FYS-STK 4155 Project 3

Zakarias L. Hejlesen

December 2, 2019

Abstract

All code and plots relevant to the project may be found at `https://github.com/ethq/FYSSTK-Project3`.

1 Introduction

In this project we aim to determine the critical temperature T_{KT} for the phase transition in the two dimensional XY model, using both a regular neural network and a convolutional neural network. To do so, we will first generate our own dataset through Markov-Chain Monte-Carlo sampling, aiming at least for a uniform distribution of states over a temperature interval containing T_{KT}.[1] We label each state with the temperature of the system it was drawn from, as thermal fluctuations render the energy an inaccurate measure for classification. We expect the CNN will outperform the NN as it has access to the 2d information directly; if possible, we will attempt to decode what the CNN/NN learns - or hasn't learnt.

2 Formalism

Let us briefly introduce the XY model. It is governed by the Hamiltonian

$$H = -J \sum_{\langle ij \rangle} s_i \cdot s_j = -J \sum_{\langle ij \rangle} \cos(\theta_i - \theta_j) \tag{1}$$

where $\langle ij \rangle$ indicates nearest-neighbour summation and s_i denotes a planar unit spin at lattice site i. This model exhibits some remarkable properties - in particular a phase transition. This is surprising in the context of the Mermin-Wagner theorem[1], which states that continuous symmetries cannot be broken in two dimensions. However, phase transitions need not be associated with symmetry breaking, and indeed the phase transition in the XY-model is one such example. Instead it is an example of a Kosterlitz-Thouless(KT) transition, and it is characterized by the behaviour of vortices. Vortices are rotational

[1] Though we unfortunately do not sample the microcanonical ensemble directly. It is not clear at this point how this will affect my results.

structures in the spin; if we sum up the spins around a loop enclosing a vortex it must be an integer multiple of 2π. The integer multiplying 2π is usually referred to as the vortex charge.

Below the transition temperature $T = T_{\mathrm{KT}}$, vortices and anti-vortices are tightly bound in pairs. This means they influence the surrounding spin field only locally, and so one finds that the spin correlation function decays algebraically in this regime. However, at $T > T_{\mathrm{KT}}$, it becomes energetically favourable for the vortices to unbind. The free or quasi-free vortices then influence the spin field in a chaotic manner, and this leads to an exponential decay in the spin correlation function. The exact value of T_{KT} can be found through Monte-Carlo simulations, and has been determined to be $T_{\mathrm{KT}}/J = 0.893 \pm 0.002$.[2]

2.1 Convolutional neural nets

As we will see, our regular neural net does not do spectacularly well when it comes to identifying T_{KT}. The hope is that we can improve our result by using a *convolutional* neural net(CNN) instead. These offer several advantages over regular neural nets, especially when it comes to computer vision. Let us list a few:

1) Less weights to train. CNN's are tailored to image analysis and make use of the fact that pixel data is often highly locally correlated. Thus there are several 'downsampling' layers in a CNN.

2) Global feature recognition. While a NN learning to recognize a rabbit in the lower left corner of an image will not be able to recognize a rabbit in the top right, a CNN can.

3) blaha

In terms of architecture, CNN's differ from regular NN's in that they include different types of layers. Commonly, a CNN will contain:

1) **A convolutional layer.** This is the backbone of the CNN. The layer has N associated *filters*, which act as the neurons. Each filter is a matrix of dimension $l_x \times l_y$ and is convolved with the preceding input layer to produce the output. Filters act as feature extractors and are thus responsible for CNN's being able to apply knowledge learned in one part of the image to another part of the image. As an example, we mention the famous Sobel filter which is mainly used for edge detection. Typically the input layer is padded with zeros to ensure that the convolutional laye does not reduce the spatial dimensions of the data. This is advantageous when it comes to preserving information at the edges, and is mandatory in more advanced architectures(e.g. so-called residual nets) where a constant spatial dimension is demanded. Choosing the number of filters in a layer is a black art, but we note that performance saturates when we have one filter centered at each pixel(including padding). Consequently there is no point in having more than 81 filters to process a 7x7 image, for example.

2) **An activation layer.** As always, we need to introduce some kind of non-linearity for our net to be a universal function approximator. We usually apply one after each convolutional layer to increase the level of feature extraction; if we simply stack two convolution layers on top then the second will act only on a linear transformation of the first input, yielding no significant advantage over just a single layer.

3) **A pooling layer.** Similar to a convolutional layer, except now the filter is replaced with either a max or average operation. Acts as regularization and downsamples the image. Typically used after several convolutional layers have been applied.

4) **A dropout layer.** Also a type of regularization, familiar from NN's. Randomly kills off a fraction of neurons/filters during training(though only for one pass, after which they are revived). Also causes the net to become more robust; if a filter has learned to recognize flying turtles and is then killed, other filters have to step in and learn how to recognize flying turtles too.

3 Methods

3.1 Data generation

Our desired dataset consists of a set of sample states of the XY model on a square lattice of dimension L, each at fixed energy and temperature. As these are one to one only in the thermodynamic limit, we pick our representative states as those with approximately energy $E = \langle E \rangle$. Sampling the microcanonical ensemble quickly becomes unfeasible, and so we resort to sampling using a Markov-Chain Monte-Carlo(MCMC) method - the Metropolis algorithm. To generate a state at temperature T, we start with a uniformly distributed spin configuration. We then run the Metropolis algorithm(shown below) until successive energy values are within a specified tolerance, at which point we assume equilibrium has been reached. For calculating expectation values - energy in particular - we then proceed to sample the state space $M = 1000$ times, each separated by $5L^2$ Metropolis steps to minimize autocorrelations in the states.

INDICATION: (170,70) ETA = 0.0001

References

[1] N. D. Mermin and H. Wagner. Absence of ferromagnetism or antiferromagnetism in one- or two-dimensional isotropic heisenberg models. *Phys. Rev. Lett.*, 17:1133–1136, Nov 1966.

[2] Peter Olsson and Petter Minnhagen. On the helicity modulus, the critical temperature and monte carlo simulations for the two-dimensional XY-model. *Physica Scripta*, 43(2):203–209, feb 1991.

```
T = get_temperature()
set_initial_config()
E_0 = energy()
for s in spins do
    flip_spin(s)
    E_1 = compute_energy()
    ΔE = E_1 − E_0
    acc = uniform_random()
    if acc < exp(−ΔE/T) then
        Update current state, store
        C_0 = C_1
        E_0 = E_1
    end
end
```

Algorithm 1: The Metropolis algorithm. Above we illustrate a full sweep, e.g. an MCMC step applied to each spin in the system.