Institute for Theoretical Physics
University of Cologne

Prof. Dr. S. Trebst
Kai Meinerz, Jan Attig

Computational Many-Body Physics

Exercise Sheet 5

Summer Term 2021

Due date: Monday, **28th June** 2021, 10 am

Website: thp.uni-koeln.de/trebst/Lectures/2021-CompManyBody.shtml

With this week's exercise sheet we want to introduce you to some of the concepts of **machine learning** and their application to statistical many-body physics. We start by providing you with a relatively simple and straightforward Julia code for the elementary example of handwritten digit recognition, introducing you to the Flux machine learning package on the way.

We will then expand this code and apply it to a physics context, by using it to classify configurations of the Ising model, to discriminate its two phases, and to identify the parametric location of its phase transition. This is done using a *supervised* learning approach as well as an *unsupervised* one.

In addition, we will showcase the (variational) autoencoder and use it to reconstruct images affected by different types of noise and obstructions.

Exercise 12: Digit Recognition (Warm-up exercise)

As a warm-up exercise we start with the classical machine learning problem of recognizing handwritten digits. The idea is is to convert 28×28 pixelated images of hand-written digits from the MNIST dataset to their actual digital counterparts.

You can download our example code from the course website and run all cells from the notebook. Make sure that the code runs without problems and that you understand the Flux syntax.

Figure 1: Example digits from the MNIST dataset, along with their label

Exercise 13: The supervised Ising model

In this exercise we want to generalize the digit recognition code to the physics of the Ising model, i.e. to do a supervised learning of its phases. These two problems are indeed closely related as it is, in both cases, the goal to classify an element (an image or ab Ising spin configuration) into a small amount of categories (digits or phases). You can read more about it in a recent publication, J. Carrasquilla and R. Melko, Nature Physics **13**, 431 (2017).

The main task of this exercise is to exchange the training data employed the code of exercise 12. Instead of supplying images of digits and their integer values, we now want to provide Ising spin configurations and assign them a label indicating the phase they belong to. This allows the identification of phases in newly generated configurations and the determination of the transition temperature between those phases as seen later on.

a) To get started, make sure you have a working Monte Carlo code that can simulate the ferromagnetic Ising model on the square lattice using either single-spin flip updates or the Swendsen-Wang cluster updates.

With this code, implement the **generation of training and testing data**. For the trainings data you should choose one low and one high temperature point sufficiently far away from the phase transition (for the square lattice, you can choose, e.g., $T_{low} = 1.0$ and $T_{high} = 4.0$). Make sure that your simulation is properly thermalized and record around 50.000 configurations for each temperature. Then compile the training data using the labels 0 and 1 respectively for the configurations at the low and high temperatures. For the test data set you should sample configurations at a greater range of temperatures. Use 20 temperatures points between your T_{low}/T_{high} and label all configurations sampled below T_c with 0 and above with 1. For this exercise a lattice size of $L = 8$ is enough.

b) With a training and a testing set at hand, move on to **designing the neural network**. You can base your design on the code provided for exercise 12, i.e. one input layer, one hidden layer and one output layer. The input layer size is fixed by the system size of your lattice, the output layer size is 2 because there are 2 phases to distinguish (similar to 10 distinguishable digits earlier).

Train your neural network and adapt the learning rate as well as batch size and epoch number to your needs (What are good values that you find in your testing?). After training, the neural network should be able to accurately distinguish between configurations of your two temperatures and therefore be able to identify if a configuration resembles the ferromagnetic or the paramagnetic phase.

c) The trained neural network can then be used to **find the critical temperature** of the phase transition between the ferromagnetic and paramagnetic phase by investigating how the network output changes when feeding the network with configuration for intermediate temperatures from the test set. For temperatures within the ferromagnetic phase, configurations look (reasonably) similar to the training data of the ferromagnetic phase and the network will likely give a similar output. But what happens when you further increase the temperature and eventually cross into the temperature regime of the paramagnetic phase? Visualize this crossover of probabilities by plotting the expectation values of the network for the two phases for each temperature. Investigate the crossing point and compare to the phase transition temperature.

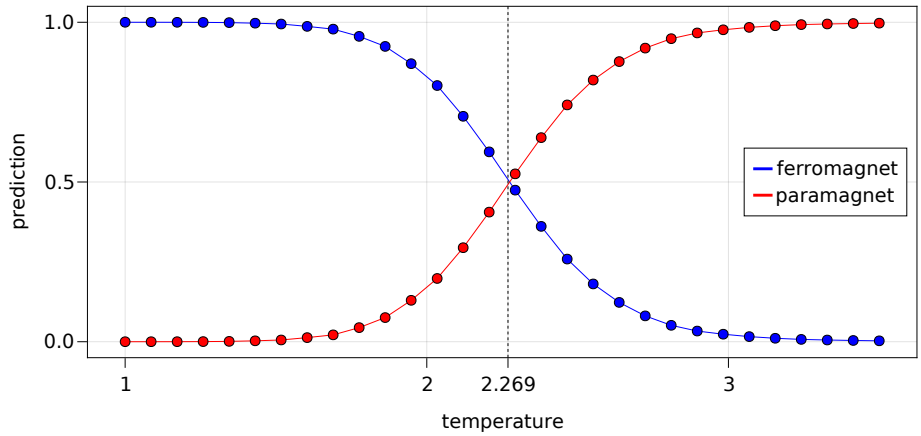

Figure 2: **Phase prediction of a neural network** for the Ising model trained at $T_{low} = 1.0$ and $T_{high} = 5.0$. The crossing of the different phase prediction matches well with the critical temperature.

d) *(Optional Exercises)* Investigate how you can further **improve the prediction quality** of the previous transition temperature plot. You might want to try and extend some of the following ideas:

- Try to use the Swendsen Wang algorithm to generate configurations for the lower temperature as it generates global updates in contrast to local single-spin flips.
- Choose different temperatures for the trainings data. What happens if one temperature significantly is closer to the phase transition then the other?
- Try out to provide not only one, but multiple temperature points as training data for the ferromagnetic and paramagnetic phase.
- Play with the values of the learning rate, batch size, and epoch number and investigate how the learning speed changes.
- ...

Exercise 14: The unsupervised Ising model

We now want to move to an **unsupervised approach** to discriminate the phases of the Ising model. This has the obvious advantage that we need no prior knowledge about its phase diagram (and no labels whatsoever). We will do so by employing the principal component analysis (PCA), which is used to cluster data by generating a low-dimensional representation. We will apply the PCA on the Ising model to cluster the data according to the different phases of the system. You can read more about PCA in the publications of Lei Wang Phys. Rev. B 94, 195105(2016) and Sebastian J. Wetzel Phys. Rev. E 96, 022140 (2017).

a) First of all you have to generate your dataset $X = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}\}$ for the PCA. Use your Monte Carlo simulation to generate various configurations $\mathbf{x_i}$ in the temperature range $T \in [1, 5]$. A small dataset with $N = 10^4$ configurations of a $L = 20$ system will be sufficient for this exercise.

b) Now we want to perform the PCA on the data by following along the steps presented in the lecture. First you have to calculate the **correlation matrix** S_X of your dataset:

$$S_X = \frac{1}{N-1} X X^T . \tag{1}$$

Next step is the diagonalization of S_X into S_Y in order to reduce the correlations. For the diagonalization you can use the eigendecomposition method. It follows:

$$S_Y = P S_X P^T , \tag{2}$$

where P is the **principal component matrix** and the matrix of eigenvectors of X at the same time.Therefore the k-th principle component $\mathbf{w}_k$ corresponds to the eigenvector of the k-th largest eigenvalue. Calculate the first two principle components.

c) Using this result we can assign our original data a new value $y_{k(i)} = \mathbf{w}_k \mathbf{x}_i$. Calculate $\mathbf{y}_1, \mathbf{y}_2$ and plot $\mathbf{y}_2$ against $\mathbf{y}_1$. What can you observe?

d) The first principle components actually identifies the magnetization of the system as its first principal component. Show this relation by plotting $\mathbf{y}_1$ against the magnetization.

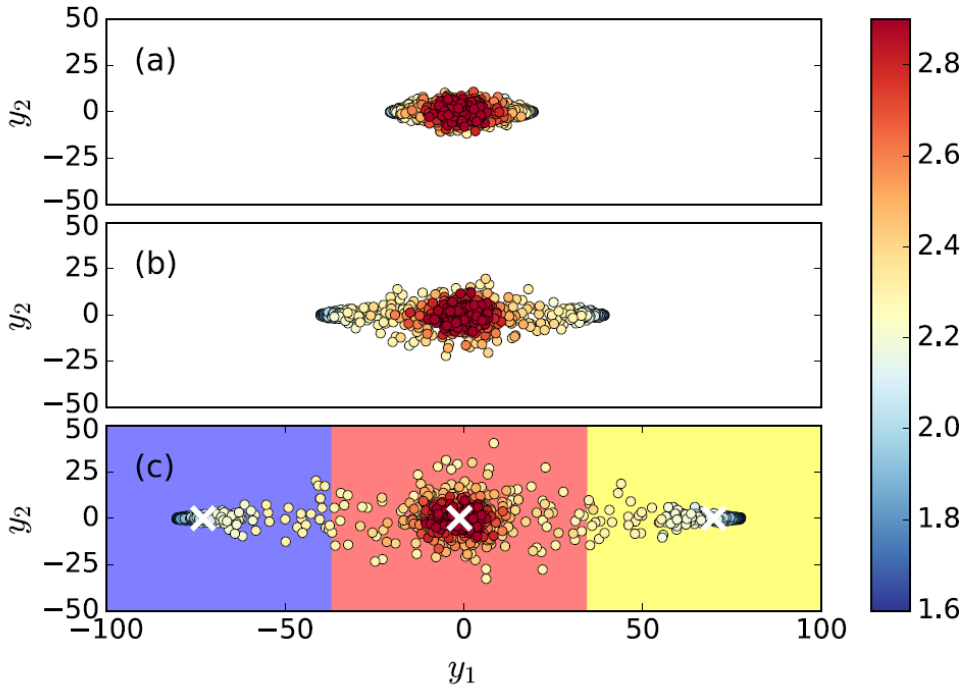

Figure 3: Projection of Monte Carlo samples (at different temperatures) using the first **two principal components of the Ising model**. The samples were generated for system sizes $L = 20, 40, 80$ for (a) - (c). In part (c) the background color indicates the distinct phases. Figure reproduced from *L. Wang, Phys. Rev. B* **94**, *195105 (2016).*

Exercise 15: Image reconstruction using Autoencoders

In the last exercise we want to shift our focus from classification problems and instead look at the topic of **image reconstruction** using a vairational autoencoder as introduced in the lecture. As discussed there, an autoencoder consists of three elementary parts – an encoder, a latent representation, and a decoder. The encoder and decoder parts are typically symmetric in that they have the same number of hidden nodes and layers.

In the encoding part the image will be broken down and compressed into a latent representation, sometimes referred to as the code. The decoder then takes this "code" and tries to recover to original image. The example below was shown in the lecture video and can also be used in this exercise.

Figure 4: **Image reconstruction** using a variational autoencoder (VAE) in the presence of noise and obstructions.

a) We provide you with a small dataset containing various images from the STL dataset. First of all you have to import the data, which you can do using the JLD package. Next you have to write a noise function in order to create the noisy images, for this you should add **gaussian noise** on top of the images. This can be easily achieved using Julias own random function adjusted with the desired standard deviation: $\sigma \cdot randn()$
Plot the original and the noise image side by side.

b) Next step is designing the neural network. Because of the structure of the autoencoder the input and output have the same dimension, given by the size of your images. Additionally since your going to work with color images your input is a three dimensional array. In order to avoid the need to flatting the input and later reconstruct we are going to use **convolutional layers** instead of fully connected ones.

The different is, that the convolutional layers only has a small number of hidden nodes which are connected to a subsection of the input (e.g. a 5×5 section of the image). This section is shifted around the image always applying the same hidden nodes. Thereby the network is able to efficiently learn local features of the images, while at the same time reduce the numbers of needed parameters drastically. In the same fashion but reversed you can apply deconvolution layers (or transposed convolution) to restore the image. For the syntax of the convolution and deconvolution layers checkout the Flux documentation. Now create a network consistent of one convolution layer followed by a deconvolution layer. Train it against the dataset. Afterwards use the network to restore the images from the test set. Plot an example of an original, noise and restored image side by side.

c) We want to repeat this process but using some **block masking noise** instead of gaussian noise. Write a function which covers some random portions of the images using squares. This can be done changing all colour values to 0.5 in the affected areas.
Once again plot the original and the noise image side by side.

d) Create a training and test dataset using block masking noise. Start with a small blocks of size 2×2 inside the images and train the your neural network. You can reuse the network from the step before. Plot your results.
Now you can start increasing the size of the blocks in the images. Repeat the training and evaluate the performance of your network. What can your observe? What can you do in order to increase the performance of the network?