FYS-STK 4155 Project 3

Zakarias L. Hejlesen

December 5, 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 (AdaBoosted) regular neural nets and various convolutional nets. 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. The inspiration for this project is ref. [1], the main topic of which is whether neural nets can learn T_{KT} in the XY-model. I hope to apply what I learn to my own research, in particular whether it is possible for a neural net to learn a particular transition temperature in quantum vortex matter.[2]

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

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

[2] To be more specific, the transition in question is the Onsager transition to negative temperature states for vortex matter in superfluids, which might be e.g. Helium or Rb-87 Bose-Einstein condensates.

particular a phase transition. This is surprising in the context of the Mermin-Wagner theorem[2], 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 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} on an infinite lattice can be found through Monte-Carlo simulations, and has been determined to be $T_{\mathrm{KT}}^{\infty} = 0.893 \pm 0.002$.[3]

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[3] $l_x \times l_y$ 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

[3]This dimension is sometimes referred to as the receptive field, as it describes how much of the image a filter will see.

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.

4) **A dense layer.** These are just the fully connected layers used in regular NN's. Typically two of these are used as the last layers in a CNN. The first of these consists of a relatively high number of neurons and gathers information across the entire image, whereas the latter is the output layer and has a number of neurons equal to the number of classes.

2.1.1 Residual nets

Deep nets suffered for a long time from the problem of vanishing/exploding gradients - that weights will either vanish or blow up by repeated multiplication through a large number of layers. Although this problem is now typically dealt with by both normalized weight initialization and the introduction of normalization layers[4], it was found that even so, deep nets attain a maximal accuracy which then subsequently begins to degrade.[5] Incidentally, this loss of accuracy - which also affects the trainng data - has nothing to do with overfitting. This degradation problem was the motivation for the introduction of so-called residual nets, or ResNets.[6]

The key insight of the creators of the ResNet architecture was that identity mappings are not easy for multiple non-linear layers to approximate. Indeed,

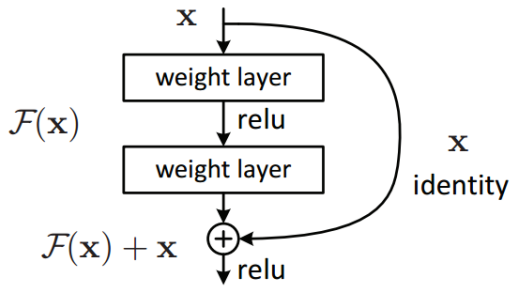

Figure 1: A residual block, figure taken from ref. [6]

experiments show that if one simply inserts several layers performing an identity mapping into a given architecture, performance degrades.

To resolve the problem, let us first consider a stack of layers whose job is to produce an approximation of a map $\mathcal{H}(x)$ capable of yielding perfect predictions. Now, if $\mathcal{H}$ is very close to the identity map, degradation is likely to occur. Hence the authors hypothesized that one may obtain better predictions by instead trying to approximate $\mathcal{F}(x) \equiv \mathcal{H}(x) - x$.[4] After our non-linear layers have approximated $\mathcal{F}(x)$, we then simply add in x to obtain an approximation of $\mathcal{H}(x)$. In total, these operations comprise a *residual block*, as shown in Fig. 1

A regular deep net can then be converted to a residual net by simply replacing stacks of convolutional layers by residual blocks. This turns out to work so well that the authors obtained first place in the 2015 ILSVRC classification competition, using an ensemble consisting of six 152-layer residual nets.

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.

[4] This is why ResNets need input/output dimensions to be equal.

```
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.

INDICATION: (170,70) ETA = 0.0001 https://arxiv.org/pdf/1207.0580.pdf Hinton paper https://machinelearningmastery.com/machine-learning-ensembles-with-r/

4 Results

We begin by reporting on the quality of our generated datasets. It's a mixed bag - while all datasets generate the expected energy and heat capacity curves(see Fig. **??**), we can determine that the statistical averages cannot be exactly correct. To do so, we refer to the fact that one may determine the scaling of T_{KT} with lattice dimension L with the help of renormalization group techniques. In particular, the scaling is given by[7]

$$T_{\mathrm{KT}}^{L} = T_{\mathrm{KT}}^{\infty} + c\frac{1}{(\log L)^2} \tag{2}$$

where c is a constant. This is unfortunately not the scaling we see, as shown in Fig. **??**. Nonetheless, each dataset *does* have a well-defined transition temperature, as may be observed by the maximum in the heat capacity. And we can still see whether our neural nets can learn this particular point if we feed them either the spin or vortex configuration.

4.1 Convolutional nets

In order to obtain the best possible fit, we have tried out several architectures.[5] We begin by looking at our results on the simplest possible CNN

References

[1] Matthew J. S. Beach, Anna Golubeva, and Roger G. Melko. Machine learning vortices at the kosterlitz-thouless transition. *Phys. Rev. B*, 97:045207, Jan 2018.

[2] 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.

[3] 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.

[4] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[5] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5353–5360, June 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[7] David R. Nelson and J. M. Kosterlitz. Universal jump in the superfluid density of two-dimensional superfluids. *Phys. Rev. Lett.*, 39:1201–1205, Nov 1977.

[5] Calling them architectures might be a little excessive; various orderings of various layers might be more appropriate!