Neural Networks and the Renormalization Group

Restricted Boltzmann Machines and RG Flow

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Physics

Author :	J. M. de Gier
Student ID :	s1619179
Supervisor :	Prof. dr. K.E. Schalm
Second corrector :	Dr. S.P. Patil

Leiden, The Netherlands, July 1, 2022

Neural Networks and the Renormalization Group

Restricted Boltzmann Machines and RG Flow

J. M. de Gier

Huygens-Kamerlingh Onnes Laboratory, Leiden University
P.O. Box 9500, 2300 RA Leiden, The Netherlands

July 1, 2022

Abstract

Neural networks have been an active field of research for years, but relatively little is understood of how they work. Specific types of Neural Networks have a layer structure with decreasing width which acts like coarse-graining, reminiscent of the renormalization group (RG). We examine the Restricted Boltzmann Machine (RBM) and discuss it's possible relation to RG. The RBM is trained on the 1D and 2D Ising model, as well as the MNIST dataset. In particular for the 2D Ising model showing a flow towards the critical point $T_c \approx 2.27$, opposite to the RG-flow. Examining the behaviour of the RBM on the MNIST dataset shows that sparse datasets can allow multiple fixed points which can be removed by artificially creating new samples. We conclude that this RBM-flow exists due to the multiple relevant length scales at the critical point and we briefly discuss why.

Contents

Chapter 1

Introduction

Machine learning is a branch of artificial intelligence that has become a cornerstone in many area's of modern science. It has been successfully used in pattern recognition in huge data sets by both scientists and corporations (most notably Google, Facebook & Amazon). While the "simpler" uses of machine learning are reasonably well understood, many of the more intricate neural networks are essentially black boxes. As such, machine learning still is an active field of research with many open questions.

Training neural networks generally fall into two categories, supervised and unsupervised training. In supervised training we train the system using samples and labels, asking the system to guess the correct label given a sample. Examples include recognising digits, is this a picture of a cat, and how many cards are there in this picture.

On the other hand, in unsupervised learning we train the system using unlabelled samples and ask the network to reproduce the sample as close to as possible. Examples of this would be the text generator GPT-3 by OpenAI and face-generators[†].

From a physics point of view a specific class of neural networks can be examined through existing principles (such as the renormalization group and random matrix theory). In these neural networks information is passed through layers which act as a discrete time axis. Understanding how these neural networks function can be done if we understand how information is preserved and lost as it travels through the layers.

In particular when subsequent layers are decreasing in size it must be that some information is lost as the information is captured in fewer neurons. As the layers get smaller and smaller more and more information is lost. But if the network is

[†]See https://openai.com/ and https://this-person-does-not-exist.com/ respectively.

still effective it should thus still recognise the cat or generate a human face.

In turn this must mean that the information lost is irrelevant to the purpose of the neural network and the shrinking width (size of the layers) of the network has the effect of removing the irrelevant information from the system, keeping only the relevant features. Of course when the system becomes too narrow it loses it's effectiveness as even the relevant information is lost.

This loss of irrelevant information through the shrinking of the system is akin the action of the renormalization group in physics where we coarse-grain the system to remove short range effects to examine the long range behaviour of the system.

The renormalization group is predominantly focused on finding and exploring critical points in a physical system. Near these points the correlation length diverges and the bulk behaviour is dominated by long range modes. The renormalization group then allows the examination of these long range modes through the so called renormalization scheme.

If neural networks can truly be related to the renormalization group it would provide a powerful tool to understand how these types of neural networks truly work. In turn, this could be used to shorten training times and computational costs for neural networks and allow even larger and more powerful neural networks to be used.

In this thesis we will examine one type of relationship between neural networks and the renormalization group. We will start by examining neural networks and discuss two different types of neural networks in depth. We will then move on to briefly discuss the renormalization group and it's application to the one-dimensional and two-dimensional Ising model.

Finally we train a generative neural network on three types of datasets: the one- and two-dimensional Ising model and the MNIST[*] dataset. This generative network gives us an RBM-flow of samples which we can compare to the RG-flow. We observe for the 2D-Ising model that the RBM-flow is towards the critical point (opposite to the RG-flow). For the MNIST dataset we observe multiple fixed distributions we can be eliminated by artificially extending the dataset.

We learn that the flow is towards the critical point due to the various length scales present there and gain a better understanding how this type of neural networks functions.

[*]The MNIST dataset is a collection of handwritten digits.

Chapter 2

Neural Networks: a Broad Review

2.1 What is a Neural Network and the Brain

The brain has been a subject of research since we discovered its function. It receives information from our senses, processes this, and then sends signals to our body. It houses our consciousness, hopes and dreams; it processes immense amounts of information and yet it only consists of many relatively simple interconnected units. These neurons, consisting of dendrites which receive signals, a cell body, and the axon which outputs a signal, are as simple as can be.[1][11][13]

A single neuron receives signals from other neurons connected to its cell body. If these signals exceed an internal threshold the neuron sends signals to other neurons to which its axon connects.

We can make a simplified model of a neuron's output y given inputs $\boldsymbol{x}$ as:

$$z = \sum_j w_j x_j + b \tag{2.1.1a}$$

$$y = f(z) \tag{2.1.1b}$$

where w contains information about the type and strength of the connection, b is the threshold, and f is some function which determines what the output of the neuron exactly is.

In a neural network the input of a neuron is usually the output of other neurons, which gives us:

$$z_i = \sum_j w_{ij} a_j + b_i \tag{2.1.2a}$$

$$a_i = f_i(z_i) \tag{2.1.2b}$$

Having taken inspiration from a real neural network, we can now define the effect of a neuron according to eq. (2.1.2), where w is the weight matrix, b is a bias and f_i is the activation function. Fig.2.1 shows common choices for deterministic activation functions.

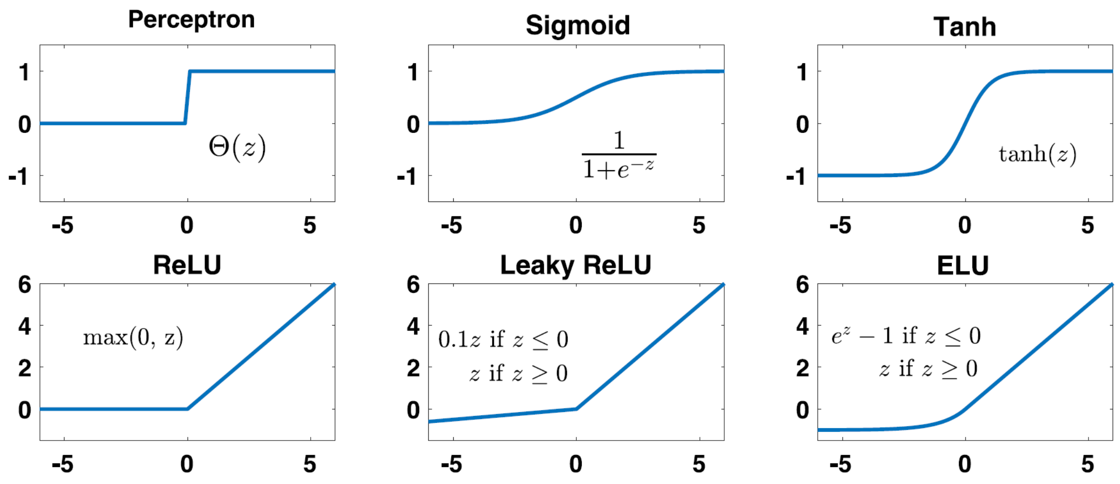


Figure 2.1: *Possible non-linear activation functions for neurons. In modern neural networks, it has become common to use non-linear functions that do no saturate for large inputs (bottom row) rather than saturation functions (top row). Image source: Mehta, et al.[12]*

An example of a non-deterministic activation function would a neuron which has output 1 with probability $p(a = 1) = \sigma(z) = \frac{1}{1+e^{-z}}$ and output -1 otherwise. We will use this activation function when we discuss the Binary Restricted Boltzmann Machine later on.

2.2 Learning a single neuron

We have a single neuron that maps some input $\boldsymbol{x}$ to some output $\hat{y}$. It does this according to the following rule:

$$\hat{y} = f\left(\boldsymbol{w} \cdot \boldsymbol{x} + b\right) = f(z) \tag{2.2.1}$$

We want to train our neuron and ensure the the output $\hat{y}$ is as close to a target y which is associated to the data. The "Closeness" of $\hat{y}$ and y is captured in the so called cost (or error) function C. The form of the cost function is closely related to the activation function (See Appendix A). Training the neuron is then equivalent to minimising this cost function with respect to the parameters (w and b). Let us examine a simple example.

2.2.1 A single neuron as a classifier

We have a dataset $\mathcal{D}$ which contains a set of N samples $\boldsymbol{x}^{(n)}$ which fall in two classes $y = 1$ and $y = 0$ (for example: the ordered versus disordered state of the Ising model). We want to identify the output of our neuron as the probability that $y = 1$: $\hat{y} = p(y = 1|\boldsymbol{x}; \boldsymbol{w}, b)$ and we thus use the sigmoid function which has a continuous output between 0 and 1:

$$f(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2.2}$$

The corresponding Cost function (see Appendix A) is the Cross-Entropy:

$$\begin{aligned} C(\boldsymbol{x}, y) &= \sum_{n=1}^{N} C\left(\boldsymbol{x}^{(n)}, y^{(n)}\right) \\ &= \sum_{n=1}^{N} \left[-y^{(n)} \ln\left(\hat{y}(\boldsymbol{x}^{(n)})\right) - (1 - y^{(n)}) \ln\left(1 - \hat{y}(\boldsymbol{x}^{(n)})\right)\right] \end{aligned} \tag{2.2.3}$$

Minimising the Cost function also minimises the distance between $\hat{y}^{(n)}$ and $y^{(n)}$. Let us thus calculate the first derivative of the cost function with respect to an arbitrary parameter θ:

$$\frac{\partial C(\boldsymbol{x})}{\partial \theta} = \sum_{n=1}^{N} \left[-\frac{y^{(n)}}{\hat{y}^{(n)}} + \frac{1 - y^{(n)}}{1 - \hat{y}^{(n)}}\right] \frac{\partial \hat{y}^{(n)}}{\partial \theta} \tag{2.2.4}$$

Using that $\hat{y} = f(z(\theta))$ we can write the second part as:

$$\frac{\partial \hat{y}^{i}}{\partial \theta} = \frac{\partial f(z^{(n)})}{\partial z^{(n)}} \frac{\partial z^{(n)}}{\partial \theta} \tag{2.2.5}$$

Using the fact that the derivative of the sigmoid function is: $\sigma'(z) = \sigma(z)(1-\sigma(z))$ we find:

$$\frac{\partial f(z^{(n)})}{\partial z^{(n)}} = f(z^{(n)})\left(1 - f(z^{(n)})\right) = \hat{y}^{(n)}\left(1 - \hat{y}^{(n)}\right) \tag{2.2.6}$$

Combining this with the original equation we find

$$\frac{\partial C(\boldsymbol{x})}{\partial \theta} = \sum_{n=1}^{N} \left[\hat{y}^{(n)} - y^{(n)}\right] \frac{\partial z^{(n)}}{\partial \theta} \tag{2.2.7}$$

We see that the change in the cost function depends linearly on the difference $\hat{\boldsymbol{y}} - \boldsymbol{y}$ multiplied with $\frac{\partial z^{(n)}}{\partial \theta}$. Now since z depends linearly on its parameters we finally find:

$$\frac{\partial C(\boldsymbol{x})}{\partial w_i} = \sum_{n=1}^{N} \left[\hat{y}^{(n)} - y^{(n)}\right] x_i^{(n)} \tag{2.2.8a}$$

$$\frac{\partial C(\boldsymbol{x})}{\partial b} = \sum_{n=1}^{N} \left[\hat{y}^{(n)} - y^{(n)}\right] \tag{2.2.8b}$$

Because of this simple relation we can now use gradient descent to update the parameters:

$$\theta^{t+1} = \theta^t - \eta \frac{\partial C(\boldsymbol{x})}{\partial \theta^t} \tag{2.2.9}$$

η is the so called learning rate. Choosing the correct learning rate is important, since a big learning rate will converge quickly but can introduce numerical instability. A small learning rate on the other hand will be accurate, but the process can be extremely long.

A different combination of activation function and cost function would lead to a different expression for equations (2.2.8). For most simple activation functions we can choose the cost function such that equation (2.2.7) holds (see Appendix A).

2.2.2 Regularisation

After sufficient updates the cost function will have been sufficiently minimised and the neuron will act as a (rough) classifier over the dataset. Sadly in the real world our datasets are finite and imperfect, and thus by minimising the cost function we will not necessarily have optimised the performance of the neuron on new data. To achieve this we use various regularisation methods.

In and out error, and early termination

Instead of optimising the performance of the neuron over the dataset $\mathcal{D}$ we can instead split the dataset into two: the training dataset $\mathcal{D}_{\text{train}}$, and the test dataset $\mathcal{D}_{\text{test}}$. We can then train the neuron over $\mathcal{D}_{\text{train}}$ and keep track of C_{train} and C_{test}, stopping the optimisation of the neuron when C_{test} starts to increase (as this is a sign of over-fitting)*. Figure 2.2 shows how the test energy reaches a minimum while the training energy is still decreasing.

*We are directly minimising C_{train} and indirectly minimising C_{test}, as such any experiments will have to be done on a third independent dataset.

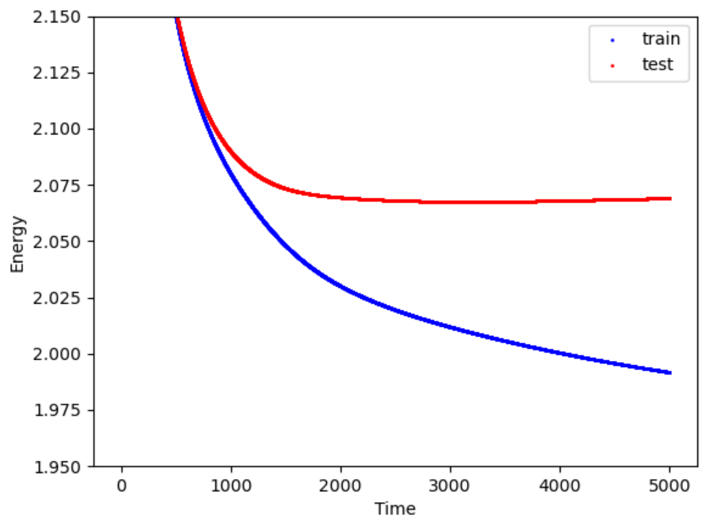


Figure 2.2: *Training versus test error. We see that after sufficient training the training error is still decreasing but the test error stops decreasing (and will actually eventually increase). This is a sign that the network is overfitting and we stop the training.*

Regularisation energy and the L_n norm

In many cases we have some additional knowledge about the weights and bias themselves, usually we expect many of them to be quite small, and we can encode this knowledge in a regularisation term. Instead of minimising the cost function directly, we minimise the energy E consisting of both the cost function and this regularisation term:

$$E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}, \boldsymbol{b}) = C(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}, \boldsymbol{b}) + R(\boldsymbol{w}, \boldsymbol{b}) \tag{2.2.10}$$

The regularisation term R can often be cast as a distance between the current parameters $\boldsymbol{\theta}$ and some "expected" parameters $\tilde{\boldsymbol{\theta}}$ (note that these "expected" parameters encode our knowledge about them, such as around which values we expect them to be centred around). One option to measure this distance is using an L_n norm, which is defined as:

$$L_n(\boldsymbol{x}) \equiv ||\boldsymbol{x}||_n = \left(\sum_i |x_i|^n\right)^{\frac{1}{n}} \tag{2.2.11}$$

Usually we take $\tilde{\boldsymbol{\theta}} = 0$ as we often have knowledge about the distribution of parameter sizes, but have no idea where they are centred around. The following norms are of particular interest:

Taxicab Norm L_1 is used when we expect many parameters to be zero, as it is beneficial decrease parameters at the cost of increasing others. This is also called LASSO regression.

Euclidean Norm L_2 is used when we expect many parameters to be small, as it is beneficial to decrease a large parameter slightly, even if that increases other (much smaller) parameters. This is also called Ridge regression.

Maximum Norm $L_\infty = \max(x_i)$ specifically punishes the largest parameter.

Zero Norm L_0 specifically punishes equal to the number of non-zero parameters (and not their size).

of these the taxicab and euclidean norm are the most common and as such we usually see:

$$R(\boldsymbol{w}, \boldsymbol{b}) = \lambda_w ||\boldsymbol{w}||_1 + \lambda_b ||\boldsymbol{b}||_1 + \frac{\mu_w}{2} ||\boldsymbol{w}||_2^2 + \frac{\mu_b}{2} ||\boldsymbol{b}||_2^2 \tag{2.2.12}$$

Minimizing the energy, which now includes the regularisation term, gives the following update rule:

$$\theta^{t+1} = \theta^t - \eta \frac{\partial E(\boldsymbol{x})}{\partial \theta^t}$$

$$\theta^{t+1} = \theta^t - \eta \left(\frac{\partial C(\boldsymbol{x})}{\partial \theta^t} + \lambda_\theta \operatorname{sgn}(\theta^t) + \mu_\theta \theta^t \right) \tag{2.2.13}$$

Due to the balancing act between the cost function and the regularisation function, choosing the right values for λ and μ is important. If they are too large the neuron won't be accurate and if they are too small it won't have much of an effect.

2.2.3 Optimisation Methods

Other than gradient descent there are other methods which minimise the cost function, each with their own (dis)advantages. We will discuss a few of them here.

Full Gradient Descent

As discussed in the previous section, in gradient descent we calculate the gradient of the energy for every sample in the dataset. We then update the parameters according to the following update rule:

$$\Delta\theta^t = -\eta \frac{\partial E}{\partial \theta^t} \tag{2.2.14a}$$

$$\theta^{t+1} = \theta^t + \Delta\theta^t \tag{2.2.14b}$$

where η is the learning rate.

Stochastic Gradient Descent

In Stochastic Gradient Descent we approximate the gradient using a single sample. This allows us to sweep through the dataset, updating the parameters for every sample in the following fashion:

1. Shuffle training dataset.
2. For every sample calculate the energy and update the parameters.
3. Repeat until minimum is reached.

Importantly, we still sweep through the entire dataset before we encounter the same sample again. This method is usually also combined with a decaying learning rate.

Batching

Instead of using the entire training dataset at every timestep or updating at every sample, we can make a good approximation of the gradient using a so called "micro-batch":

$$C(\boldsymbol{x}) = \sum_{n=1}^{N} C(\boldsymbol{x}^{(n)}) \approx \frac{N}{M} \sum_{n \in \mathcal{B}_M} C(\boldsymbol{x}^{(n)}) \tag{2.2.15}$$

where $\mathcal{B}_M$ is a micro-batch containing M random samples*. As M increases this approximation becomes highly accurate. In contrast with Stochastic Gradient Descent we can encounter the same sample multiple times before we have seen the entire dataset.

Momentum

By including a "momentum" term we update the parameters using a linear combination of the gradient and the previous update:

$$\Delta\theta^t = \alpha \Delta\theta^{t-1} - \eta \frac{\partial E}{\partial \theta^t} \tag{2.2.16}$$

*These micro batches can be made with and without repeating samples.

ADAM

For ADAM (Adaptive Moment Estimation) we keep track of two momentum terms: m and v

$$m_\theta^t = \beta_1 m_\theta^{t-1} + (1-\beta_1)\frac{\partial E}{\partial \theta^t} \tag{2.2.17a}$$

$$v_\theta^t = \beta_2 v_\theta^{t-1} + (1-\beta_2)\left(\frac{\partial E}{\partial \theta^t}\right)^2 \tag{2.2.17b}$$

$$\hat{m}_\theta = \frac{m_\theta^t}{1-\beta_1} \tag{2.2.17c}$$

$$\hat{v}_\theta = \frac{v_\theta^t}{1-\beta_2} \tag{2.2.17d}$$

$$\theta^{t+1} = \theta^t - \eta\frac{\hat{m}_\theta}{\sqrt{\hat{v}_\theta}+\varepsilon} \tag{2.2.17e}$$

β_1 and β_2 control the "mass" of the momenta (usually 0.99 and 0.999 respectively) and ε a really small factor to prevent division by zero, $\varepsilon \sim 10^{-8}$. ADAM has several advantages and superior convergence (of the methods shown here)[8].

2.3 Many neurons, Feedforward Networks, and Multilayer Perceptrons

Having understood the workings of a single neuron, we can now link together many neurons into a neural network, which interact according to eq. (2.1.2):

$$z_i = \sum_j w_{ij} a_j + b_i \tag{2.1.2a}$$

$$a_i = f_i(z_i) \tag{2.1.2b}$$

The parameters over which we minimise ($\boldsymbol{w}$ and $\boldsymbol{b}$) are an $N \times N$ matrix and an N dimensional vector. This gives our system essentially $N(N+1)$ free parameters and every neuron we add to the system adds $2(N_{\text{old}}+1)$ new parameters to the system. As such even moderately sized neural networks have an enormous amount of parameters to fit the system. It is thus quite reasonable that we want to add additional neurons to a network.

We can separate the neurons into three functions:

Input The output of these neurons are set to specific values for training: a_i^{Input}. The output of these neurons can still change, if it depends on other neurons.

Output The output of these neurons are what we are interested in: a_i^{Output}.

Hidden We have no interaction with these neurons and they are here as free parameters: a_i^{Hidden}.

be aware that a neuron can be both an input and output neuron.

2.3.1 How does a neural network update and learn

We have build a neural network, consisting of input, output, and hidden neurons; all connected by weights and each with a bias. How do we now proceed?

We start with setting the input neurons equal to some initial values: $a_i^{\text{Input}} = x_i$. We then choose a neuron and update it according to equation (2.1.2), repeating this a "sufficient" number of times. Finally we get the output from the output neurons.

To learn the neural network we change a single parameter and repeat this. We can then compute the change in energy and keep the new parameter depending on this energy difference.

This method is extremely tedious due to the fact that all neurons are fully connected with each other. Changing the parameters of a single neuron thus impacts every other neuron. By restricting which neurons connect to each other we can circumvent this problem allowing for better methods of updating and learning the network. Good network architecture is crucial for powerful and fast neural networks.

We will discuss two types of neural network architecture: Feed-forward networks and the Boltzmann Machine.

2.3.2 Feed-forward networks and the Multilayer Perceptron

Feed-forward neural networks have the property that they can be organised in layers, where there is no interaction inside a layer and there is only interactions between layer l and $l-1$. This means we can write the activity of a neuron i as:

$$a_i^l = f_i^l\left(\sum_j w_{ij}^l a_j^{l-1} + b_i^l\right) \tag{2.3.1}$$

This allows us to update every neuron in a single layer simultaneously. Because a neuron only depends on the neurons in the layers before it, it cannot influence itself. As such information can only propagate through the network in a single direction (from the input layer, through the hidden layers, to the output layer).

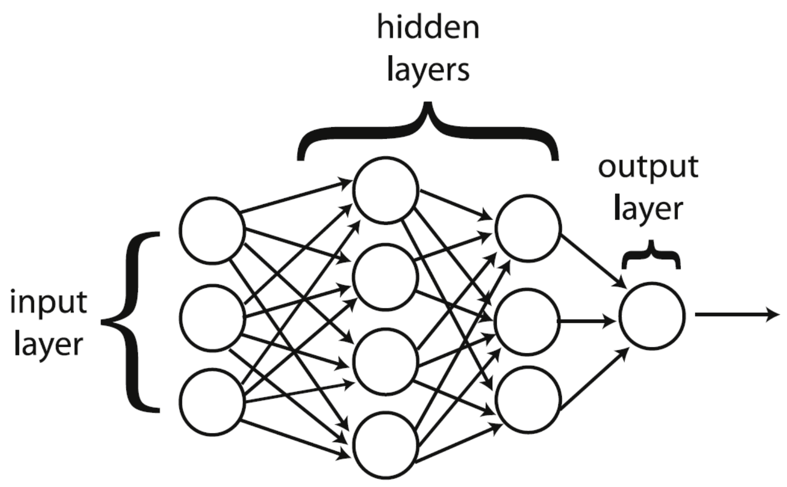


Figure 2.3: *Visualisation of the architecture of a Feed-forward Deep neural network. Image source: Mehta, et al.[12]*

As such starting from the input layer, we only need to update a neuron once massively reducing the computations.

Furthermore this restriction allows us to use a powerful tool: The backpropagation algorithm, which we derive here. This allows us to perform gradient descent on all weights and biases simultaneously, which significantly improves training speed.

The Backpropagation Algorithm

We have a neural network consisting of an input layer, $L-2$ hidden layers and an output layer, for a total of L layers. The activation of a single neuron is defined to be:

$$a_i^l = f_i^l(z_i^l) \tag{2.3.2a}$$

$$z_i^l = b_i^l + \sum_j w_{ij}^l a_j^{l-1} \tag{2.3.2b}$$

with f_i^l the activation functions. We also have an Energy which consists of the cost function C which measures the accuracy of the neural network and a regularisation function R which only depends on the weights and biases. $E = C + R$

Let us now find the change in the cost function with respect to z_i^l:

$$\Delta_i^l = \frac{\partial C}{\partial z_i^l} = \frac{\partial C}{\partial a_i^l} f'^l_i(z_i^l) \tag{I}$$

Since z_i^l linearly depends of b_i^l we see that:

$$\Delta_i^l = \frac{\partial C}{\partial z_i^l} = \frac{\partial C}{\partial b_i^l} \tag{II}$$

The strength of the backpropagation algorithm comes from the fact that the error of layer l only contributes through the activation of the subsequent layer $l+1$.

$$\Delta_i^l = \frac{\partial C}{\partial z_i^l} = \sum_j \left(\frac{\partial C}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{z_i^l} \right) \tag{2.3.3}$$

$$= \sum_j \left(\Delta_j^{l+1} \frac{\partial z_j^{l+1}}{z_i^l} \right) \tag{2.3.4}$$

$$\frac{\partial z_j^{l+1}}{z_i^l} = w_{ji}^{l+1} {f'}_i^l(z_i^l) \tag{2.3.5}$$

$$\Delta_i^l = {f'}_i^l(z_i^l) \sum_j \left(\Delta_j^{l+1} w_{ji}^{l+1} \right) \tag{III}$$

Finally we can find the error with respect to the weights as:

$$\frac{\partial C}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l} = \Delta_i^l a_j^{l-1} \tag{IV}$$

This leads to the following algorithm:

1. **Activation at input layer:** Set the activations a_i^1 of all the neurons in the input layer.

2. **Feedforward:** starting with the first layer, use Eqs. (2.3.2) to compute z^l and a^l for the subsequent layers.

3. **Error at top layer:** Calculate the error of the top layer Δ_i^L using Eq. (I). This requires expressions for the derivative of the cost function C and the activation functions f_i^L.

4. **"Backpropagate" the error**: Use Eq. (III) to propagate the error backwards and calculate Δ_i^l for all layers. This requires expressions for the derivative of the activation functions f_i^l

5. **Calculate gradient:** Use Eqs. (II) and (IV) to calculate $\frac{\partial C}{b_i^l}$ and $\frac{\partial C}{w_{ij}^l}$.

So in total the derivative of the energy is found as:

$$\frac{\partial E}{\partial b_i^l} = \frac{\partial C}{\partial b_i^l} + \frac{\partial R}{\partial b_i^l} = \Delta_i^l + \frac{\partial R}{\partial b_i^l} \tag{2.3.6a}$$

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial C}{\partial w_{ij}^l} + \frac{\partial R}{\partial w_{ij}^l} = \Delta_i^l a_j^{l-1} + \frac{\partial R}{\partial w_{ij}^l} \tag{2.3.6b}$$

with R the regularisation. Updates are done according to the optimisation method (discussed in section 2.2.3)

Multilayer Perceptrons

A multilayer perceptron (MLP) is a discriminative feed-forward neural network. It tries to fit the samples to a given output. The performance of the MLP depends on the choice of cost function, activation functions, number of layers and neurons per layer. As such one should choose these carefully.

The output of a Multilayer Perceptron can sometimes be interpreted as a probability, the probability that sample $\boldsymbol{x}$ has label $\boldsymbol{y}$. The final layer then consists of a single neuron with a sigmoid activation function $\sigma(z) = (1 + e^{-z})^{-1}$ (in the case of a binary label) or multiple neurons with a SoftMax activation function.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.3.7}$$

The corresponding Cost function is then the so-called CrossEntropy (see Appendix A)

$$C = -\sum_m y_m \log(\hat{y}_m) + (1 - y_m) \log(1 - \hat{y}_m) \tag{2.3.8}$$

where

$$\hat{y}_m = p(y_m = 1|\boldsymbol{x}; \boldsymbol{w}) = \sigma(z_m) = \frac{e^{z_m}}{\sum_{k=1}^{M} e^{z_k}} \tag{2.3.9a}$$

$$y_m = \begin{cases} 1 & y = m \\ 0 & \text{else} \end{cases} \tag{2.3.9b}$$

2.4 Generative Neural Networks and (Restricted) Boltzmann Machines

A generative neural network tries to generate new samples given a dataset. Our samples $\boldsymbol{x}^{(n)} \in \mathcal{D}$ are drawn according to some probability distribution q and the goal of the generative model is to ensure that new samples are generated according to a probability distribution $p \approx q$. Instead of constructing a cost function which tries to minimise the "distrance" between the output and a given value, a generative network tries to minimise the "distance" between these probability distributions given by $C(p||q)$. We often use the Kullback-Leibler Divergence:

$$KL(q||p) = \sum_i q_i \ln\left(\frac{q_i}{p_i}\right) \tag{2.4.1}$$

Generative models come in many different forms, including Boltzmann machines and Variational Autoencoders. In this work we only use Boltzmann Machines and thus we will discuss those in detail.

2.4.1 Boltzmann Machines

A Boltzmann machine is a stochastic[*] generative model where the current state of the network is given by a Hamiltonian:

$$p(\{v_i\},\{h_a\};\{\theta_k\}) = \frac{1}{Z} e^{-H(\{v_i\},\{h_a\};\{\theta_k\})} \tag{2.4.2a}$$

$$Z = \sum_{\{v_i\},\{h_a\}} e^{-H(\{v_i\},\{h_a\};\{\theta_k\})} \tag{2.4.2b}$$

where the visible units v_i are our samples (and thus are both input and output neurons) and the hidden units h_a are free parameters. As always, the partition function Z has a sum over all configurations.

We can capture the "distance" between the probability distribution of the samples $\boldsymbol{x}^{(n)} \in \mathcal{D} \sim q$ and the probability distribution p using the Kullback-Leibler (KL) divergence:

$$\mathrm{KL}(q||p) = \sum_{\{v_i\},\{h_a\}} q(\{v_i\}) \log\left(\frac{q(\{v_i\})}{p(\{v_i\},\{h_a\};\{\theta_k\})}\right) \tag{2.4.3}$$

Minimising the cost function with respect to our parameters (we will drop the $\{v_i\}, \{h_a\}$ and $\{\theta_k\}$ for readability):

$$\frac{\partial \mathrm{KL}(q||p)}{\partial \theta} = -\sum \frac{q}{p}\frac{\partial p}{\partial \theta} \tag{2.4.4}$$

The derivative of p with respect to the parameters is simple due to eq. (2.4.2)

$$\frac{\partial p}{\partial \theta} = \frac{\partial}{\partial \theta}\frac{e^{-H}}{Z} = -\frac{e^{-H}}{Z}\frac{\partial H}{\partial \theta} - \frac{e^{-H}}{Z^2}\frac{\partial Z}{\partial \theta} = -p\left(\frac{\partial H}{\partial \theta} + \frac{1}{Z}\frac{\partial Z}{\partial \theta}\right) \tag{2.4.5}$$

We can now further expand the second term as

$$\frac{1}{Z}\frac{\partial}{\partial \theta}Z = \frac{1}{Z}\frac{\partial}{\partial \theta}\sum e^{-H} = -\sum \frac{e^{-H}}{Z}\frac{\partial H}{\partial \theta} = -\sum \frac{\partial H}{\partial \theta} p \tag{2.4.6}$$

This final term is a weighted sum over the probability distribution p. Since we can write $\langle f \rangle = \sum_i f_i p_i$ we also see that:

$$\frac{1}{Z}\frac{\partial Z}{\partial \theta} = -\left\langle \frac{\partial H}{\partial \theta} \right\rangle_p \tag{2.4.7}$$

[*]The stochastic nature of the neurons is encoded in a non-deterministic activation function, which will be made explicit when we discuss the binary restricted Boltzmann machine in section 2.4.2

Plugging this back into eq. (2.4.5) we find

$$\frac{\partial p}{\partial \theta} = p\left(\left\langle \frac{\partial H}{\partial \theta} \right\rangle_p - \frac{\partial H}{\partial \theta}\right) \tag{2.4.8}$$

Finally plugging this into eq. (2.4.4)

$$\frac{\partial \mathrm{KL}(q||p)}{\partial \theta} = -\sum q\left(\left\langle \frac{\partial H}{\partial \theta} \right\rangle_p - \frac{\partial H}{\partial \theta}\right) \tag{2.4.9}$$

and using that $\sum_i q_i = 1$ we find

$$\frac{\partial \mathrm{KL}(q||p)}{\partial \theta} = \left\langle \frac{\partial H}{\partial \theta} \right\rangle_q - \left\langle \frac{\partial H}{\partial \theta} \right\rangle_p \tag{2.4.10}$$

Here $\langle \dots \rangle_p$ averages over the model, while $\langle \dots \rangle_q$ averages over the data, so we can write this as:

$$\frac{\partial \mathrm{KL}(q||p)}{\partial \theta} = \left\langle \frac{\partial H}{\partial \theta} \right\rangle_{\text{data}} - \left\langle \frac{\partial H}{\partial \theta} \right\rangle_{\text{model}} \tag{2.4.11}$$

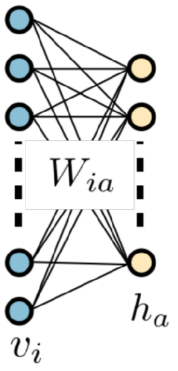
Updating the parameters is then done using an optimisation method (discussed in section 2.2.3).

Having found the derivative of the KL divergence, the question remains is: How do we actually find these averages? The average over the data only enforces the probability distribution over the visible units (since the data is only concerned with the visible units). As such we "clamp" the visible units to a sample, and freely update the hidden units. After the system has reached an equilibrium (remember that the system is stochastic in nature), we then start sampling $\frac{\partial H}{\partial \theta}$ to find the average over the data. The average over the model lets all units update freely, again sampling after the system has reached equilibrium.

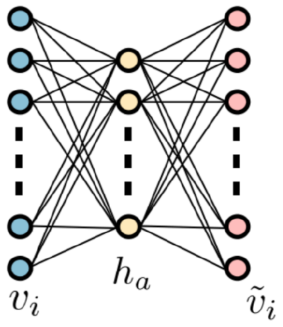
Just like for feed-forward neural networks, we would like to update all parameters at the same time. Previously we did this by creating layers of neurons which are not connected among themselves. By restricting the Hamiltonian of the Boltzmann machine we can achieve an equivalent result.

2.4.2 Restricted Boltzmann Machine

In a Restricted Boltzmann Machine (RBM) we have made the following restrictions: there are no connections between visible units, there are no connections



(a) *Visualisation of the architecture of an RBM*



(b) *Visualisation of the construction of a new configuration $\tilde{v}_i$ from a configuration v_i.*

Figure 2.4: *Visualisation of the architecture of a Restricted Boltzmann Machine. Image source: Iso, et al.[7]*

between hidden units. This gives the following Hamiltonian:

$$H(\{v_i\},\{h_a\}) = -\left(\sum_{i,a} v_i w_{ia} h_a + \sum_i g_i^{(v)}(v_i) + \sum_a g_a^{(h)}(h_a)\right) \tag{2.4.12}$$

where the function g usually has the form:

$$g_i(a_i) = b_i a_i \qquad \text{if } a_i \text{ binary} \tag{2.4.13a}$$
$$g_i(a_i) = \frac{a_i^2}{2\sigma_i^2} \qquad \text{if } a_i \text{ continuous} \tag{2.4.13b}$$

Because we have arranged the units in two intra-connected but not inter-connected layers, we can update all the visible units at the same time, and all hidden units at the same time. Since it now holds that:

$$p(\{h_a\}|\{v_i\}) = \prod_a p(h_a|\{v_i\}) \tag{2.4.14a}$$
$$p(\{v_i\}|\{h_a\}) = \prod_i p(v_i|\{h_a\}) \tag{2.4.14b}$$

which will also allow us to calculate the expected value of these neurons as:

$$\langle h_a\rangle_{\{v_i\}} = \sum_{h_a} h_a p(h_a|\{v_i\}) \tag{2.4.15a}$$
$$\langle v_i\rangle_{\{h_a\}} = \sum_{v_i} v_i p(v_i|\{h_a\}) \tag{2.4.15b}$$

finding explicit expressions for this will allow us to not only update all neurons in a layer, but by explicitly calculate the average value of a layer essentially skip the need to let the system equilibrate. This will increase the training speed of the RBM tremendously.

We will further restrict ourselves to the so-called binary RBM and show the strength of this neural network.

Binary Restricted Boltzmann Machine

For the binary restricted Boltzmann machine the Hamiltonian is given by:

$$H(\{v_i\},\{h_a\}) = -\left(\sum_{i,a} v_i w_{ia} h_a + \sum_i b_i^{(v)} v_i + \sum_a b_a^{(h)} h_a\right) \tag{2.4.16}$$

which in turn gives the following explicit form for the derivatives:

$$\frac{\partial \mathrm{KL}(q||p)}{\partial w_{ia}} = \langle v_i h_a \rangle_{\text{model}} - \langle v_i h_a \rangle_{\text{data}} \tag{2.4.17a}$$

$$\frac{\partial \mathrm{KL}(q||p)}{\partial b_i^{(v)}} = \langle v_i \rangle_{\text{model}} - \langle v_i \rangle_{\text{data}} \tag{2.4.17b}$$

$$\frac{\partial \mathrm{KL}(q||p)}{\partial b_a^{(h)}} = \langle h_a \rangle_{\text{model}} - \langle h_a \rangle_{\text{data}} \tag{2.4.17c}$$

The neurons can take on only two values $\{\alpha, \gamma\}$ (where we take w.l.o.g. $\gamma > \alpha$). We see that:

$$\frac{p(a_i = \alpha)}{p(a_i = \gamma)} = e^{-(H(a_i=\alpha)-H(a_i=\gamma))} = e^{-\Delta H_i} \tag{2.4.18}$$

Using that $p(a_i = \gamma) + p(a_i = \alpha) = 1$ we see that the probability that the neuron takes one of these values is given by the sigmoid function:

$$p(a_i = \gamma) = \sigma(\Delta H_i) = \left(1 + e^{-\Delta H_i}\right)^{-1} \tag{2.4.19a}$$

$$p(a_i = \alpha) = 1 - p(a_i = \gamma) = \left(1 + e^{\Delta H_i}\right)^{-1} \tag{2.4.19b}$$

For the visible and hidden units this becomes:

$$\Delta H_i = H(v_i = \alpha) - H(v_i = \gamma) = (\gamma - \alpha)\left(\sum_a w_{ia} h_a + b_i^{(v)}\right) \tag{2.4.20a}$$

$$\Delta H_a = H(h_a = \alpha) - H(h_a = \gamma) = (\gamma - \alpha)\left(\sum_i v_i w_{ia} + b_a^{(h)}\right) \tag{2.4.20b}$$

Now introducing $\delta = \gamma - \alpha$ and reusing our original definition of z_i (see eq. (2.1.2a)) we see that we can combine all of this as:

$$p(a_i = \gamma) = \sigma(\delta z_i) = \left(1 + e^{-\delta z_i}\right)^{-1} \tag{2.4.21a}$$

$$p(a_i = \alpha) = 1 - p(a_i = \gamma) = \left(1 + e^{\delta z_i}\right)^{-1} \tag{2.4.21b}$$

The expected values of the neurons can now be explicitly found (using eq. (2.4.15)) as:

$$\langle a_i \rangle = \frac{\gamma}{1 + e^{-\delta z_i}} + \frac{\alpha}{1 + e^{\delta z_i}} = \frac{\gamma e^{\delta z_i/2} + \alpha e^{-\delta z_i/2}}{e^{\delta z_i/2} + e^{-\delta z_i/2}} \tag{2.4.22}$$

There are two options for $\{\alpha, \gamma\}$ that are of particular interest:

$$\langle a_i(\alpha = 0, \gamma = 1) \rangle = \frac{1}{1 + e^{-z_i}} = \sigma(z_i) \tag{2.4.23a}$$

$$\langle a_i(\alpha = -1, \gamma = 1) \rangle = \frac{e^{z_i} - e^{-z_i}}{e^{z_i} + e^{-z_i}} = \tanh(z_i) \tag{2.4.23b}$$

Finding the Gradient of the Energy

We have found an explicit expression for the expected values of the hidden units given the visible units (and vice versa). This allows us to directly sample the conditional probability distributions, even though we still can't directly sample the full joint probability distribution. Because we have clamped the visible units to a sample $\boldsymbol{x}^{(n)}$, we see that:

$$\langle v_i h_a \rangle_{\text{data}} = x_i^{(n)} \langle h_a \rangle_{\boldsymbol{x}^{(n)}} \qquad \langle v_i \rangle_{\text{data}} = x_i^{(n)} \qquad \langle h_a \rangle_{\text{data}} = \langle h_a \rangle_{\boldsymbol{x}^{(n)}} \tag{2.4.24}$$

The next step is to find $\langle \dots \rangle_{\text{model}}$, which depends on the full joint probability distribution. Luckily we can use the conditional probabilities to get a good approximation of the full distribution using Gibbs sampling.

In Gibbs sampling we start with a given configuration of visible units $\{v_i\}$. From this configuration e can directly find $\{h_a\}$, which in turn allows us to find $\{v_i\}$. By repeating this, the Markov Chain will eventually reach a stationary point and sampling from this is equivalent to sampling from the joint distribution directly.

We can even go a step further using contrastive divergence (CD). Instead of finding the stationary point, we approximate the gradient using only the first k steps of the Markov chain:

$$\langle v_i h_a \rangle_{\text{model}} = \langle v_i h_a \rangle^k \qquad \langle v_i \rangle_{\text{model}} = \langle v_i \rangle^k \qquad \langle h_a \rangle_{\text{model}} = \langle h_a \rangle^k \tag{2.4.25}$$

In total this combines to:

$$\frac{\partial \mathrm{KL}(q||p)}{\partial w_{ia}} \approx \langle v_i h_a \rangle^k - x_i^{(n)} \langle h_a \rangle^0 \tag{2.4.26a}$$

$$\frac{\partial \mathrm{KL}(q||p)}{\partial b_i^{(v)}} \approx \langle v_i \rangle^k - x_i^{(n)} \tag{2.4.26b}$$

$$\frac{\partial \mathrm{KL}(q||p)}{\partial b_a^{(h)}} \approx \langle h_a \rangle^k - \langle h_a \rangle^0 \tag{2.4.26c}$$

where each of these is calculated using sample $\boldsymbol{x}^{(n)}$. It is common to use values as low as $k = 1$ and we have done so as well.

Using an Restricted Boltzmann Machine

Having trained an (Binary) RBM, how does one actually use it?

1. Initialise the visible units of the RBM (to a random value)
2. Compute the hidden units.
3. Compute the visible units.
4. Repeat steps 2 and 3 a sufficient number of times. After enough updates the visible layer is distributed according to the probability $p(\{v_i\})$. The RBM is then said to be in equilibrium.
5. Sample the visible layer, updating sufficiently between each sample.

Sampling the visible layer generates a new dataset $\mathcal{D}'$ which is distributed with probability distribution $p(\{v_i\}) = \sum_{\{h_a\}} p(\{v_i\}, \{h_a\})$. This probability distribution should approximate the trained distribution $q(\{v_i\})$ after enough training.

Chapter 3

The Renormalization Group

3.1 Introduction

The renormalization group is a technique which analyses in the macroscopic behaviour of a system by integrating out the microscopic degrees of freedom. The idea is that transformation transforms the Hamiltonian H of the original system to a new Hamiltonian H' of the transformed system. The partition function, which describes the behaviour of the system remains unchanged and we can often cast the new Hamiltonian in the same form as the old Hamiltonian but with different coupling constants. This means that we can examine the change in the coupling as we integrate out short range behaviour.

The renormalization group is often discussed in the context of phase transitions, fixed points and critical behaviour. This is because at a continuous phase transition the system becomes scale invariant as more and more lengthscales become relavant and the long range behaviour starts to dominates the system. The coupling constants must thus remain invariant under the renormalization group transformation.

Let us start by examining the two-point correlation function $G(\boldsymbol{r}, \boldsymbol{r}')$ and the correlation length ξ. The two-point correlation function $G(\boldsymbol{r}, \boldsymbol{r}')$ is defined, for a spin system, as:

$$G(\boldsymbol{r}, \boldsymbol{r}') = \langle S(\boldsymbol{r})S(\boldsymbol{r}')\rangle - \langle S(\boldsymbol{r})\rangle\, S(\boldsymbol{r}') \tag{3.1.1}$$

Far away from the boundary the system is translationally invariant, and so symmetry enforces that $G(\boldsymbol{r}, \boldsymbol{r}') = G(\boldsymbol{r} - \boldsymbol{r}')$. It can be shown[5] that this function decays exponentially, allowing us to write:

$$G(r) = e^{-r/\xi} \tag{3.1.2}$$

where ξ is the correlation length. Let us imagine that we have a lattice with lattice spacing a and we express ξ in terms of this lattice spacing. Now we apply the renormalization group to the system, integrating out the short range interactions and then scales the system down. This new system can be expressed with a lattice with lattice spacing $a' = ba$ (with $b > 1$). Since the system itself hasn't changed, the real correlation length also remains unchanged, but when we express the correlation length in terms of the new lattice spacing, we see that:

$$\xi = \xi_b(ba) = \xi_1 a \Longleftrightarrow \xi' = \frac{\xi}{b} \tag{3.1.3}$$

this decreases. Now at a special so called fixed point the system transforms into itself, so we must have:

$$\xi' = \frac{\xi}{b} \overset{\substack{\text{fixed}\\\text{point}}}{=\!=} \xi \tag{3.1.4}$$

From this we can conclude that fixed points come in two flavors: critical fixed points ($\xi = \infty$) and trivial fixed points ($\xi = 0$). Finding and examining the fixed points is one of the uses of the renormalization group.

We will examine the Ising model model using the renormalization group.

3.2 The Ising Model

We will analyse the 1D and 2D ising model. In general we can write the Hamiltonian[*] of any interacting spin system as:

$$\beta H = -K^0 - \sum_i K_i^1 S_i - \sum_{ij} K_{ij}^2 S_i S_j - \sum_{ijk} K_{ijk}^3 S_i S_j S_k - \dots \tag{3.2.1}$$

The Ising model makes a couple of important simplifications: There are only interactions between nearest neighbours and $S_i = \pm 1$. This changes the Hamiltonian to the standard Ising Hamiltonian

$$\beta H = -\sum_i h_i S_i - \sum_{\langle ij \rangle} K_{ij} S_i S_j \tag{3.2.2}$$

where $\langle ij \rangle$ denotes that ij are neighbours and $K_{ij} = \beta J_{ij}$, with J_{ij} the coupling constant. We will generally work with K_{ij} since we only work with βH. Furthermore we make the the approximation that the lattice is a square lattice and that the coupling constants are constant as a function of position:

$$\beta H = -\sum_i h S_i - \sum_{\langle ij \rangle} K S_i S_j \tag{3.2.3}$$

[*]We will absorb the inverse temperature $\beta = (k_B T)^{-1}$ into the various coupling constants. We will generally treat K^2 as the inverse temperature.

3.2.1 The 1D Ising Model

We have a ring of spins all interacting according to the standard Hamiltonian with periodic boundary conditions:

$$\beta H = -h\sum_i S_i - K\sum_{\langle ij\rangle} S_iS_j = -\sum_i \left(\frac{h}{2}(S_i + S_{i+1}) + KS_iS_{i+1}\right) \tag{3.2.4}$$

The partition function is found as:

$$Z = \sum_{\{S_i\}} e^{-\beta H} = \sum_{\{S_i\}}\prod_i e^{\frac{h}{2}(S_i+S_{i+1})+KS_iS_{i+1}} \tag{3.2.5}$$

It is obvious that we can rewrite the partition function according to

$$Z = \sum_{\{S_i^{\text{odd}}\}}\sum_{\{S_i^{\text{even}}\}}\prod_i e^{\frac{h}{2}(S_i+S_{i+1})+KS_iS_{i+1}} \tag{3.2.6a}$$

$$= \sum_{\{S_i^{\text{odd}}\}}\sum_{\{S_i^{\text{even}}\}} e^{\frac{h}{2}(S_1+S_2)+KS_1S_2} e^{\frac{h}{2}(S_2+S_3)+KS_2S_3} e^{\frac{h}{2}(S_3+S_4)+KS_3S_4}\dots \tag{3.2.6b}$$

$$= \sum_{\{S_i^{\text{odd}}\}}\left[\sum_{\{S_2\}} e^{\frac{h}{2}(S_1+S_2)+KS_1S_2} e^{\frac{h}{2}(S_2+S_3)+KS_2S_3}\sum_{\{S_4\}}\dots\right] \tag{3.2.6c}$$

Let us examine one particular even sum. We also introduce a constant K^0, which doesn't affect the partition function since the partition function is invariant under multiplicative constants. We introduce this to account for any additive constants to the Hamiltonian after the renormalization process.

$$\begin{gathered}\sum_{S_j=\pm1} e^{K^0+\frac{h}{2}(S_i+S_{i+1})+KS_iS_{i+1}} e^{K^0+\frac{h}{2}(S_{i+1}+S_{i+2})+KS_{i+1}S_{i+2}} = \\ e^{2K^0}e^{\frac{h}{2}(S_i+S_{i+2})}\left[e^{(h+KS_i+KS_{i+2})} + e^{-(h+KS_i+KS_{i+2})}\right]\end{gathered} \tag{3.2.7}$$

$$\begin{gathered}\sum_{S_j=\pm1} e^{\frac{h}{2}(S_i+S_{i+1})+KS_iS_{i+1}} e^{\frac{h}{2}(S_{i+1}+S_{i+2})+KS_{i+1}S_{i+2}} = \\ 2e^{2K^0}e^{\frac{h}{2}(S_i+S_{i+2})}\cosh\left(h + K(S_i + S_{i+2})\right)\end{gathered} \tag{3.2.8}$$

On the other hand, the new system has the same form as the original one: a ring of spins interacting with their neighbours. We could describe it using the Ising model, with parameters K'^0, h', K'. Because both descriptions are of the same model, the partition functions must be equal and thus:

$$e^{K'^0+\frac{h'}{2}(S_i+S_{i+2})+K'S_iS_{i+2}} = 2e^{2K^0}e^{\frac{h}{2}(S_i+S_{i+2})}\cosh\left(h + K(S_i + S_{i+2})\right) \tag{3.2.9}$$

Thus we find:

$$e^{K'^0+h'+K'} = 2e^{2K^0}e^{h}\cosh\left(h+2K\right) \qquad S_i = S_{i+2} = 1 \tag{3.2.10a}$$
$$e^{K'^0-h'+K'} = 2e^{2K^0}e^{-h}\cosh\left(h-2K\right) \qquad S_i = S_{i+2} = -1 \tag{3.2.10b}$$
$$e^{K'^0-K'} = 2e^{2K^0}\cosh\left(h\right) \qquad S_i = -S_{i+2} \tag{3.2.10c}$$

So we find that:

$$e^{4K'^0} = 16e^{8K^0}\cosh(h+2K)\cosh(h-2K)\cosh^2(h) \tag{3.2.11a}$$
$$e^{2h'} = e^{2h}\frac{\cosh(h+2K)}{\cosh(h-2K)} \tag{3.2.11b}$$
$$e^{4K'} = \frac{\cosh(h+2K)\cosh(h-2K)}{\cosh^2(h)} \tag{3.2.11c}$$

As mentioned before, K'^0 is a multiplicative constant to the partition function and thus does not affect the physics of the system. The state of the system is only determined by h' and K'.

$$h' = h + \frac{1}{2}\ln\left[\frac{\cosh(h+2K)}{\cosh(h-2K)}\right] \tag{3.2.12a}$$
$$K' = \frac{1}{4}\ln\left[\frac{\cosh(h+2K)\cosh(h-2K)}{\cosh^2(h)}\right] \tag{3.2.12b}$$

From here we can find the fixed points of the system $h' = h = h^*$ and $K' = K = K^*$

$$h^* = h^* + \frac{1}{2}\ln\left[\frac{\cosh(h^*+2K^*)}{\cosh(h^*-2K^*)}\right] \tag{3.2.13a}$$
$$K^* = \frac{1}{4}\ln\left[\frac{\cosh(h^*+2K^*)\cosh(h^*-2K^*)}{\cosh^2(h^*)}\right] \tag{3.2.13b}$$

From Eq (3.2.13a) we find that:

$$\cosh(h^*+2K^*) = \cosh(h^*-2K^*) \tag{3.2.14}$$

which is fulfilled when $h^* = 0$ or $K^* = 0$ or $h^* \to \pm\infty$ or $K^* \to \pm\infty$. In the case where $h^* \to \pm\infty$ we see, from Eq. (3.2.13b), that $K^* = 0$. This corresponds to the system being fully aligned with the magnetic field. In the case where $h^* = 0$ we find that

$$2K' = \ln\left[\cosh(2K)\right] \tag{3.2.15}$$

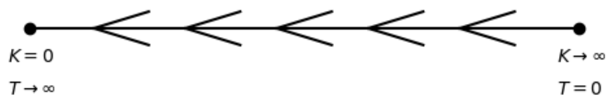


Figure 3.1: *RG-flow for the 1D Ising model*

from which we find that:

$$\tanh(K') = \frac{e^{2K'}-1}{e^{2K'}+1} = \frac{\cosh(2K)-1}{\cosh(2K)+1} = \frac{e^{2K}-2+e^{-2K}}{e^{2K}+2+e^{-2K}} \tag{3.2.16}$$

$$\tanh(K') = \tanh^2(K) \quad \text{for } h=0 \tag{3.2.17}$$

This implies that $K^* = 0$ or $K^* \to \infty$.

In this case it's also easy to see that for $h = 0$ there is a flow to $K = 0$ since $\tanh^2(K) \leq \tanh(K)$ with equality only for the fixed points.

At every fixed point we see that K flows towards zero, or equivalently $T \to \infty$.

We have found that in the entire phase space there is an RG flow towards the high temperature fixed point at $K = 0$.

3.2.2 The 2D Ising Model

For the 2D Ising model we will examine the phase transition at $h = 0$, following[9]. Let us write down the partition function:

$$Z = \sum_{\{S_i\}} \exp(-\beta H) = \sum_{\{S_i\}} \exp\left(K \sum_{\langle ij \rangle} S_i S_j\right) \tag{3.2.18}$$

Now we can use that $S_i S_j = \pm 1$ to write:

$$\begin{aligned} \exp(K S_i S_j) &= \cosh(K S_i S_j) + \sinh(K S_i S_j) \\ &= \cosh(K) + S_i S_j \sinh(K) \\ \exp(K S_i S_j) &= \cosh(K)\big[1 + S_i S_j \tanh(K)\big] \end{aligned} \tag{3.2.19}$$

Substituting this into the partition function gives us:

$$Z = \sum_{\{S_i\}} \prod_{\langle ij \rangle} \cosh(K)\big[1 + S_i S_j \tanh(K)\big] \tag{3.2.20}$$

If the system consists of N spins, then there are $2N$ pairs of nearest neighbours.

$$\frac{Z}{\cosh^{2N}(K)} = \sum_{\{S_i\}} \prod_{\langle ij \rangle} \left[1 + S_i S_j \tanh(K)\right] \tag{3.2.21}$$

We can now work out this finite product and arrange it in orders of $\tanh(K)$, giving a low K expansion*. The first few terms are then given by:

$$\frac{Z}{\cosh^{2N}(K)} = \sum_{\{S_i\}} \left(1 + \sum_{\langle ij \rangle} S_i S_j \tanh(K) + \sum_{\langle ij \rangle} \sum_{\langle kl \rangle \neq \langle ij \rangle} S_i S_j S_k S_l \tanh^2(K) + \dots\right) \tag{3.2.22}$$

We also use the following trivial identities:

$$\sum_{\{S_i\}} S_i S_j = 0 \qquad\qquad \sum_{\{S_i\}} 1 = 2^N \tag{3.2.23}$$

Using this we see that:

$$\begin{aligned} &\sum_{\{S_i\}} (S_{i,j} S_{i,j+1})(S_{i,j+1} S_{i+1,j+1})(S_{i+1,j+1} S_{i+1,j})(S_{i+1,j} S_{i,j}) \\ =&\sum_{\{S_i\}} S_{i,j}^2 S_{i,j+1}^2 S_{i+1,j+1}^2 S_{i+1,j}^2 = \sum_{\{S_i\}} 1 = 2^N \end{aligned} \tag{3.2.24}$$

From this we recognise that unless we form a closed loop of nearest neighbours the contribution is going to be zero.

The lowest order terms are: 4 spins in a square, 6 spins in a rectangle, and 8 spins which can be arranged into a single square, a rectangle, an L-shape and in 2 squares (which can be neighbours, but may not overlap). A square can be in N places, a rectangle can be horizontal or vertical and can also be in N places. Finally two squares can be in $\frac{N(N-5)}{2}$ places (N for the first square, $N-5$ remaining spots for the second square since the bonds can only be counted once and finally a factor of $\frac{1}{2}$ to account for the symmetry between the two squares) and the L-shape has 4 orientations and can be in N places. These shapes are also summarised in figure 3.2.

The first few surviving terms are then given by:

$$\frac{Z}{\cosh^{2N}(K) 2^N} = 1 + N \tanh^4(K) + 2N \tanh^6(K) + \frac{N(N+9)}{2} \tanh^8(K) + \dots \tag{3.2.25}$$

*Unless we terminate the sequence this expansion is and remains exact even at low temperatures. We are just ordering the terms in orders of $\tanh(K)$.

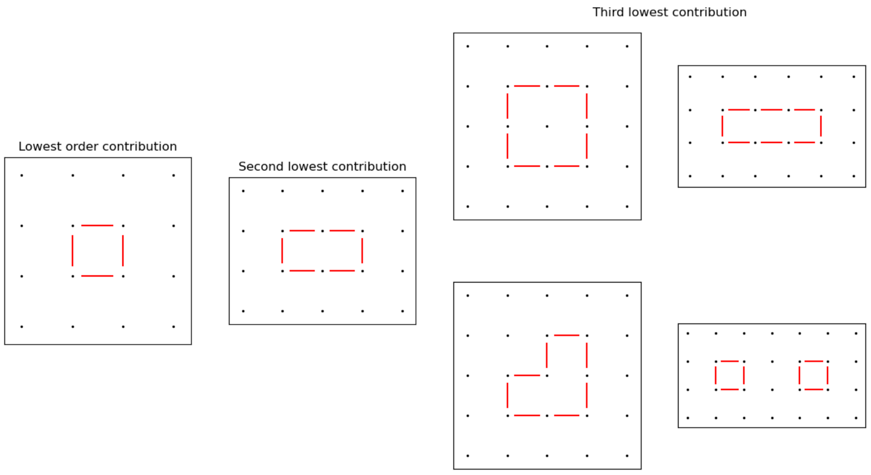
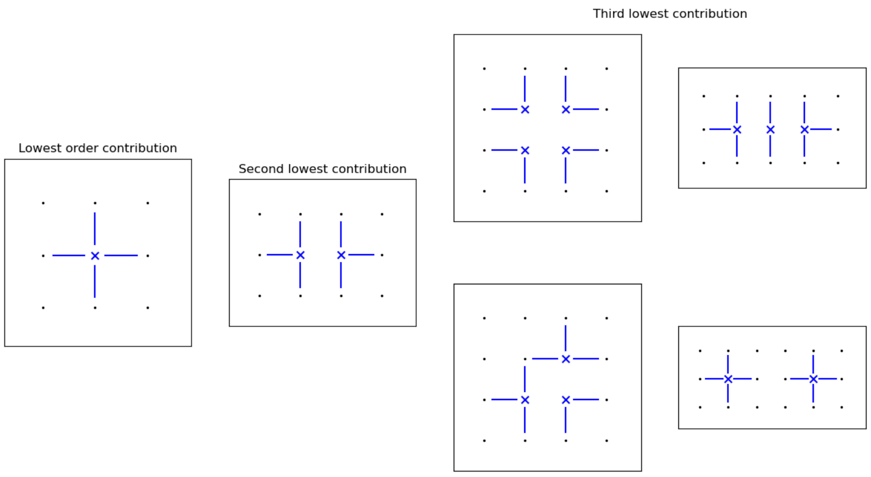

Lowest order contribution
Second lowest contribution
Third lowest contribution

Figure 3.2: *Visualisation of the first three lowest order terms of the low K expansion. The red lines indicate pairs of nearest neighbours (S_iS_j) in the product. Image reproduced from Iso, et al.[7]*

where the last term is the sum of the three contributions $\frac{1}{2}N(N+9) = \frac{1}{2}N(N-5) + N + 2N + 4N$.

We could also build the partition function by starting with a fully aligned state and then perturb this by flipping more and more spins. This high K expansion is again exact if we take all finitely many terms into account. The lowest two contributions are then a single flipped spin, two flipped spin which are neighbours. The third contribution consists of two flipped spins which are not neighbours, four spins in a square, three neighbouring spins on a line and three neighbouring spins in an L-shape.

The fully aligned state has energy $\beta H = -K\sum_{\langle ij\rangle} 1 = -2NK$. For a single flipped spin there are N places where it can be and the energy difference (from the fully aligned state) is due to four broken bonds gives $\beta\Delta H = 2*4K$, for two spins which are neighbours there are N places for horizontal and vertical neighbours with an energy difference of $2*6K$. Finally we can create eight broken bonds by having a square of 2×2 spins, a rectangle of 1×3, two flipped spins which are not neighbours which can be in $N(N-5)/2$ places (again due to symmetry reasons and the 5 inaccessible spins due to not being neighbours or being the same spin) and three spins in an L-shape (which can be in 4 orientations and N positions) with an energy difference of $2*8K$. Again we have summarised these terms in



Lowest order contribution
Second lowest contribution
Third lowest contribution

Figure 3.3: *Visualisation of the first three lowest order terms of the high K expansion. The blue crosses indicate anti-aligned spins and the blue lines indicate the resulting broken bonds. Image reproduced from Iso, et al.[7]*

figure 3.3. We can put all of this together, with the first few terms being:

$$\frac{Z}{e^{2NK}} = 1 + Ne^{-8K} + 2Ne^{-12K} + \frac{N(N+9)}{2}e^{-16K} + \dots \tag{3.2.26}$$

We want to stress, again, that these two expansions would be exact if we take all terms into account. Furthermore these two description mirror each other and if the flipped spins in the high K expansion are surrounded by a square, the graphical rules of the low K expansion map onto the high K expansion. To make this precise we define the following coupling:

$$\tanh(K_{\text{low}}) = \exp\left(-2K_{\text{high}}\right) \tag{3.2.27}$$

Let us start with rewriting the coupling equation as:

$$\sinh\left(2K_{\text{high}}\right) = \frac{1}{2\tanh(K_{\text{low}})}\left(1 - \tanh^2(K_{\text{low}})\right) \tag{3.2.28}$$

using that $1 - \tanh^2(x) = \frac{1}{\cosh^2(x)}$ and $2\sinh(x)\cosh(x) = \sinh(2x)$ we get:

$$\sinh\left(2K_{\text{high}}\right)\sinh\left(2K_{\text{low}}\right) = 1 \tag{3.2.29}$$

Using this transformation we can also equate the partition functions as

$$\frac{Z(K_{\text{low}})}{\cosh^{2N}(K_{\text{low}})2^N} = \frac{Z(K_{\text{high}})}{e^{2NK_{\text{high}}}} \tag{3.2.30}$$

or:

$$\frac{Z(K_{\text{low}})}{\sinh^{N/2}(K_{\text{low}})2^N} = \frac{Z(K_{\text{high}})}{\sinh^{N/2}(K_{\text{high}})} \tag{3.2.31}$$

Finally we can find the critical coupling by finding $K_c = K_{\text{low}} = K_{\text{high}}$:

$$\sinh^2(2K_c) = 1 \tag{3.2.32}$$

$$K_c = \frac{1}{2}\operatorname{arsinh}(1) = \frac{1}{2}\ln\left(1+\sqrt{2}\right) \tag{3.2.33}$$

where we used that $K_c > 0$ and real. We have found the critical temperature.

Because we can map the two descriptions onto each other, the RG flow in these phases must be exactly opposite. So let us examine the high coupling regime. We will perform an RG transformation called the block spin transformation, which is defined as:

$$S_I = \frac{1}{|m_I|}\frac{1}{l^d}\sum_{i\in I} S_i \tag{3.2.34a}$$

$$|m_I| = \frac{1}{l^d}\sum_{i\in I}\langle S_i\rangle \tag{3.2.34b}$$

now if we use blocks of 3×3 then this reduces to the majority rule, in which we replace a block of spins with a single spin pointing in the same direction as the majority of that block. Now if we have a system deep in the high coupling regime, then most spins are pointing in the same direction, with some small regions pointing in the opposite direction. When we now apply the transformation we see that single unaligned spins and small anti-aligned squares of 2×2 get removed from the system. So we must have moved further into the strong coupling regime.

The RG-flow is then away from the critical fixed point K_c towards the relative trivial bulk fixed points at $K = 0$ and $K \to \infty$.

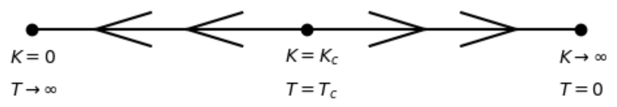


Figure 3.4: *RG-flow for the 2D Ising model*

Chapter 4

Connecting Neural Networks and the Renormalization Group

Having discussed neural networks and the renormalization group in the previous chapters we can now try to connect the two. There are various different ways in which we can connect the renormalization group to the neural networks. We could examine how information flows through a deep (feed-forward) neural network[6]. We could examine the properties of learning neural networks[10]. We could train a feed-forward neural network on a physical system and interpret its output as an order parameter[2][3]. We could examine the evolution of states of the restricted Boltzmann machine (RBM) in the light of the renormalization group[7][14].

The restricted Boltzmann machine has a particularly striking resemblance with the renormalization group. It consists of two layers, the visible layer which acts as the sample and a hidden layer. Where constructing the hidden layer removes information from the system (if the hidden layer is smaller than the visible layer) and constructing the visible layer creates a new sample. It feels almost natural to connect this to the renormalization group where we integrate out the short range modes and then rescale the system.

We follow in the footsteps of Satoshi Iso Shotaro Shiba, and Sumito Yokoo[7] who have trained an RMB on a 2D Ising system. They examined the flow of states generated by the RBM and found that they flowed towards the critical point, opposite to the renormalization group flow.

This opposite flow is puzzling in light of the renormalization group, where any loss of information and rescaling should push you away from the critical point.

4.1 Connecting the Restricted Boltzmann Machine to the Renormalization Group

In chapter 2 we discussed the Binary Restricted Boltzmann Machine (section 2.4.2). There we showed that given a visible sample $\{v_i\}$ we can find the hidden units $\{h_a\}$ which in turn generates a new visible sample $\{v_i^1\}$. When we repeat this ad infinitum the system will explore the entire phase space according to the probability distribution p (which for a trained system should approximate the real world probability distribution q). Now if the number of hidden units is smaller then the number of visible units $N_h < N_v$ we can imagine that a sample $\{v_i^\alpha\}$ gets scaled down to $\{h_a^\alpha\}$ and then scaled back up to the next configuration $\{v_i^{\alpha+1}\}$. This procedure looks a lot like the Renormalization Group procedure that we discussed in chapter 3.

We use the following general procedure:

1. Use the Metropolis Algorithm to generate a set of configurations at different temperatures ($T = K^{-1} \in \{0, 0.25, \dots, 5.75, 6\}$). We generate 3 datasets, the training and test datasets and a final dataset

2. Train a MultiLayer Perception (MLP) on the training dataset to estimate the probability that a spin configuration was generated at a particular temperature as discussed in chapter 2. The final layer of the MLP has the SoftMax activation function which allows us to interpret the N output neurons as probabilities belonging to specific temperatures (the output is a vector $(p(T = 0), P(T = 0.25), \dots)$ which is normalised to 1). The corresponding cost function is the Cross-Entropy. While the temperature estimation of a single configuration is nearly impossible (unless the configurations are absurtly large) estimating the temperature of a collection of samples is quite feasible.

3. Train a Restricted Boltzmann Machine on (a portion of) the training dataset. Hereafter we no longer update the parameters of the neural networks.

4. We can now set the visible units $\{v_i^0\}$ equal to a configuration from the final dataset generated at a particular temperature T.

5. Using the fixed parameters of the RBM we can find the hidden units $\{h_a^0\}$ as discussed before. The expectation value is given by (we use that for the 2D Ising model $v_i, h_a = \pm 1$):

$$\langle h_a^0 \rangle = \tanh\left(\sum_i v_i^0 w_{ia} + b_a^{(h)}\right) \tag{4.1.1}$$

6. Using the newly constructed hidden layer we can find the visible units $\{v_i^1\}$, whose expectation value is given by:

$$\langle v_i^1 \rangle = \tanh\left(\sum_a w_{ia} h_a^0 + b_i^{(v)}\right) \tag{4.1.2}$$

7. Repeating this we generate a flow of samples:

$$\{v_i^0\} \to \{h_a^0\} \to \{v_i^1\} \to \{h_a^1\} \to \{v_i^2\} \to \{h_a^2\} \to \dots \tag{4.1.3}$$

$$\{v_i^0\} \to \{v_i^1\} \to \{v_i^2\} \to \dots \tag{4.1.4}$$

8. We can repeat this for multiple samples from the same thermal distribution to see how the entire thermal distribution changes: $\{\boldsymbol{v}_{(n)}^0\} \to \{\boldsymbol{v}_{(n)}^1\} \to \dots$

9. Finally we can use our MLP to estimate the temperature at each "time step" to convert this RBM-flow to a temperature flow: $T \to T' \to T'' \to \dots$

The neural networks are coded in python using NumPy, without any additional packages.

We want to stress that there are multiple different (and independent) temperatures at play here. We have the temperature set from which we draw samples and over which the MLP is trained $\{T_{\text{train}}\}$, over which the RBM is trained $\{T_{\text{RBM}}\}$ (which is a subset of $\{T_{\text{train}}\}$) and the temperature of the starting configuration at the start of RBM-flow T_{flow}. Specifically T_{flow} does not need to be in $\{T_{\text{RBM}}\}$.[*] Furthermore temperature is measured in units of k_B/J.

Finally feeding a collection of spin configurations into the multilayer perceptron outputs a probability distribution from which we can estimate the temperature of the collection. This assumes that the collection is actually a thermal distribution, and while this assumption is obviously correct for the spin configurations at the start of the RBM-flow we do not have the guarantee that this holds at later "time steps". Specifically while we have the guarantee that sampling an RBM trained on a thermal distribution gives that thermal distribution, we do not have the guarantee that it maps a thermal distribution to a thermal distribution. As such keep this in mind when we start talking about the temperature distributions of collections spin configurations.

[*]T_{flow} does need to be in $\{T_{\text{train}}\}$ since we want to be able to measure the temperature of subsequent generated samples

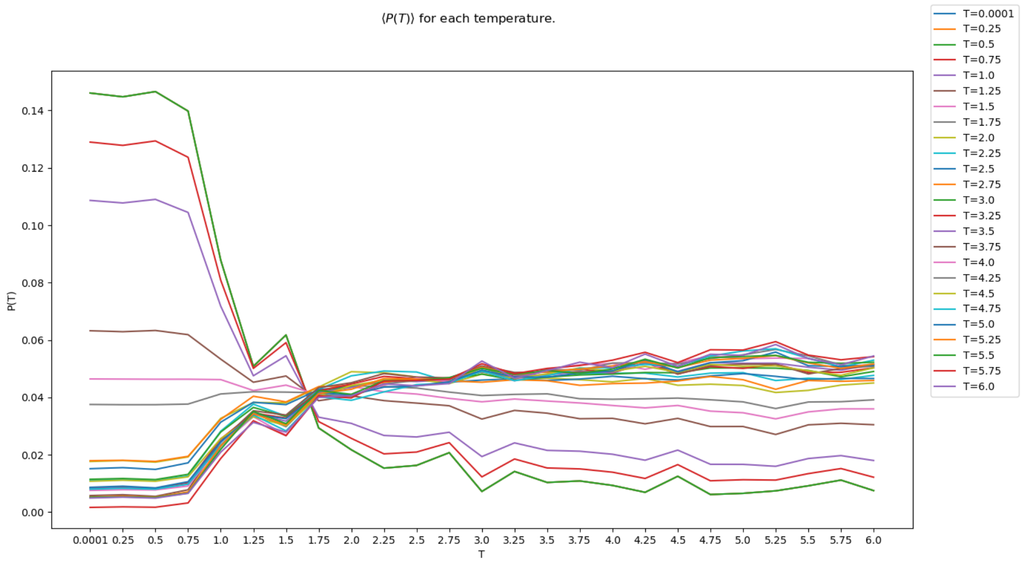


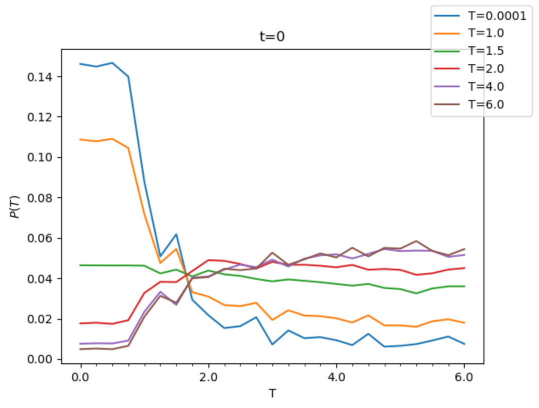
Figure 4.1: ***1D Ising***. *The various probability curves as a function of temperature for different temperatures. The temperature is expressed in units of k_B/J and equal to $T = K^{-1}$. The important thing is that different temperature curves can be distinguished from each other. Each probability curve is averaged over 1600 samples.*

4.2 1D Ising

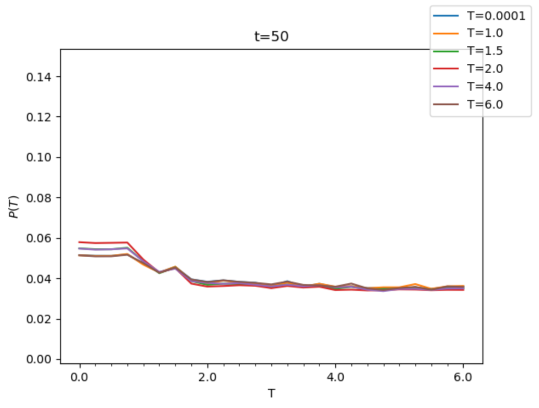
We start by examining the one-dimensional Ising model, which is discussed by ter Hoeve[14]. Our 1D Ising model consists of only 8 spins. The first layer of both our neural networks thus has 8 neurons. The hidden layer of the RBM has 4 neurons, while the MLP has 40 neurons in the hidden layer and 25 neurons in the output layer (equal to the number of different temperatures we have). Both the RBM and MLP use tanh activation functions, with the output layer of the MLP having a SoftMax activation function. We also employ gradient descent without adding regularisation terms to the energy.

For every temperature we have three datasets containing 1600 samples. We use one of these datasets to train the neural networks, one to test the neural network and one dataset to generate our results.

We have trained our discriminative multi-layer perceptron (MLP) to find the temperature of a collection of samples, minimising the test-energy (as discussed in section 2.2.3). The resulting probability distributions for the thermal distributions are shown in Figure 4.1. The important fact to extract is that it's possible to distinguish different temperatures based on their distributions with varying ac-



(a) *The temperature probability distributions at the start of the RBM-flow.*



(b) *The temperature probability distributions at the end of the RBM-flow.*

Figure 4.2: ***1D Ising***. *The temperature probability curves after the RBM procedure discussed above for an RBM trained on all temperatures. Importantly all thermal distributions flow towards the same stable temperature (at roughly $T = 1.5$).*

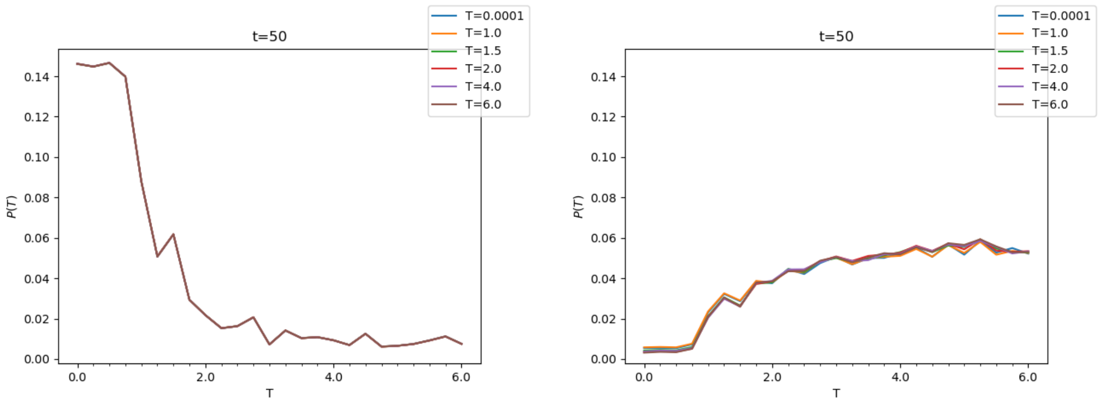
curacy. The overlap of various temperature curves is a finite size effect, as the curves become more distinct as the size of the system grows. We want to stress again that when we state "The temperature of the final distribution is T" what we mean is "The temperature distribution of the final collection of spin configurations is sufficiently close to the temperature distribution of a thermal distribution with temperature T".

The question that remains is on which subset of temperatures do we train our restricted Boltzmann machine (RBM). For visual clarity we will only show 6 temperature distributions ($T \approx 0$, $T = 1$, $T = 1.5$, $T = 2$, $T = 4$ and $T = 6$). We can now train the RBM based on three different subsets: Figure 4.2 shows the RBM trained on all temperatures; Figure 4.3a shows the RBM trained only on $T = 0$; and Figure 4.3b shows the RBM trained only on $T = 6$.

We want to point out to the reader that there are two important realisations to be taken away from the 1D case.

First, when we train the RBM on a single temperature all distributions converge to that temperature independent of the distribution at the start of the RBM-flow. This is as expected, since the RBM is supposed to recreate the trained distribution. This is in agreement with ter Hoeve[14]

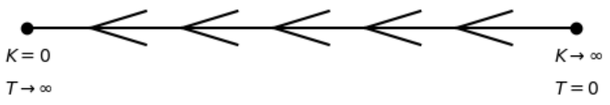
Second, when the RBM is trained over a range of temperatures the final convergent temperature distribution appears to be the distribution of a thermal distribution (at $T \approx 1.5$). That is, looking only at the final temperature distribution it seems like the spin configurations follow the thermal distribution at roughly



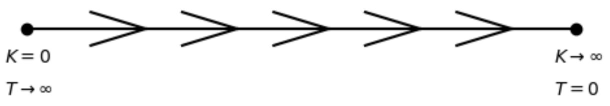
(a) *RBM trained on $T = 0$.* **(b)** *RBM trained on $T = 6$.*

Figure 4.3: ***1D Ising**. The probability distributions for RBMs trained at only a single temperature. In both cases we see that the all six probability distributions flow towards the probability distribution of the trained temperature.*

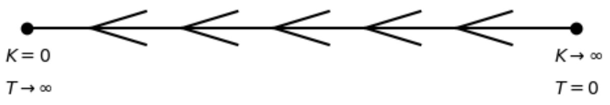
$T \approx 1.5$. The convergence to a single fixed point might be surprising, since the 1D Ising model has no non-trivial fixed points (only at $T = 0$ and $T \to \infty$). But we do expect all distributions to converge, since eventually the output of the RBM should be distributed according to the internal RBM probability distribution $p(\{v_i\})$ (see section 2.4.2). The question remains whether the convergent distribution is actually a thermal distribution, or just a mixture of various different temperature distributions.



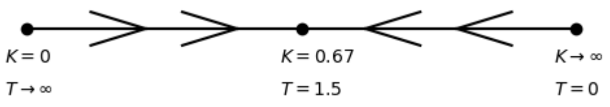
(a) *Theoretical RG-flow.*



(b) *RBM trained on $T = 0$.*



(c) *RBM trained on high T ($T = 10^4$).*



(d) *RBM trained on all T.*

Figure 4.4: ***1D Ising*** *RG-flow and RBM-flows for the three examined RBMs.*

4.3 2D Ising

We now continue by discussing the two-dimensional Ising model, which is originally examined by Iso, et al.[7]. In the two-dimensional case our system consists (like in Iso, et al.) of 10×10 spins (and thus the first layer of both our neural networks have 100 neurons), the RBM has an hidden layer with 25 (5×5) neurons and the MLP has an hidden layer with 64 neurons and an output layer with 25 neurons. The activation functions and method of optimisation are equal to those used in the 1D case.

The datasets used consist of 1000 samples per temperature. Where we used one dataset to train the neural networks, one dataset to test the neural network and one dataset to generate our results.

Figure 4.5 shows the probability curves for the various temperatures. Again the overlap of the lowest temperature curves and the overlap of the highest temperature curves is due to finite size effects.

Figure 4.6 shows the RBM flow for an RBM trained at all 25 temperatures. We observe that the temperatures converge to about $T = 2.5$ or roughly the critical temperature. We've also trained the RBM on the low temperatere and high temperature regimes, both including the critical temperature. In both cases the final temperature of the RBM flow is near T_c.[*] From this we can only conclude that something at (or near) the critical point is attracting the RBM flow. This is exactly opposite to the RG-flow and has been noted by both Iso, et al.[7] and ter Hoeve[14].

Furthermore we can train the RBM on all temperatures excluding $T = 1.75$ till $T = 2.75$ (20 temperatures in total), the result summarised in figure 4.8. Again, while the RBM-flow does not exactly end up at T_c it still flowed towards it and ended up near $T = 2$. This seems to support the claim that the RBM is performing an inverse RG-flow.

Iso, et al. conjecture that the scale invariant nature of configurations at the critical point causes it to act as a stabiliser and attractor for the RBM. In particular they argue that since the configurations are scale invariant they contain all the features of various temperatures simultaneously, and consequently act as stabilisers and attractors.

We decided to try and apply the RBM to an abstract dataset such as MNIST to test this further.

[*]On first glance the temperature distributions in figure 4.7b might look like the high temperature distribution ($T > 4$) but note the zig-zag in the high temperature distribution around $T = 4$ which is absent for all temperature distributions $T \geq 4$.

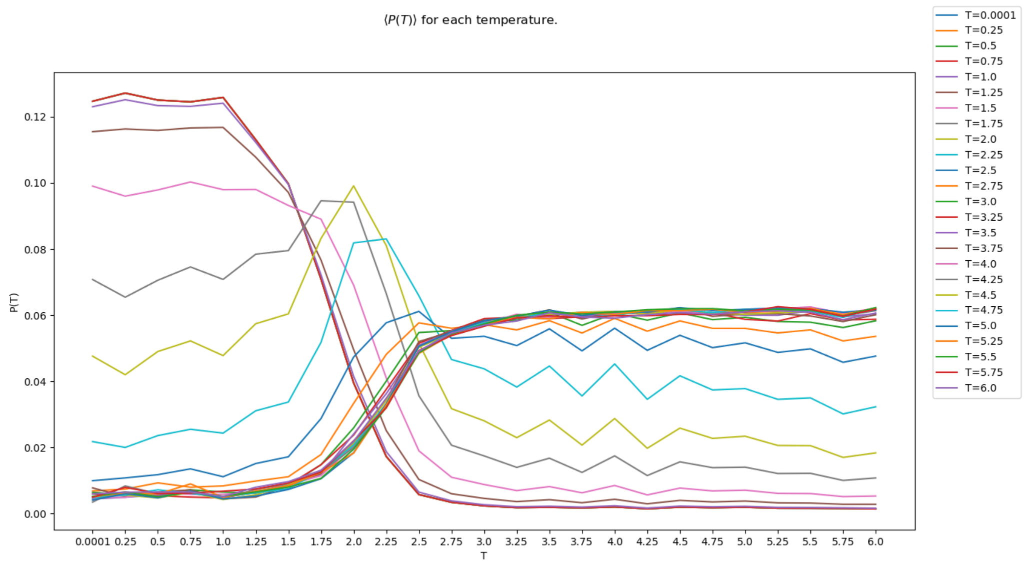
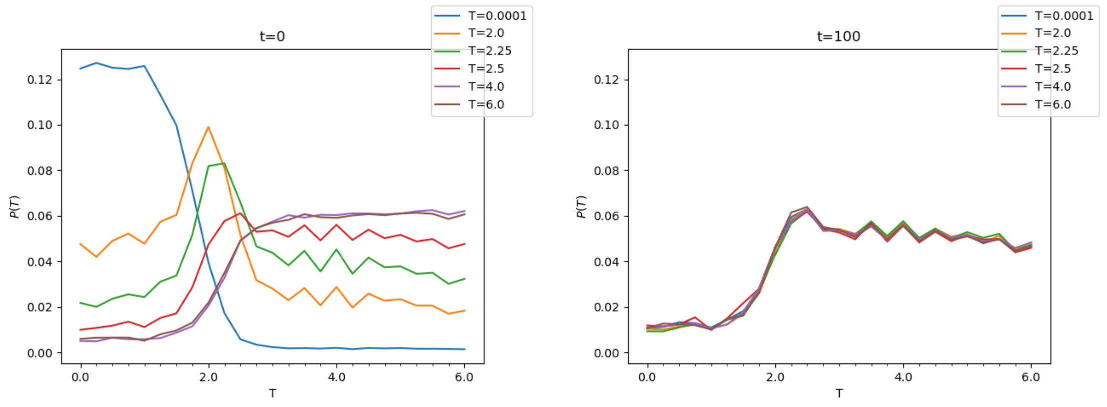
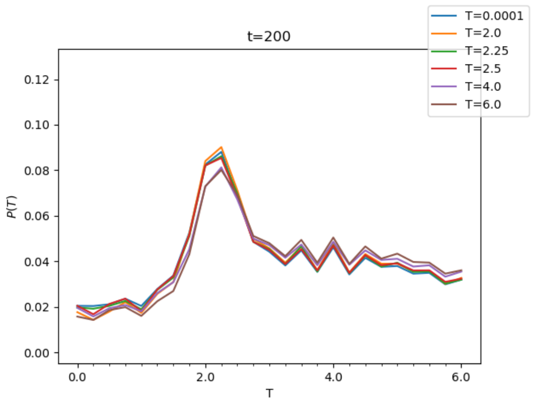


Figure 4.5: ***2D Ising***. *The various probability curves as a function of temperature for different temperatures. The critical temperature* $T_c = \frac{2}{\ln(1+\sqrt{2})} \approx 2.269$ *lies near the light blue line. Each probability curve is averaged over 1000 samples.*

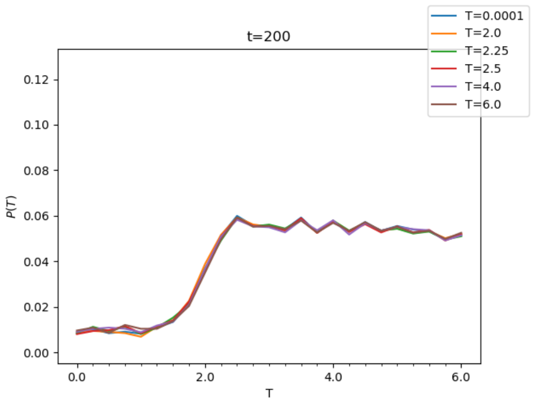


(a) *The temperature probability distributions at the start of the RBM-flow.* **(b)** *The temperature probability distributions at the end of the RBM-flow.*

Figure 4.6: ***2D Ising***. *The temperature probability curves after the RBM procedure for an RBM trained on all temperatures. The curves converge to a temperature at roughly* $T = 2.5$ *(which is near* T_c*)*



(a) *RBM trained on* $T \in \{0, 0.25, \ldots, 2.5, 2.75\}$.



(b) *RBM trained on* $T \in \{1.75, 2, \ldots, 5.75, 6\}$.

Figure 4.7: ***2D Ising***. *The temperature distributions at the end of the RBM flow for RBMs trained only on the low temperature/high temperature regime. Both regimes include a small region around T_c. While the two RBM flows don't converge to the exact same distribution, the final distribution in both cases is close to T_c.*

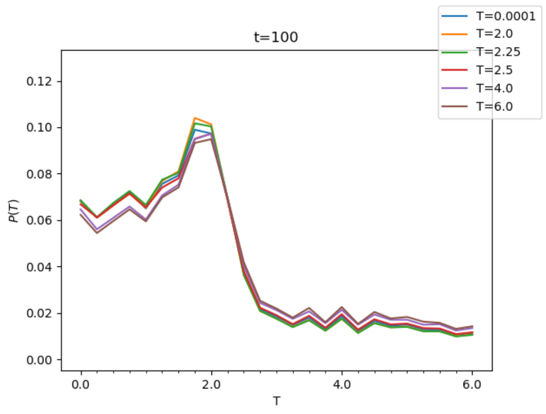
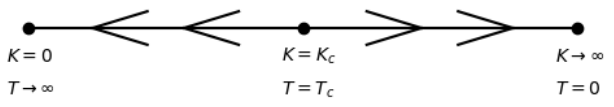
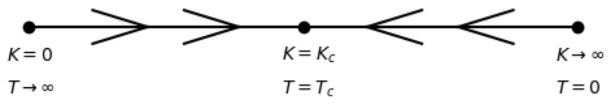


Figure 4.8: ***2D Ising*** *The temeprature distributions at the end of the RBM flow for an RBM trained on temperatures excluding $T = 1.75$ till $T = 2.75$.*



(a) *Theoretical RG-flow.*



(b) *RBM-flow for all examined cases.*

Figure 4.9: ***2D Ising*** *The RG-flow and RBM-flow for the examined RBMs.*

4.4 MNIST

The Modified National Institute of Standards and Technology dataset is a large dataset of handwritten digits. Figure 4.10 visualises some examples of these digits. In contrast with our previous examples we can no longer properly visualise phase space since it now has many dimensions. This means that it's impossible to estimate which numbers lie "close" together and which are far apart. But we can use dimensional reduction techniques such as principle component analysis and t-distributed stochastic neighbour embedding (t-SNE) to attempt an semi-accurate visualisation[12]. Figure 4.11 shows such a visualisation for the MNIST data found online. While t-SNE tries to preserve clusters, it does not preserve distance and length-scales can be deformed. It does preserve ordination, which at least allows us to make an educated guess which digits lie close together. From this picture we gather that fours, eights and nines are probably good candidates to train our RBM on.

We start by defining our neural network architecture. The input layer of both networks has 784 (28^2) neurons. The hidden layer of the RBM has 49 (7^2) neurons, while the MLP has three hidden layers with 196 (14^2), 49 (7^2) and 20 neurons. The output layer of the MLP has 10 neurons. The RBM uses sigmoid activation functions and the MLP has ReLU activation functions for the hidden layers and the SoftMax activation function for the output layer. The RBM now uses batch

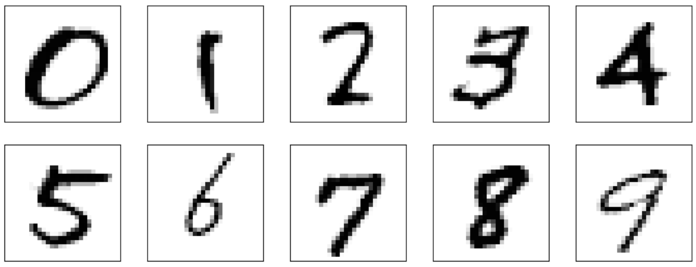


Figure 4.10: ***MNIST*** *Examples of digits from the MNIST dataset. Each image is* 28×28 *pixels.*

gradient descent without adding regularisation terms to the energy. The MLP on the other hand uses batch gradient descent in combination with ADAM and an L1-regularisation term (see section 2.2.3). This pays off in the fact that the probability curves are now sharply peaked around their respective number as well as an increase in training speed.

From the Ising model we would assume that if we train the RBM on two digits, there will be an RBM-flow towards a critical mixture of the two digits. That we get a flow towards a single distribution which is close to the phase boundary between these two digits.

We train the RBM on a dataset of fours and eights, see figure 4.12 and we observe that the probability distributions of fours and eights have essentially remained stable, but also the probability distribution of ones has stayed relatively stable. On the other hand we see that the other distributions are smeared out and or concentrated at four or eight. We want to note the high peak at one, for some reason this is a fixed point.

The fact that the there are multiple stable ensembles is surprising, since in every single RBM-flow previously we converged to a single ensemble (corresponding to a "temperature"). This means that it's possible, under specific conditions, for multiple fixed points to exist (just like in real RG). But we would expect that the fixed points at a single number correspond to the bulk phases of system and thus be attractive in RG. If the RBM-flow is an inverse RG-flow then these should be repulsive, but instead we see that they remain attractive.

We hypothesise that multiple stable distributions exist because of how sparse the dataset is in phase space. The MNIST dataset contains high amounts of structure (due to the nature of written digits) and as such we expect little overlap between different digits. On the other hand the Ising model has a lot of overlap, since the same energy (and thus configuration) is possible at different temperatures.

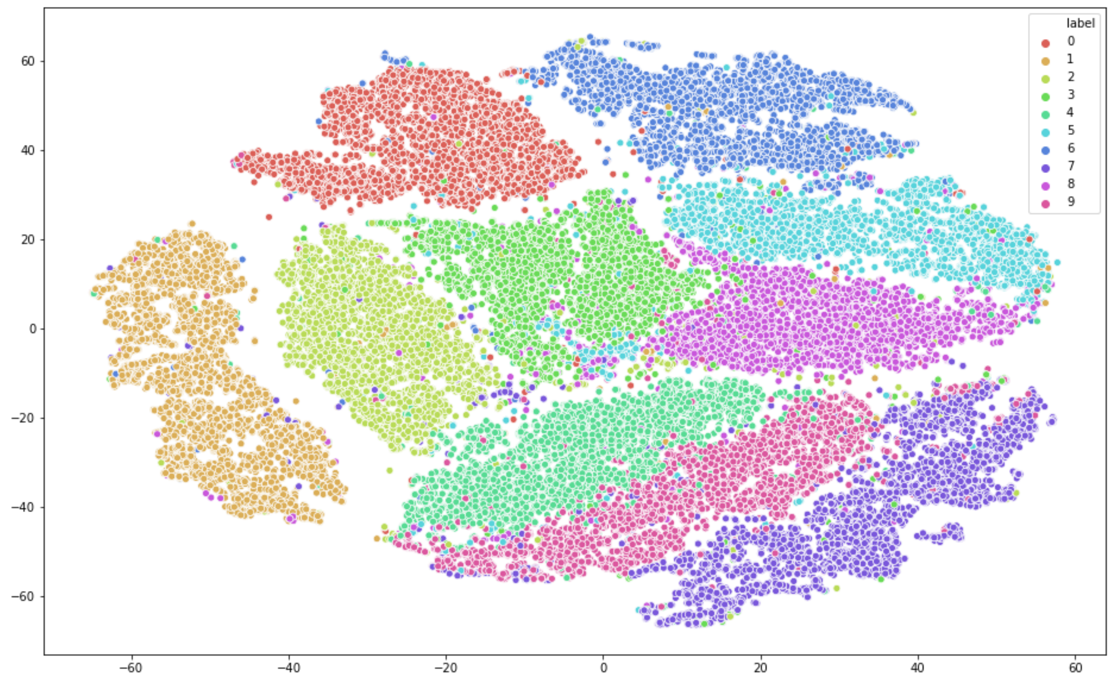
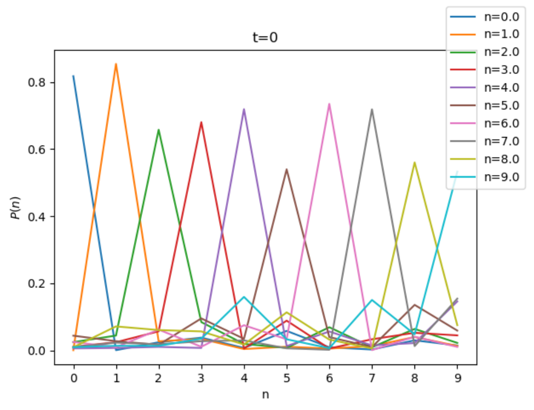
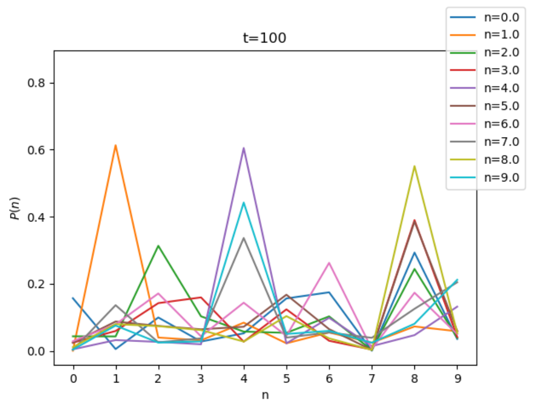


Figure 4.11: ***MNIST*** *2D Scatter plot of MNIST data after applying PCA (n_components = 50) and then t-SNE. Image scource from Dehao Zhang[15]*



(a) *Probability distribution at the start of the RBM-flow.*



(b) *Probability distribution at the end of the RBM-flow for an RBM trained on fours and eights.*

Figure 4.12: ***MNIST*** *RBM-flow for an RBM trained at fours and eights.*

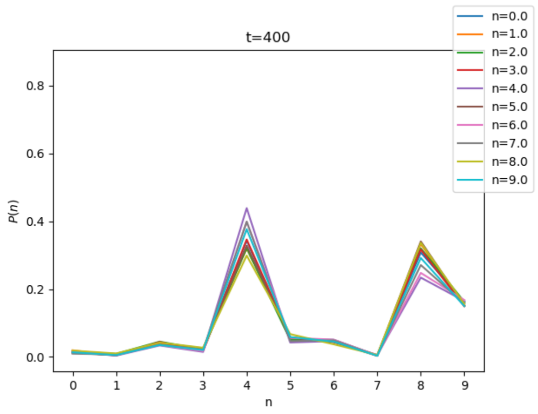


Figure 4.13: ***MNIST*** *The end of the RBM-flow for an RBM trained at fours, eights and a mixture of fours and eights.*

We can artificially connect the fours and eights by creating artificial samples which are a combination of both. We can make a new sample by taking a sample from four and a sample from eight and adding these together according to:

$$n' = (1 - \alpha)n_1 + \alpha n_2 \tag{4.4.1}$$

Our training data per digit is about 2700 samples and we add 4500 samples with $\alpha \in \{0.1, 0.2 \ldots, 0.9\}$ (500 samples per α). The majority of the data still lies near the trivial bulk fixed points, with a line now connecting them.

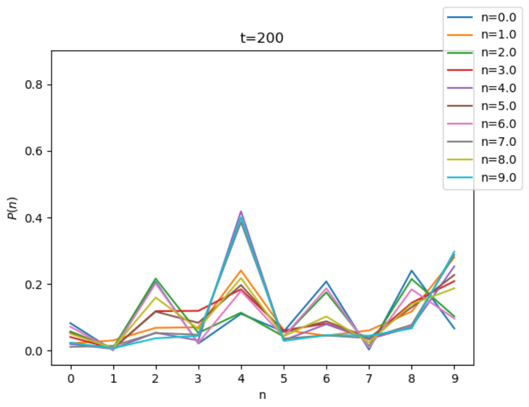
Figure 4.13 shows that the resulting RBM-flow again converges to a single distribution, just like for the Ising-model. This new distribution is now a mixture between a four and an eight and we have lost the bulk fixed points.

We also tried only adding 900 samples (100 samples per α), but this did not result in a single convergent distribution.

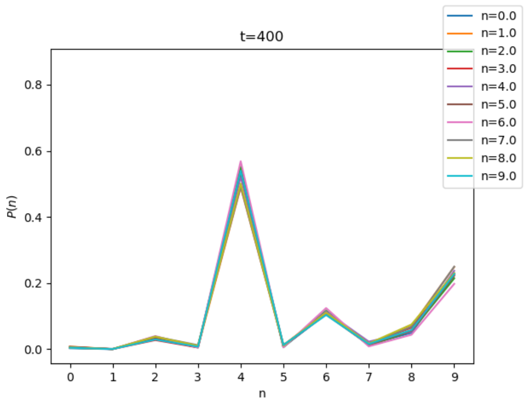
We can repeat this for an RBM trained on fours and nines, as shown in figure 4.14. Again we see that there are multiple fixed points when the RBM is trained only on fours and nines. We point out that the peak at one we saw earlier (trained on four and eight) has disapeared. When the RBM is also trained on a mixture of fours and nines (4500 additional samples) the distributions converge.

Right now we are connecting the two bulk phases through a linear path. We observe that the final state is somewhere in between the two (though not necessarily on the path itself). But what if we connect the two not directly to each other, but through the chaotic state? That is we add progressively more noise to the samples to make a bridge between the fully ordered numbers and a disordered state.

$$n'_{ij} = (1 - \alpha)n_{ij} + \alpha\eta \qquad \eta = \{-1, 1\}, \alpha \in [0, 0.1, \ldots, 0.5] \tag{4.4.2}$$



(a) *An RBM trained at fours and nines.*



(b) *An RBM trained at fours, nines and a mixture.*

Figure 4.14: ***MNIST*** *The end of the RBM-flow for an RBM trained at fours and nines and for an RBM also trained on a mixture of fours and nines.*

We used $\alpha \in [0, 0.1, \ldots, 0.5]$ with 450 samples per α per digit (adding to 2700 per digit).

Figure 4.15 shows the result of this. Importantly we observe that distributions again converge to the same distribution. This is almost reminiscent of stochastic resonance and thermal fluctuations, where adding some noise/fluctuations to the system drives it away from an unstable fixed point.

We have examined what happens when we artificially add samples to a sparse dataset. But we could also have looked at a sparser Ising dataset.

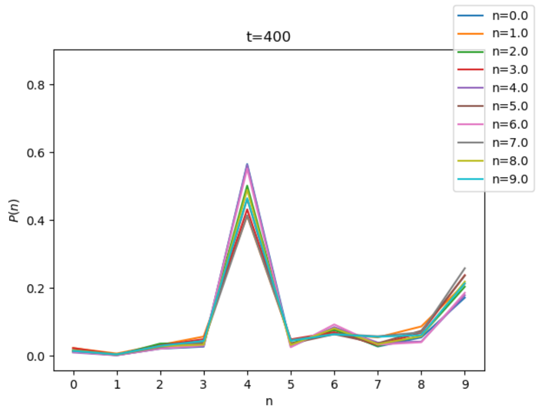


Figure 4.15: ***MNIST*** *The end of the RBM-flow trained on fours, nines with multiple levels of noise.*

4.5 2D Ising again

In the previous section we examined the MNIST dataset and hypothesised that multiple fixed distributions can exist when the dataset is sufficiently sparse. We can also try to apply this to the 2D Ising model. In particular the $T = 0$ distribution is naturally split into two separate cases: mean magnetisation $m = +1$ and mean magnetisation $m = -1$.

We encountered two particular problems. First of all, we are working with a relatively small subset of samples which means that there is a high chance of overfitting. As such we would like to introduce an equivalent notion to the test-energy of the MLP. We decided to use the following idea.

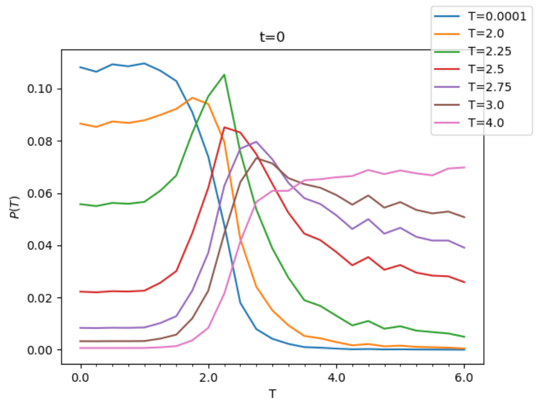
1. Update the parameters using the training samples.
2. Set the visible layer equal to a test-sample $v^{(n)}$
3. Find the hidden layer and then the new visible player $v'^{(n)}$
4. The test energy for that sample is then $E_{\text{test},n} = \sum_i (v_i^{(n)} - v_i'^{(n)})^2/2$
5. Repeat steps 2-4 for all test-samples, giving E_{test}.
6. Repeat steps 1-5 until test energy is minimised.

The second problem we ran into was visualising the two distinct distributions, since we only visualised the temperature previously and not the magnetisation. The magnetisation is easily found from the samples themselves, since m is just the mean spin value of the sample. Rounding m to the nearest decimal allows us to construct a $P(m)$ equivalent to the MLP probability curves.

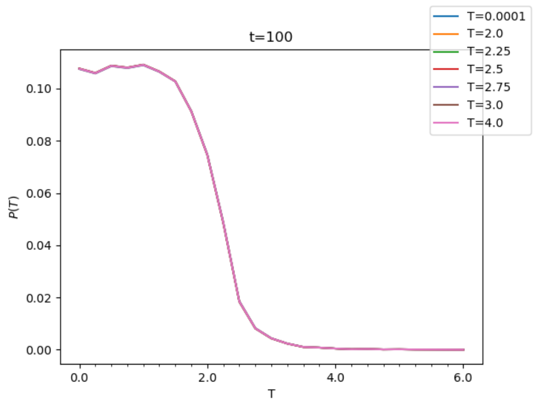
We start with a dataset of 2000 samples at $T = 0$, 1000 with mean magnetisation $m = +1$ and 1000 with $m = -1$. The results are summarised in figures 4.16 and 4.17. As expected at the end of the RBM flow the system is in the $T = 0$ state with about 50% $m = -1$ and 50% $m = +1$.

We also add a third distribution to this: 1000 samples with $m = 0$ selected from the $T \to \infty$ distribution. The resulting RBM-flow for the RBM trained on these 3000 samples is shown in figure 4.18. We see that multiple fixed distributions exist in this artificially sparse dataset.

We will combine all this in the next chapter.

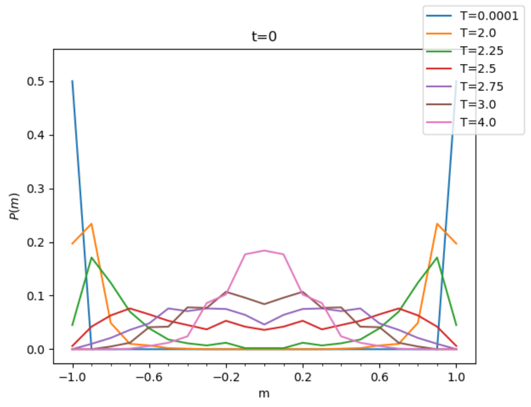


(a) *Start of the temperature RBM-flow*

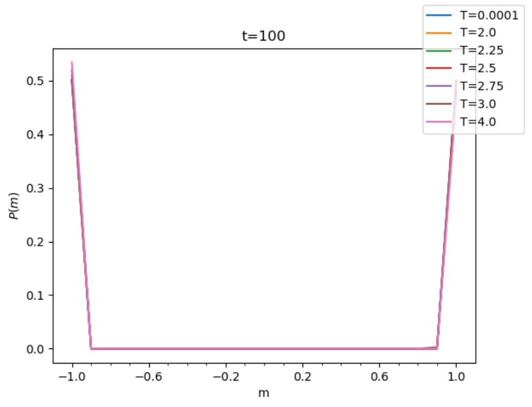


(b) *End of the temperature RBM-flow*

Figure 4.16: ***2D Ising*** *Temperature RBM-flow for an RBM trained on* $T = 0$.

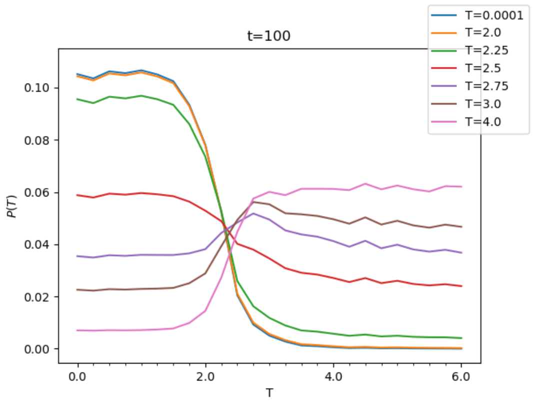


(a) *Start of the magnetisation RBM-flow*

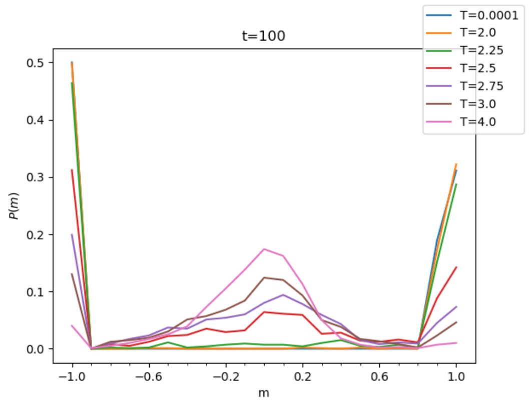


(b) *End of the magnetisation RBM-flow*

Figure 4.17: ***2D Ising*** *Magnetisation RBM-flow for and RBM trained on* $T = 0$.



(a) *End of the temperature RBM-flow*



(b) *End of the magnetisation RBM-flow*

Figure 4.18: ***2D Ising*** *Temperature and magnetisation RBM-flow for an RBM trained on $T = 0$ and $T \to \infty$ with $m = 0$.*

Chapter 5

Discussion and Conclusion

In the previous chapter we saw that the RBM-flow for the 2D Ising model flowed towards the critical point. The critical point stands out since it was attractive, even when it was on the edge of the training temperatures (figure 4.7) or not even part of the training samples (figure 4.8).

We also learned that if the phase space is sparsely sampled there can be multiple fixed points, but any sampling between these fixed points will allow the flows to converge. Sampling in a straight line or through the chaotic state surprisingly gave the same result.

Let us revisit what the Restricted Boltzmann Machine is designed to do, and how it achieves this.

The goal of an RBM is to reproduce the dataset on which it is trained. One does this by updating the RBM and sampling the visible layer after it has reached equilibrium. This immediately explains why, when the RBM is trained on a single temperature/number, there is an RBM-flow towards that temperature/number. Because after sufficient steps, the RBM is in equilibrium and any samples are distributed according to the trained distribution.

When the RBM is trained on a more complicated dataset (multiple temperatures/numbers) we have to remember that the RBM is trained by minimizing the Kullback-Leibler (KL) divergence:

$$KL(q||p) = \sum_{\{v_i\},\{h_a\}} q(\{v_i\}) \log \left(\frac{q(\{v_i\})}{p(\{v_i\},\{h_a\};\{\theta_k\})} \right) \tag{2.4.3}$$

and thus the properties of the KL-divergence impact how the RBM behaves.

We want to point out two important facts: setting $p \neq 0$ while $q = 0$ is not directly punished, on the other hand setting $p = 0$ while $q \neq 0$ is heavily punished (since $KL \to \infty$). As such we can expect that in local minima $p \neq 0$ even where $q = 0$. We can imagine this as regions around samples ($q \neq 0$) have some weight, while sufficiently far away do not. This explains why in a sparse dataset such as MNIST the distributions can't converge, since the samples are surrounded by a volume of $p = 0$. Adding additional samples in those areas (as done through the linear interpolation and the addition of chaos) fills the phase space with enough samples that there exist a path where $p \neq 0$ which allows the RBM-flow to happen.

Finally let us ask why specific distributions are fixed points (most notably the critical temperature). Research as shown that RBMs can be used for feature extraction, as a substitute for principle component analysis (PCA) and more[7][4]. Casting the RBM in the light of PCA and feature extraction we can decompose any sample into various features. The more a specific feature is encountered in the training dataset, the more important it is for the RBM to preserve information about this feature.

Near the critical point the correlation length ξ diverges, the system becomes scale invariant and effects of all length scales become important. But if we train the system on a range of different temperatures with a range of different correlation lengths, then the RBM must learn to reproduce the features corresponding to these different length scales. Features which inherently describe the physics at those length scales. Then when the RBM tries to generate a new sample with all those features, it looks like the system near the critical point since that's exactly where all those features are equally important.[7]

We can also look at this from the other way around, reproducing a sample with multiple features is a good method of decently reproducing samples with some of those features.

Furthermore the existence of multiple fixed points in the MNIST dataset which disappear when we artificially connect them also fits this picture. Starting in one of the Bulk phases expresses some of these features and suppresses others. When we connect them we create a state where all of the features are present, which acts as an attractive critical point.

Finally by also considering the magnetisation and artificially sparsening the two-dimensional Ising dataset we recreated the existence of multiple fixed points. This shows that sparseness can be a feature of under-sampled real physical datasets. It is furthermore also evident that the thermal fluctuations are key to connecting the various distributions to each other and allowing the RBM-flow to exist.

We would argue that an RBM is not actually performing an RG transformation. The reason why the RBM-flow is towards the critical point is because of the diverging correlation length at the critical point. The diverging correlation length at makes the critical point an attractive point since it shares features with samples at all other points in phase space (in the simple examples that we have considered).

Future work could examine these claims further. Examining a system with more than one critical point would also allow us to observe how the RBM-flow changes. We were also intrigued by the removal of the MNIST fixed points due to the introduction of noise and further research could be done on whether this can be used in other abstract datasets. Finally, further research could investigate whether the attractive nature of the critical fixed point could remove the fixed points even in large sparse datasets.

The KL-divergence ($KL(q_{\text{data}}||p_{\text{model}})$) punishes $p = 0$ where $q \neq 0$, while tolerating $p \neq 0$ where $q = 0$, which leads to the smoothing of the probability distribution and in turn allows the RBM flow to cross slightly sparse regions, as discussed above. We could, in theory[*], also have chosen to minimise $KL(p_{\text{model}}||q_{\text{data}})$ which would instead have punished $p \neq 0$ where $q = 0$ and tolerated $p = 0$ where $q \neq 0$. This creates an even sparser p which in turn could stop the RBM-flow.

The Jensen-Shannon Divergence:

$$\text{JS}(p, q) = \frac{1}{2}\left[\text{KL}\left(p \middle|\middle| \frac{p+q}{2}\right) + \text{KL}\left(q \middle|\middle| \frac{p+q}{2}\right)\right] \tag{5.1}$$

is the squared metric which, in principle, is the basis of Generative Adversarial Networks (GANs). It could be interesting to see whether a generative network trained through adversarial learning still supports an RBM-flow.

Finally, further research could also combine this with adding other regularisation terms to the RBM energy.

[*]In practice $\text{KL}(p_{\text{model}}||q_{\text{data}})$ is impossible to compute.

Appendix A

Choosing the correct cost function to your activation function

We have a dataset $\mathcal{D}$ which contains samples $\boldsymbol{x}^{(i)}$ and associated targets $\boldsymbol{y}^{(i)}$. We are trying to train a neural network using this dataset where the ouput neurons have a specific activation function f and we want to know which cost function is best.

A cost function must be positive definite: $C(\hat{\boldsymbol{y}}, \boldsymbol{y}) \geq 0$ and $C(\hat{\boldsymbol{y}}, \boldsymbol{y}) = 0 \Leftrightarrow \hat{\boldsymbol{y}} = \boldsymbol{y}$. This means that by minimising the cost function, we minimise the distance between $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$. Let us examine the derivative of the cost function with respect to some parameter θ:

$$\frac{\partial C}{\partial \theta} = \frac{\partial C}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \theta} = \left(\frac{\partial C}{\partial \hat{\boldsymbol{y}}}\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}\right)\frac{\partial \boldsymbol{z}}{\partial \theta} = \Delta \frac{\partial \boldsymbol{z}}{\partial \theta} \tag{A.1}$$

The derivative $\frac{\partial \boldsymbol{z}}{\partial \theta}$ depends on which parameter we are looking at, but z is linear in θ. We see that a good choice of cost function can linearise $\Delta = \frac{\partial C}{\partial \hat{\boldsymbol{y}}}\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}$.

Linear activation function

In the case of a simple linear activation function $\hat{\boldsymbol{y}} = f(\boldsymbol{z}) = \boldsymbol{z}$ we see that the simple squared error function gives:

$$\Delta = \frac{\partial C}{\partial \hat{\boldsymbol{y}}}\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}} = \frac{\partial}{\partial \hat{\boldsymbol{y}}}\left[\frac{1}{2}\left(\hat{\boldsymbol{y}} - \boldsymbol{y}\right)^2\right]\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{z}} = \hat{\boldsymbol{y}} - \boldsymbol{y} \tag{A.2}$$

Single sigmoid activation function

Now in the case of a single sigmoid activation function we see that:

$$\frac{\partial f(z)}{\partial z} = \frac{\partial}{\partial z}\left(1+e^{-z}\right)^{-1} = \frac{e^{-z}}{\left(1+e^{-z}\right)^{2}} = f(z)\big(1-f(z)\big) = \hat{y}\,(1-\hat{y}) \tag{A.3}$$

So if we have an cost function whose derivative is:

$$\frac{\partial C}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} \tag{A.4}$$

then we find that $\frac{\partial C}{\partial \hat{y}}\frac{\partial f}{\partial z} = \hat{y} - y$. Integrating this we find the cross-entropy:

$$C = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \tag{A.5}$$

This activation function is used when the output is the probability that the sample is part of some class: $\hat{y} = p(y=1)$

Tanh activation function

For a tanh activation function we see that:

$$\frac{\partial f(z)}{\partial z} = \frac{\partial \tanh(z)}{\partial z} = \frac{1}{\cosh^2(z)} = 1 - \tanh^2(x) = 1 - \hat{y}^2 = (1-\hat{y})(1+\hat{y}) \tag{A.6}$$

We see that if:

$$\frac{\partial C}{\partial \hat{y}} = \frac{1}{2}\left(\frac{1-y}{1-\hat{y}} - \frac{1+y}{1+\hat{y}}\right) \tag{A.7}$$

then $\frac{\partial C}{\partial \hat{y}}\frac{\partial f}{\partial z} = \hat{y} - y$ and so we find a modified cross-entropy:

$$C = -\frac{1}{2}\Big[(1+y)\log\left(1+\hat{y}\right) + (1-y)\log\left(1-\hat{y}\right)\Big] \tag{A.8}$$

which is unsurprising since $\tanh(x) = 2\sigma(2x) - 1$

SoftMax and CrossEntropy

In the case of multiple classes we can use the SoftMax activation function:

$$\hat{y}_m = p(y_m = 1|\boldsymbol{x};\boldsymbol{w}) = \sigma(z_m) = \frac{e^{z_m}}{\sum_{k=1}^{M} e^{z_k}} \tag{A.9a}$$

$$y_m = \begin{cases} 1 & y = m \\ 0 & \text{else} \end{cases} \tag{A.9b}$$

where $\boldsymbol{y}$ is a one-hot vector (a vector whose m-th value is 1 if the input belongs to class m and all other values are 0) associated with the input and $\hat{\boldsymbol{y}}$ is the output of the neural network. Let us look at a neural network with the Cross Entropy cost function per sample:

$$C = -\sum_m y_m \log(\hat{y}_m) + (1 - y_m)\log(1 - \hat{y}_m) \tag{A.10}$$

We have that:

$$\Delta_m = \frac{\partial C}{\partial z_m} = \frac{\partial C}{\partial \hat{y}_m}\frac{\partial \sigma(z_m)}{z_m} \tag{A.11}$$

The first term is rather simple:

$$\frac{\partial C}{\partial \hat{y}_m} = -\frac{y_m}{\hat{y}_m} + \frac{1 - y_m}{1 - \hat{y}_m} \tag{A.12}$$

The second term can be found using the quotient rule:

$$\begin{aligned}\frac{\partial \sigma(z_m)}{\partial z_m} &= \frac{e^{z_m}\sum_{k=1}^{M} e^{z_k} - e^{z_m}e^{z_m}}{\left(\sum_{k=1}^{M} e^{z_k}\right)^2} = \sigma^L(z_m) - (\sigma(z_m))^2 \\ &= \sigma(z_m)\,(1 - \sigma(z_m)) = \hat{y}_m\,(1 - \hat{y}_m)\end{aligned} \tag{A.13}$$

so we see that:

$$\begin{aligned}\Delta_m &= \left(-\frac{y_m}{\hat{y}_m} + \frac{1 - y_m}{1 - \hat{y}_m}\right)(\hat{y}_m(1 - \hat{y}_m)) \\ &= -y_m(1 - \hat{y}_m) + (1 - y_m)\hat{y}_m \\ \Delta_m &= \hat{y}_m - y_m \qquad\qquad (A.14)\end{aligned}$$

General

In general, if $f(\boldsymbol{z})$ is differentiable everywhere and $\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}} \neq 0$, then we can find the cost function as:

$$C = \int \frac{\hat{\boldsymbol{y}} - \boldsymbol{y}}{\frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}}} \, \mathrm{d}\hat{\boldsymbol{y}} \tag{A.15}$$

then we have that:

$$\Delta = \frac{\partial C}{\partial \hat{\boldsymbol{y}}} \frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z}} = \hat{\boldsymbol{y}} - \boldsymbol{y} \tag{A.16}$$

and thus:

$$\frac{\partial C}{\partial \theta} = (\hat{\boldsymbol{y}} - \boldsymbol{y}) \frac{\partial \boldsymbol{z}}{\partial \theta} \tag{A.17}$$

In the case of piece-wise continuous functions with a non-zero derivative everywhere (such as the leaky ReLU and ELU) then one could use a piece-wise cost function such that equation (A.17) holds almost everywhere.

For the leaky ReLU this then gives:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{z}) = \begin{cases} \boldsymbol{z} & \boldsymbol{z} \geq 0 \\ \alpha \boldsymbol{z} & \boldsymbol{z} < 0 \end{cases} \tag{A.18a}$$

$$C = \begin{cases} \frac{1}{2} (\hat{\boldsymbol{y}} - \boldsymbol{y})^2 & \boldsymbol{z} \geq 0 \\ \frac{1}{2\alpha} (\hat{\boldsymbol{y}} - \boldsymbol{y})^2 & \boldsymbol{z} < 0 \end{cases} \tag{A.18b}$$

where $0 < \alpha < 1$ for the leaky ReLU. While for ELU this gives:

$$\hat{\boldsymbol{y}} = f(\boldsymbol{z}) = \begin{cases} \boldsymbol{z} & \boldsymbol{z} \geq 0 \\ e^{\boldsymbol{z}} - 1 & \boldsymbol{z} < 0 \end{cases} \tag{A.19a}$$

$$C = \begin{cases} \frac{1}{2} (\hat{\boldsymbol{y}} - \boldsymbol{y})^2 & \boldsymbol{z} \geq 0 \\ \hat{\boldsymbol{y}} - (1 + \boldsymbol{y}) \log(1 + \hat{\boldsymbol{y}}) - \frac{1}{2}\boldsymbol{y}^2 & \boldsymbol{z} < 0 \end{cases} \tag{A.19b}$$

ReLU is usually coupled with a squared error.

The perceptron (or heavyside step function) is the only activation function shown in figure 2.1 for which gradient descent doesn't work.

Appendix B

Feed-Forward Neural Networks, Renormalisation group and Phase Transitions

In this thesis we have discussed the connection between the restricted Boltzmann machine and the renormalization group extensively. We have also looked into another possible connection between neural networks and RG, originally investigated by Bachtis and Aarts[2][3].

In this Appendix we will discuss our investigation into these papers. While we failed to reproduce their results, their method remains interesting and worth investigating. Here we examine the renormalization group using a neural network.

We examine a system described by a Hamiltonian H. Samples σ gathered at equilibrium occur with probability:

$$p_\sigma = \frac{e^{-\beta H_\sigma}}{\sum_{\sigma'} e^{-\beta H_{\sigma'}}} \tag{B.1}$$

and the expectation value of an arbitrary observable O is given by:

$$\langle O \rangle = \frac{\sum_\sigma O_\sigma e^{-\beta H_\sigma}}{\sum_\sigma e^{-\beta H_\sigma}} \tag{B.2}$$

Let us now take a pair of conjugate variable X and Y where X is extensive and Y is intensive. We can now define a new Hamiltonian H_Y:

$$H_Y = H - XY \tag{B.3}$$

The expectation value of a arbitrary observable O is then given[2] by:

$$\langle O \rangle \frac{\sum_\sigma O_\sigma p_\sigma^{-1} e^{-\beta H_{Y,\sigma}}}{\sum_\sigma p_\sigma^{-1} e^{-\beta H_{Y\sigma}}} \tag{B.4}$$

where p_σ is the probability used to sample the configuration in equilibrium. That is we first remove the natural weighting of the samples by deviding with p_σ and then reweight the samples with the new Hamiltonian. But p_σ is given by equation (B.1) and so we get[3]:

$$\langle O \rangle \frac{\sum_\sigma O_\sigma e^{\beta XY}}{\sum_\sigma e^{\beta XY}} \tag{B.5}$$

We can use this to calculate the expectation value for a modified system at non-zero Y by using configurations sampled at a given β and $Y = 0$.

We can now introduce the extensive observable $Vf(\cdot)$. Where $f(\cdot)$ is the output of a trained neural network. By varying Y around $Y = 0$ we can put greater importance on large/small $f(\cdot)$.

In particular, Bachtis and Aarts train a neural network to recognise the phase of a 2D Ising sample, outputting the probability that the sample is in the ordered phase $f(\cdot) = P_\sigma^{\text{ordered}}$. This has an important effect: by varying Y they put more/-less importance on the ordered phase, triggering a phase transition; $f(\cdot)$ acts as an order parameter.

Furthermore one can train two different neural networks:

- $f(\cdot)$ is trained on 2D Ising samples σ of 32×32 at inverse temperatures β.
- $f'(\cdot)$ is trained on renormalized 2D Ising samples from 64×64 to σ' of 32×32 at β' (renormalized through block spin transformation).

Note that all configurations are 32×32, the renormalized samples σ' were created with size 64×64 at inverse temperatures β' and then renormalized using block spin transformation to 32×32 (with a new, unkown temperature).

For these two systems we can find the expectation of the neural network function and it's susceptibility:

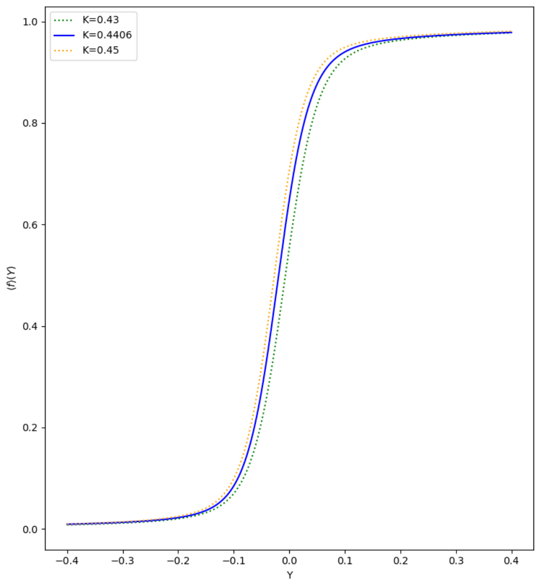
$$\langle f \rangle = \frac{\sum_\sigma f_\sigma e^{\beta V f_\sigma Y}}{\sum_\sigma e^{\beta V f_\sigma Y}} \tag{B.6a}$$

$$\chi_f = \frac{\partial \langle f \rangle}{\partial Y} = \beta V \left(\langle f^2 \rangle - \langle f \rangle^2 \right) \tag{B.6b}$$

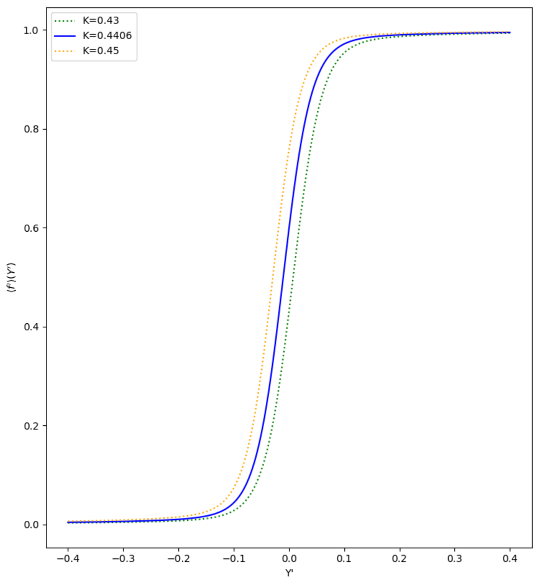
This is summarised in figures B.1 and B.2

Due to the scale invariant nature of the critical point it must be that $f(\beta_c) = f'(\beta_c)$. This allows one to find the critical point, by intersecting the two curves. Furhtermore Bachtis and Aarts conclude that[3]:

$$f(\beta') = f'(\beta) \tag{B.7}$$

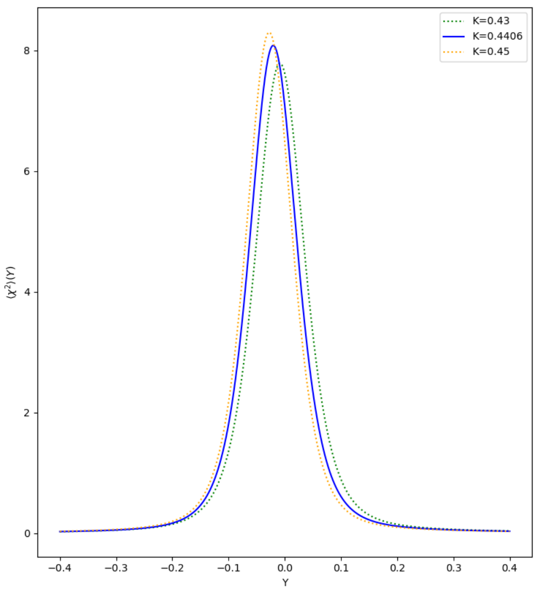


(a) *Normal system*

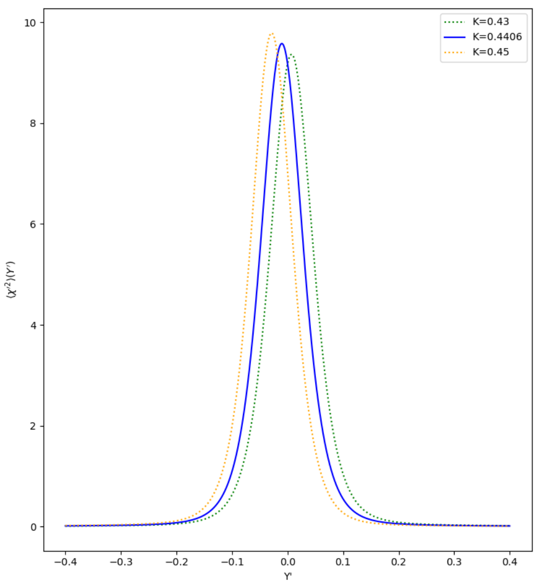


(b) *Renormalised system*

Figure B.1: *Mean neural network function* $\langle f \rangle$ *versus external field* Y *at inverse temperature* $\beta = 0.43; 0.440687; 0.45$ *(right to left). Reproduced from[3]*



(a) *Normal system*



(b) *Renormalised system*

Figure B.2: *Mean susceptibility of the neural network function* $\langle \chi_f \rangle$ *versus external field* Y *at inverse temperature* $\beta = 0.43; 0.440687; 0.45$ *(right to left). Reproduced from[3]*

which gives the renormalization mapping:

$$\beta' = f^{-1}(f'(\beta)) \tag{B.8}$$

which when found allows one to find the correlation length scaling exponent:

$$\nu = \frac{\ln(b)}{\ln \left.\frac{\mathrm{d}\beta'}{\mathrm{d}\beta}\right|_{\beta_c}} \tag{B.9}$$

with $b = 2$ in our case. We did not manage to properly reproduce the correct scaling exponent and we are still unsure why.

One could also do this with the Y variable, which gives:

$$Y' = f^{-1}(f'(Y)) \tag{B.10}$$

and

$$\theta_Y = \frac{\ln(b)}{\ln \left.\frac{\mathrm{d}Y'}{\mathrm{d}Y}\right|_{Y=0}} \tag{B.11}$$

The result from [3] was extremely close to the scaling exponent for the magnetisation. Again we did not manage to reproduce this result.

In this particular case we treat the output of a neural network as a physical observable. This observable is always accessible and we can use it to probe the scaling exponents of systems which could be difficult to examine.

Bibliography

[1] B. Alberts, A. Johnson, J. Lewis, D. Morgan, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, Taylor & Francis Group, LLC, 2015.

[2] Dimitrios Bachtis, Gert Aarts, and Biagio Lucini. Extending machine learning classification capabilities with histogram reweighting. *Physical Review E*, 102(3), sep 2020.

[3] Dimitrios Bachtis, Gert Aarts, and Biagio Lucini. Adding machine learning within hamiltonians: Renormalization group transformations, symmetry breaking and restoration. *Physical Review Research*, 3(1), February 2021.

[4] Bu, Yude, Zhao, Gang, Luo, A-li, Pan, Jingchang, and Chen, Yuqin. Restricted boltzmann machine: a non-linear substitute for pca in spectral processing. *Astronomy&Astrophysics*, 576:A96, 2015.

[5] Nigel Goldenfeld. *Lectures on Phase Transitions and the Renormalization Group*. Westview Press, 1992.

[6] Kevin T. Grosvenor and Ro Jefferson. The edge of chaos: quantum field theory and deep neural networks. *SciPost Phys.*, 12:81, 2022.

[7] Satoshi Iso, Shotaro Shiba, and Sumito Yokoo. Scale-invariant feature extraction of neural network and renormalization group flow. *Physical Review E*, 97(5), May 2018.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[9] John Kogut. An introduction to lattice gauge theory and spin systems. *Revieuws of Modern Physics*, 51:659–713, October 1979.

[10] Qianyi Li and Haim Sompolinsky. Statistical mechanics of deep linear neural networks: The back-propagating renormalization group. *CoRR*, abs/2012.04030, 2020.

[11] H. Lodish, A. Berk, C.A. Kaiser, M. Krieger, A. Bretscher, H. Ploegh, A. Amon, and K.C. Marrtin. *Molecular Cell Biology.* W.H. Freeman and Company, 2016.

[12] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124, 2019.

[13] Philip C. Nelson. *Biological Physics. Energy, Information, Life.* W.H. Freeman and Company, 2014.

[14] J.J. ter Hoeve. Renormalization group connected to neural networks. Bachelor's thesis, Universiteit Utrecht, June 2018.

[15] Dehao Zhang. Dimensionality reduction using t-distributed stochastic neighbor embedding (t-sne) on the mnist dataset. Towards Data Science. https://tinyurl.com/46u79e7r, 2020.