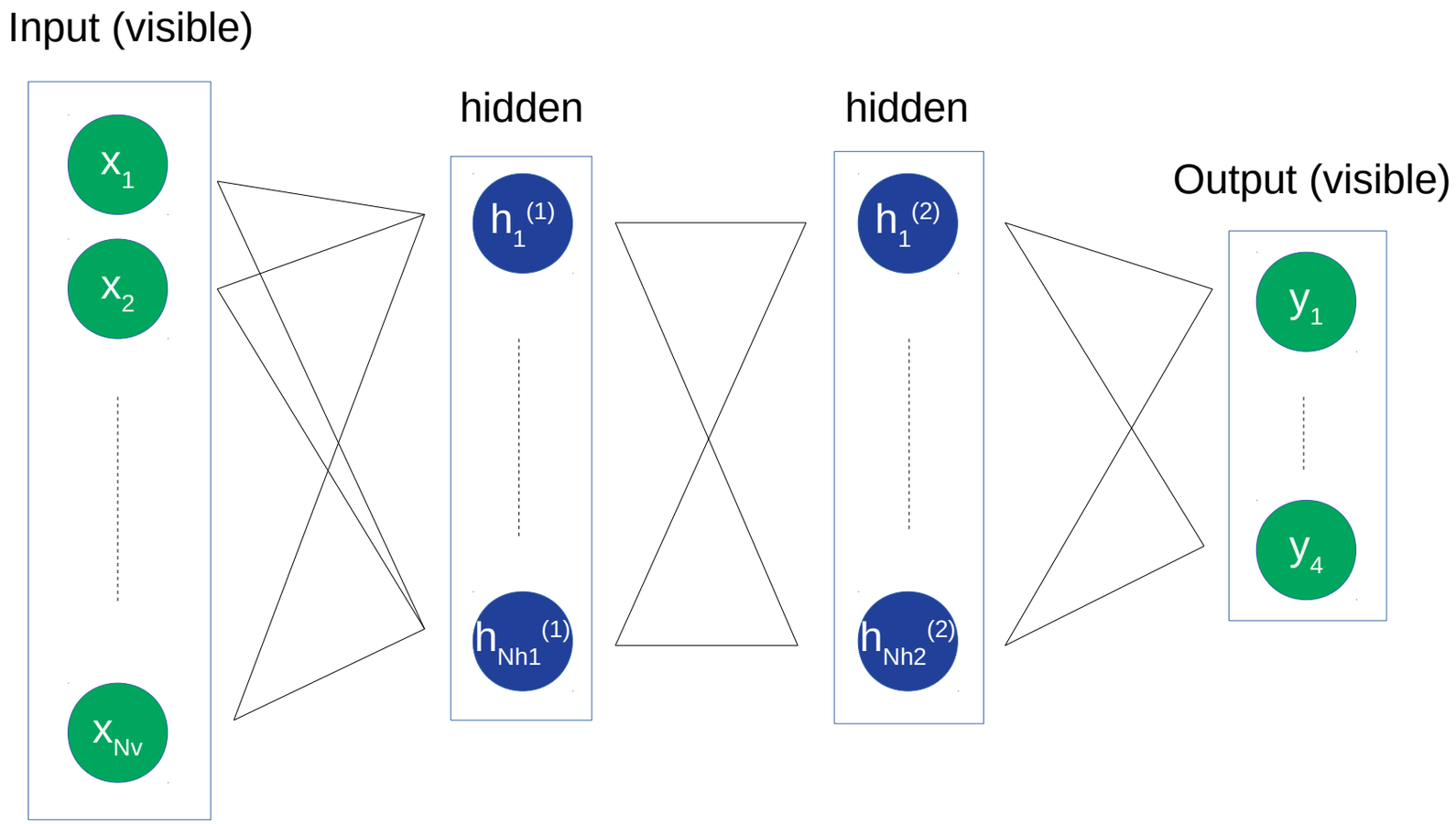
update h

go back and forth (update) k times

$$a_j \rightarrow a_j - \epsilon \left( \langle v_j \rangle_{data} - \langle v_j \rangle_{model}^{k} \right)$$

$$b_i \rightarrow b_i - \epsilon \left( \langle h_i \rangle_{data} - \langle h_i \rangle_{model}^{k} \right)$$

# Feed forward network




Input
(spin configuration)

Output
(temperature
- probability)

# Feed forward network

Training:

- Update weights and biases
- Minimize *cost function*

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

output

correct/expected output

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

# Feed forward network

Training:

- Update weights and biases
- Minimize *cost function*
- Gradient descent

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

# Exercise sheet 6

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

**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.

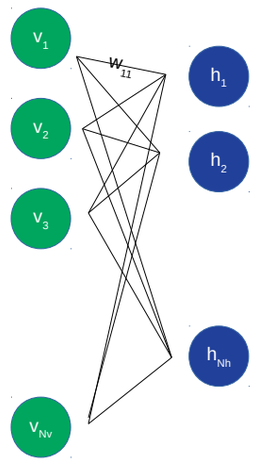



Figure 1: Schematic of a RBM. Visible layer (green). Hidden layer (blue).

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

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

# Instructions

- Read carefully through chapter 1.9
- Make sure you understand the concepts
- Implement contrastive divergence
- Train the RBM and investigate the generated samples

- Build up the network
- Implement your network in *measuring_temperature.py*
- Determine temperature of the samples generated by the RBM

# Python skeleton