Universiteit Leiden
The Netherlands

Opleiding Informatica

Identifying the phase change of the Ising model using a Spiking Neural Network

Christian Steennis

Supervisors:
Evert van Nieuwenburg

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

03/07/2023

Abstract

Spiking Neural Networks (SNN's) have a lot of unexploited advantages like less power consumption and lower computational needs when used on a dedicated chip. These features make SNN's a promising asset for the future as they could be used in small chips with low power consumption. The Ising model is a mathematical model in statistical physics used to understand ferromagnetism. The model consists of spin latices that simulate the spontaneous magnetisation of a ferromagnetic material when its temperature drops. For this thesis the Ising model will be used to discuss the e ciency and e ectiveness of SNN's.

Contents

1 Introduction

In 1920 the Ising model was invented by Wilhelm Lenz. He gave the model as a problem to his student Ernst Ising who solved the one-dimensional Ising model in his PhD thesis [Bru67]. Ising proved with his solution that the one-dimensional model does not have a phase transition at any temperature [Isi24]. However in 1936 Rudolf Peierls showed that the model has a phase change for the two dimensional model [Pei36]. The two-dimensional Ising model was then analytically solved by Lars Onsager in 1945 [Ons44]. Using the Kramers-Wannier duality [KW41] Onsager calculated the critical temperature T_C.
The Ising model was first meant as a simple representation of a ferromagnetic material. Although, since the Ising model was solved it has gained a lot of popularity as a workhorse for statistical physics models, as well as its applications to phase changes in gasses, fluids, magnets and other materials [Cha20].
Onsager gave an exact solution for the two-dimensional Ising model using an analytical method, however there are several ways to compute the critical temperature. The temperature can be estimated using Finite Size Scaling or more recent machine learning techniques. Finite Size Scaling estimates T_C by calculating the magnetic moment of the spin lattice. And then scaling this method to larger lattices to estimate the actual T_C. The machine learning methods let a machine learning algorithm, like a neural network, guess the critical temperature by learning on known samples and guess the T_C by predicting unknown samples.
There are two methods to train a neural network, supervised learning methods and unsupervised learning methods. Both have been demonstrated before on the Ising model [CM17, Wan16]. Even a combination of supervised and supervised learning has been applied [vNLH17]. An example implementation of a Feedforward Neural Network using this last method can be found on the Github of the CRC183 summer school [1].
Lately a new generation of neural networks is on the rise. These so called third generation neural networks, the Spiking Neural Networks, try to mimic the human brain as close as possible. These types of networks are said to be way more energy efficient compared to Deep Neural Networks and Convolutional Neural Networks. On dedicated chipsets SNN's are reported to perform seven times better than GPU-accelerated deep-learning classification networks energy wise [Won17]. To test how well this new generation of Neural Networks performs on classical problems like the Ising model, the following research question arises:

How e cient are Spiking Neural Networks at identifying the magnetic phase change of the Ising model, in comparison to Feedforward Neural Networks?

1.1 Thesis overview

This is a bachelor thesis for LIACS, supervised by Evert van Nieuwenburg. To answer the research question I will start with an background on Spiking Neural Networks and the Ising model in the following section. Then follows a section for the methods that were used to train the network and get the results. Thereafter we will discuss the results. Finally we will conclude the efficiency of the Spiking network and name some items for future research.

[1] https://github.com/CRC183-summer-school/school_2021/blob/main/notebooks/03_learning_the_classical_ising_transition.ipynb

2 Background

2.1 SNN

Neural networks can be seen as artificial simulations of the human brain. An artificial neural network consist of fully connected layers of neurons, the neurons within the layer itself are not connected. Each neuron receives all of the outputs of the previous layer and multiplies them with a weight. The result of the sum of all these multiplications is then given as input to the activation function. This function determines the output value of the neuron [Kro08].
A SNN is a type of network that takes this principle one step further. This type of network endeavours to resemble the human brain even more closely than other types of artificial neural networks. Instead of analog values through which the neurons communicate a SNN uses spiketrains. These discrete sequences of zero's and one's represent whether or not there is a impulse current that flows between two neurons, like inside of the human brain.

2.1.1 Spiking Neuron

As mentioned in the previous section 2.1 a neural network is constructed of connected layers of neurons. In a Spiking Neural Network all of these neurons are, quite obviously, called spiking neurons.

First lets take a look at the biological neuron. The neurons in the human brain communicate with each other through links, these links are called synapses. The neuron itself has three distinct parts, the dendrites, soma and axon. The dendrites take input from other neurons and send it to the soma, there all the input currents are summed up and when this reaches a certain threshold an output signal is generated and sent through the axon to other neurons.

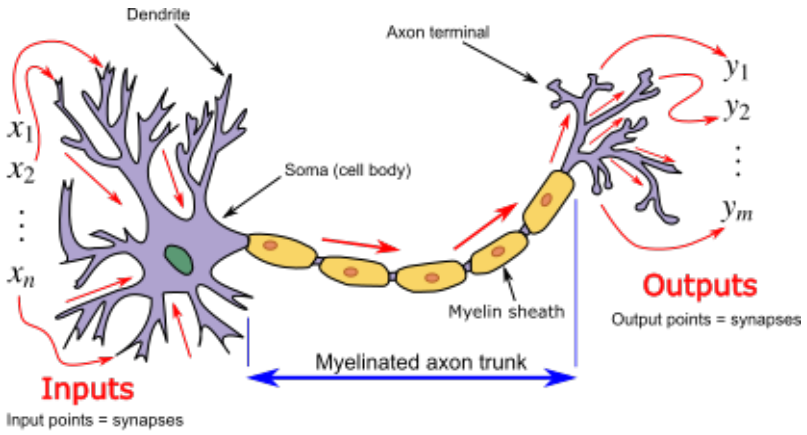


Figure 1: The biological neuron

In a spiking neuron these parts return as can be seen in figure 2, where the input spikes are comparable with the dendrites, the sum and membrane potential correspond to the soma and the output spike is equal to the axon.

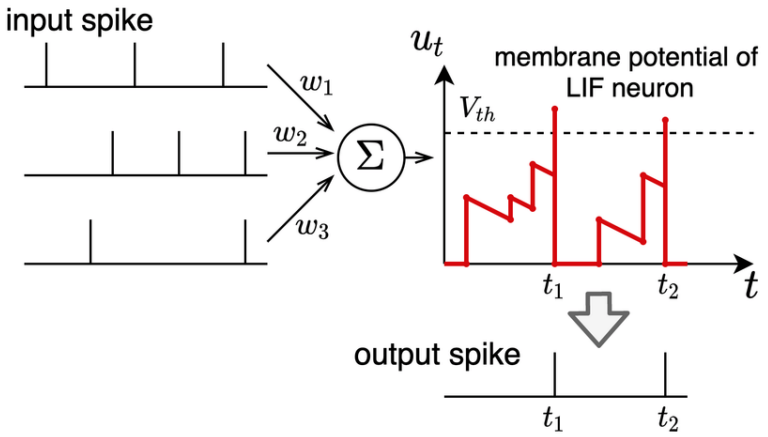


Figure 2: Artificial Spiking Neuron. Source: [KMH21]

The model described in figure 2 is a Leaky Integrate-and-fire neuron. However there exist a lot more neuron network models. They can be divided in two groups.
The first group consists of conductance-based models and the second group are models that generate an impulse at a certain threshold.

1. Conductance-based
    - Hodgkin-Huxley model
    - FitzHugh-Nagumo model
    - Morris-Lecar model
    - Hidmarsh-Rose model
    - Izhkevich model
    - Cable theory
2. Threshold models
    - Perfect Integrate-and-fire
    - Leaky Integrate-and-fire
    - Adaptive Integrate-and-fire
    - Fractional-order leaky Integrate-and-fire

All of these neuron models attempt to model the biological neuron. They also increase in complexity. Most of these models can be traced back to the Hodgin-Huxley and Perfect Integrate-and-fire model. The Leaky Integrate-and-fire is most commonly used in Spiking Neural Networks. So lets take a look at how these models work.

Hodgkin-Huxley The Hodgkin-Huxley neuron model, named after Hodgkin and Huxley [HH52] is based on a circuit representation, figure 3, of a cell membrane. The model is based on the movement of ions between the inside and outside of a cell. The Hodgkin-Huxley model forms the basis for all of the other conductance based models [Ski06].

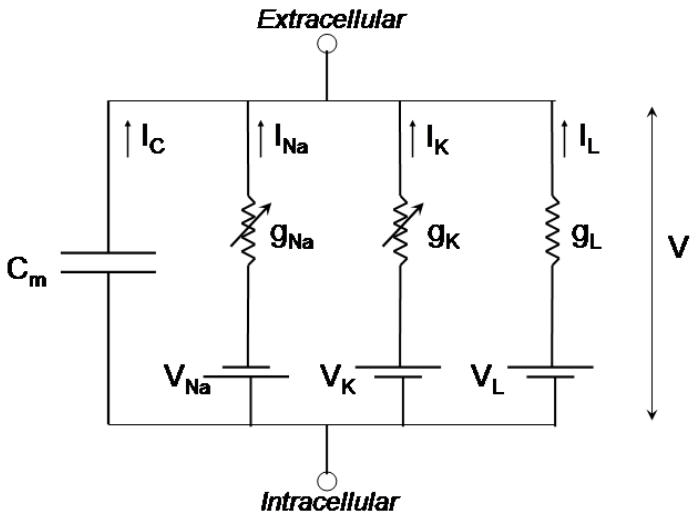


Figure 3: Circuit of the Hodgkin-Huxley model. Image source: [Ski06]

Perfect Integrate-and-fire One of the first models of the biological neuron is the Perfect Integrate-and-fire neuron, introduced by Lapique in 1907[Abb99]. This neuron model has a membrane potential and a threshold. When a spike is encountered at an input this is multiplied with a weight. This weight represents the strength of the connection between two neurons. The result of this multiplication can be interpreted as a current. This current is then added to the membrane potential of the neuron. When the membrane potential surpasses its threshold a output spike is generated.

Leaky Integrate-and-fire The Leaky Integrate-and-fire neuron model [EWN+21, p 9-11] looks a lot like the Perfect Integrate-and-fire model. However the Leaky model decreases its potential over time the potential by a small amount to its resting point. When the potential surpasses the threshold due to incoming spikes an output spike is generated.

$$U[T] > U_{\text{thr}} => S[T+1] = 1 \tag{1}$$

Next the membrane potential is reset to zero if the reset mechanism is enabled.

$$U[t+1] = \beta U[t] + I_{\text{syn}}[t+1] \quad R(\beta U[t] + I_{\text{in}}[t+1]) \tag{2}$$

I_{in} - Input current

U - Membrane potential

U_{thr} - Membrane threshold

R - Reset mechanism: if active, $R = 1$, otherwise $R = 0$

β - Membrane potential decay rate

2.1.2 Spiketrain generation

A Spiking Neural Network requires special input. The SNN endeavours to work just like the human brain. Therefore one would expect the input to be just like the input data the human brain receives. In the optimal situation SNN's would be getting a continues flow of spacio-temporal data same as we humans would through our eyes. This analog input is then converted to spiketrains before being forwarded through the network.
There are many different methods of converting analog input to discrete spiketrains. Some of these methods are rate coding, temporal coding, phase coding, burst coding, and delta modulation [GFES21]. SnnTorch currently supports three of these encoding methods, rate coding, latency coding and delta modulation [EWN+21].

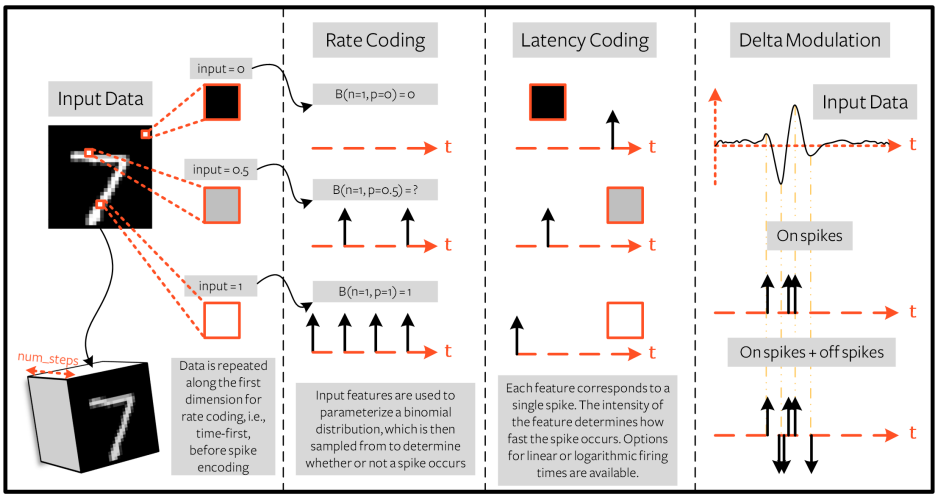


Figure 4: Spiketrain generation for the different encodings on an image from the MNIST dataset as example [EWN+21].

Rate coding The rate encoding distributes the spikes along the time interval by applying a Bernoulli distribution on the input. For this conversion the pixel intensity of the input should be between zero and one, so that the input can be interpreted as a probability for the Bernoulli function. This means that the probability of a spike occurring at time t can be denoted as shown in equation 3. Where S_t is a time step and thus a possibility for a spike to occur in the spiketrain. Repeating this over the desired number of steps a spiketrain for pixel $input_{ij}$ will emerge [Bri23, EWN+21].

$$X = input_{ij}, \ where \ 0 \quad input_{ij} \quad 1$$
$$P(S_t = s) = \begin{cases} X, & s = 1 \\ 1 \quad X, & s = 0 \end{cases} \tag{3}$$

Latency coding In the case of latency coding the timing of the first spike is the most important. This first spike occurs at an early time step for high intensity pixels and later in the spiketrain for low intensity pixels [EWN+21]. Because the pixel value is converted to a time that the first spike occurs in the spiketrain, the spikes are a lot sparser then when the spiketrain is rate coded.

Delta modulation For input data that changes over time Delta modulation is best suited. As for this technique a spike occurs when the difference between two consecutive values of the input is greater than a given threshold. This type of data conversion also allows for so called off spikes for negative changes [EWN+21].

2.1.3 Training

There are two methods for training Neural Networks. Supervised learning and unsupervised learning. With supervised learning the labels of the training data is supplied with the data. With unsupervised learning the labels are either ignored or not supplied. When training the network supervised to predict the right labels for the input data. The network should know how far off the prediction is from the targeted label.

Learning methods Neural networks learn by adjusting the weights of the network. The usual supervised method is backpropagation. On non-Spiking Neural Networks the updated weights are calculated using gradient descent. The gradients are calculated on the basis of a loss function. From the result of the loss function the gradient is taken and then propagated from back to front through the network with the gradient descent [Kos19].
For Spiking Neural Networks the same and other methods can be applied. Shadow training and backpropagation each have their pro's and con's [EWN+21]. With shadow training a non-Spiking Neural Network is trained, and then converted to a SNN. Backpropagation uses backpropagation through time, to update the weights of the network this method computes the gradients of the current time step as $\frac{L[t]}{W[t]}$ and applies it to the previous time steps as $\frac{L[t]}{W[t-k]}$.

Loss functions The error is calculated with a loss function. Common loss functions used on Artificial Neural Networks are mean squared error and cross entropy loss. Both of these functions can also be used in a SNN.
Mean squared error usually calculates square of the difference between the target value and predicted value. As Spiking Neural Networks work with spiketrains the network also outputs a spiketrain. Because this is not a single value it can not be directly used with a normal mean squared error function. So the output spiketrain needs to be summed up and the output classes are converted to a desired number of spikes per the given number of steps.
Cross entropy loss takes the entropy of each output class and measures how far off the prediction is from the true value of the class. This happens logarithmic so when a value is far off the target value the loss penalty is very high. For values that are close to what they should be the loss penalty is very low. For Spiking Neural Networks the spikes can either be summed up beforehand or the Cross Entropy Loss function can be applied to each time step in the spiketrain [EWN+21].

Gradient Once the loss is calculated using a loss function the gradient can be calculated for updating the weights of the model. The gradient function in equation 5 is most commonly used in Spiking Neural Networks. This is a Heaviside step function for the backward pass. Equation 4 shows how the chain rule connects the loss with the weight of a single neuron. This gradient determines how much that weight should change [EWN+21].

$$\frac{\delta L}{\delta W} = \frac{\delta L}{\delta S}\frac{\delta S}{\delta U}\frac{\delta U}{\delta I}\frac{\delta I}{\delta W} \tag{4}$$

$$\frac{\delta S}{\delta U} = \begin{cases} 1 & \text{if } U \quad U_{\text{thr}} \\ 0 & \text{if } U < U_{\text{thr}} \end{cases} \tag{5}$$

2.2 Ising model

The Ising model is a mathematical model that tries to understand ferromagnetism, the spontaneous magnetization of a material in particular. A ferromagnetic material consists of atomic spins, each having a magnetic moment in positive or negative direction. On high temperatures the spins are all randomly ordered in the lattice, for low temperatures the spins are all aligned. If the spins in a ferromagnetic material are aligned they work together as one magnet, making the ferromagnetic material magnetic. This so called spontaneous magnetization takes place when the temperature of the material drops below a critical point, also called Curie temperature T_C. The material then undergoes a disorder-order transition [Cip87].

The Ising model is a simple representation of such a ferromagnetic material and consists of a lattice of variables that represent the spins. The interaction between the spins is defined by the hamiltonian function in equation 6. Because of its simplicity the Ising model can be universally applied to other systems where a phase change occurs.

$$H(\sigma) = \sum_{ij} J_{ij}\sigma_i\sigma_j \quad \mu \sum_j h_j\sigma_j \tag{6}$$

2.2.1 Wolff Monte Carlo

The two-dimensional Ising model consists of two-dimensional lattices. These can be generated with a Wolff Monte Carlo simulation [Wol89, SvNLW21]. This simulation generates two-dimensional spin lattices of the Ising model resulting in the images shown in figure 5.

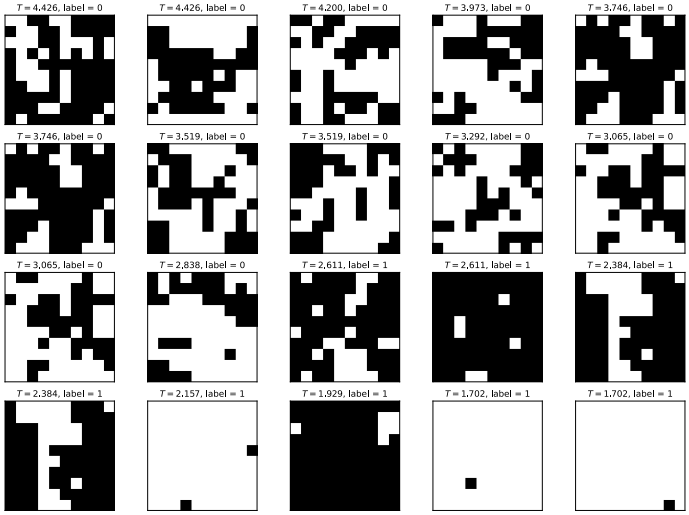


Figure 5: Lattices of two-dimensional Ising model generated by the code from [SvNLW21].

The samples are generated by clustering and flipping spins based on their direct neighbours and the temperature of the sample. This generation starts on a single spin, this spin, σ_x is flipped. Subsequently a bond is activated with its nearest neighbours with the following probability:

$$P(\sigma_x, \sigma_y) = 1 \quad exp(min\mathbf{f}0, \beta\sigma_x \quad [1 \quad R(r)]\sigma_y\mathbf{g}) \tag{7}$$

If the bond is activated σ_y is also flipped and added to the cluster. This continues until there are no neighbours of the cluster left that are unvisited [Wol89].

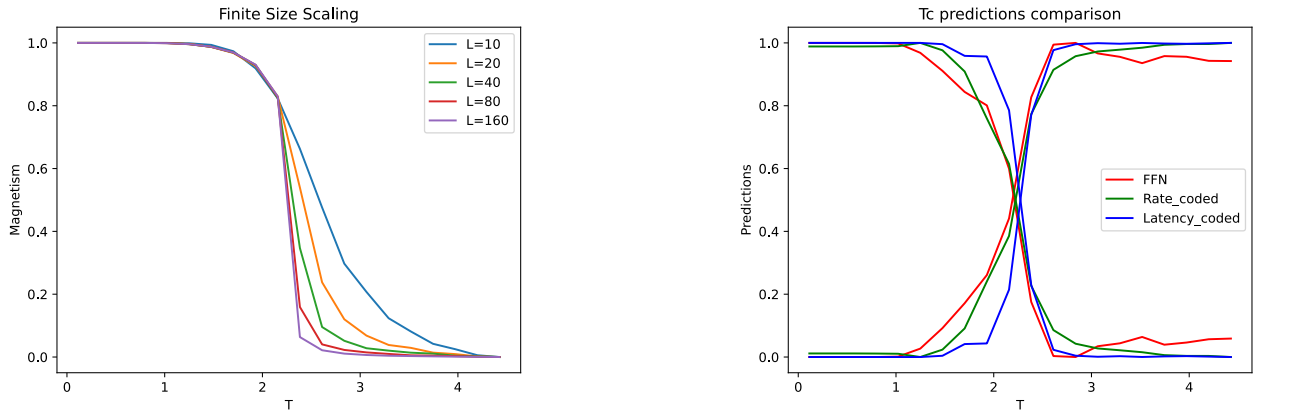
2.2.2 Extracting Curie Temperature T_C

As mentioned in the introductory section 2.1 the two-dimensional Ising model was solved by Lars Onsager. With his solution he determined a T_C 2.2692, equation 8. This can also be achieved by other methods. One method is known as Finite Size Scaling, other possibilities use machine learning.

$$\begin{aligned} T_c &= \frac{2J}{k \ln(1 + \sqrt{2})} \\ \frac{kTc}{J} &= \frac{2}{\ln(1 + \sqrt{2})} = 2.269185 \end{aligned} \tag{8}$$

Finite Size Scaling Using Finite Size Scaling the critical temperature can be estimated [Pri90]. For this technique the sum of the image is squared and then normalized to a 0 to 1 range for nicer plotting. The idea behind this technique is that when the image has a higher resolution the sum of the pixels is closer to the real magnetization of the sample. As can be seen in figure 6a, the higher the image resolution the steeper the angle becomes. Eventually the switching point of the graph should be at the exact value of T_C.

Blanking Another possibility is to train a neural network to guess the critical temperature. The sample data could not be labelled with the actual T_C since it should be unknown. Therefore a neural network is trained using known data samples, the samples where there is no question whether the sample is magnetic or not. These samples can be found in the extremes of the temperature range. Then the network will predict for of all of the data samples if they are magnetic or not. T_C should then be on the 50/50 point for both the magnetic prediction and the non-magnetic prediction. An example of this is depicted in figure 6b. To find the exact crossing point of the graph a sigmoid function can be fitted to the results and the intersection of the two sigmoids gives the predicted T_C.



(a) Finite Size Scaling to benchmark T_C.

(b) Predictions for various neural networks, with T_C at the crossing point.

Figure 6: Two methods for extracting T_C.

3 Methodology

The first step in the process is to build a Spiking Neural Network. The snnTorch[2] library is a well documented library for building SNN's. This library is based on PyTorch and extends this library by adding other types of neuron's like the Leaky Integrate and Fire type.[EWN+21] There exist other libraries especially built for spiking networks, like SpikeTorch, however snnTorch comes with a clear and concise documentation and a lot of examples. So for this reason snnTorch was the best choice for this project.

3.1 The model

SnnTorch lets you build a model for a spiking network in the same way as Torch would. But snnTorch has build in classes for creating a Spiking Neural Network. A number of neuron models, some of them from the list in section 2.1.1, are also implemented in the library.
For this project the Leaky Integrate-and-fire model seemed like the best model to use, as it is complicated enough to have a reasonably accuracy in modeling the biological neuron. But simple enough to understand and use in the correct way.
For the experiments a simple neural network was used. It consists of an input layer and a hidden layer of 200 LIF neurons and an output layer of 2 LIF neurons.

3.2 Training loop

A training loop for the SNN consists of a forward step a backward step and an optimization step. The input data is batched into batches of 128 images. The forward, backward and optimization steps are all done per batch. This means that the weights of the network are optimized after every batch. This training sequence is then repeated for a total of 20 epochs.

3.3 SNN on Ising

Now that there is a network model to train some data is needed. The Ising model's data for this project is generated with code from the CRC183 Summer School [SvNLW21]. This git provides functionality for generating training data for the two-dimensional Ising model for a given set of temperatures. The data generated consists of an L by L image with a pixel value of either -1 or +1. L is the linear size of the image. The number of samples per temperature can also be specified for the generation.

3.4 Spiketrains

A SNN handles data in the form of spiketrains. The spiketrains are generated during the forward pass of the training loop. This causes some delay in the training process, however it makes the model more noise resistant and dynamic, at least for the rate coding. Because of the Bernoulli distribution used for this coding scheme the spiketrains are random every time, even if the input image is the same.

To convert the input data to spiketrains rate encoding and latency encoding is used since the input

[2] snnTorch: https://snntorch.readthedocs.io/en/latest/index.html

data consists of static images. The Ising model consists of two-dimensional lattices where every pixel is either -1 or +1. However the encoding schemes only work with input data between 0 and 1. For this reason the input data had to be clipped to these values to fit the criteria for encoding. Therefore from now on when mentioning a pixel with value -1 we will say this pixel has value 0.

There are some properties of the input data and the encoding schemes that can lead to a problem in the training sequence. These properties are the way data is encoded by converting the input data pixel intensity as explained in section 2.1.2, and the fact that the input data of the Ising model only can adopt two values. These lead to a situation where only extreme spiketrains are generated. A rate coded spiketrain generated for a single spin, or pixel, of the Ising model, spikes either at every time step for positive pixel values, and spikes never for negative values. For the latency coded spiketrains the spike either occurs at the first time step for positive pixels and in the last time step for pixels with a negative value.
Interpreting the input as probabilities leads to only two possible options for spiketrains. For this reason the generation of spiketrains is no longer stochastic. And for interpretation as spike timing there are also only two different types of spiketrains generated. The other problem is that this translates to a static image being sent through the network at every time step. So this creates a problem where the time variant properties of the SNN are also not fully exploited by the two-dimensional Ising model. This makes it harder for the SNN update its weights and learn and ultimately also to predict the correct T_C.

3.5 Output

The output from the model consists of two spiketrains. One for each class, magnetic or non-magnetic. The spiketrains are then decoded following the same coding scheme as by which the input data was encoded. To predict T_C the blanking technique from 2.2.2 is used. For interpreting the output spiketrains as predicted classes, the spikes need to be summed up. The argmax can then be taken as the predicted class.

4 Experiments and Results

In this research the accuracy of the Spiking Neural Network is determined by comparing the predicted critical temperature T_C. The SNN will be compared to a Feedforward Neural Network where all parameters are the same as for the SNN. Furthermore the effect of different number of time steps and the effect of different lattice sizes will be tested. Also the effect of a solution for the problem from section 3.4 will be tested.

4.1 Identify T_C

With this experiment we will see which network is best at predicting T_C. For this experiment the networks are trained with 10x10 images. Again there are 200 neurons in the hidden layer and 2 outputs. The critical temperature will be determined using the blanking technique explained in section 2.2.2.

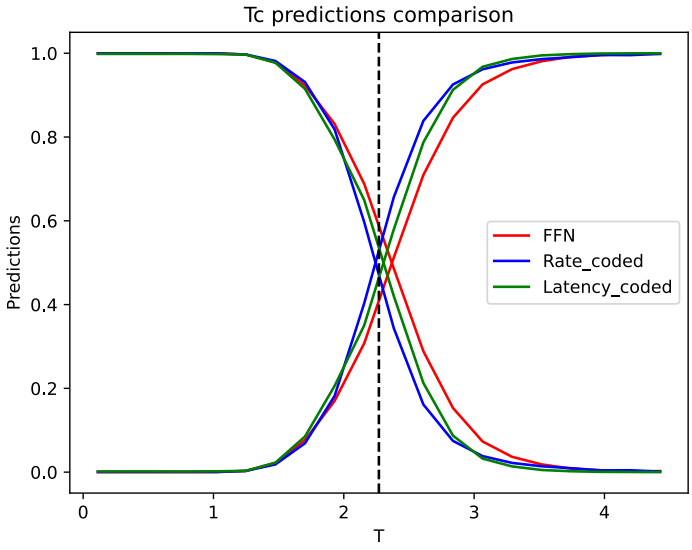


Figure 7: Comparison of predicted T_C's for L=10.

In figure 7 the results can be observed. As can be seen some nice crossing points have emerged. All are fairly close to the actual T_C. However the Spiking Neural Networks, as well for a rate encoding as for a latency encoding, both are closer than the Feedforward Neural Network. This leads us to believe that a SNN performs better than a Feedforward network.
We observe in figure 8 that the rate coded network finds it harder to learn using the lower temperatures of input data for a larger image scale. This might be because the images of the Ising data only exist of zero's or one's. When the lattice only has zero values the generated spiketrains are empty for the rate encoding and they all have a spike at the very last time step for latency coding. This leads to completely inactive neurons in the hidden layer, making it hard or even impossible for the SNN to learn. Because when there is little to no input there are also barely spikes in the spiketrains, therefore the membrane potentials threshold will never be reached and there will never be an output spike.

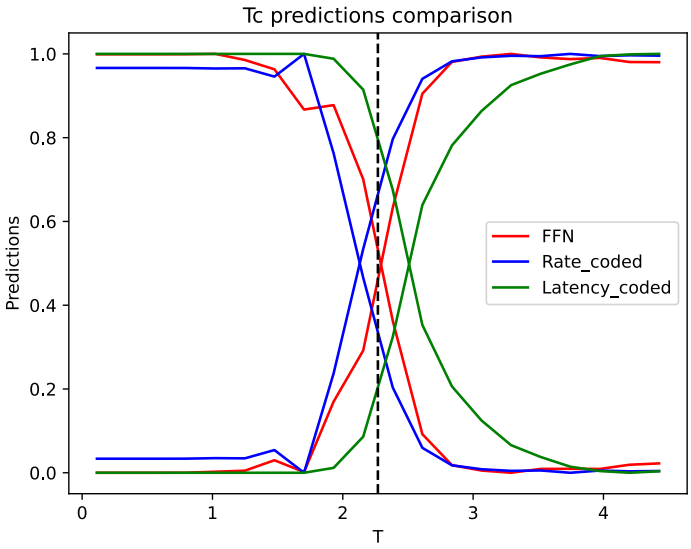


Figure 8: Comparison of predicted T_C's for L=20.

The other possibility is that all of the values in the lattice are one. In this case all of the neurons are active and should reach their membrane potentials. Now there actually happens something in the network therefore a loss can be calculated and the gradients can be backpropagated to update the weights so that the input spikes lead to a correct prediction. Because for the low temperatures images with zeros and one's are alternated the networks learns a little bit but not enough to have the needed accuracy to correctly predict T_C.
Looking at the results in figure 8 this negatively affects both the latency coded and rate coded SNN's.

4.2 Averaging

To solve the problem from the previous section and the section about spiketrains, section 3.4, the input pixel values need to be adjusted. Instead of values of 0 and 1 after clipping, the values should be in between 0 and 1. This way the rate encoding is more stochastic and also for latency encoding the properties of SNN's are being used in a better way.
When converting an Ising configuration to an image with other values it is important to make sure that the new two-dimensional spin lattice still represents the same magnetism. Therefore the pixels of the new image should assume the average itself and of the pixels directly above, beneath, left and right of them. This should maintain the right magnetism since each spin in the Ising model interacts with its direct neighbours.

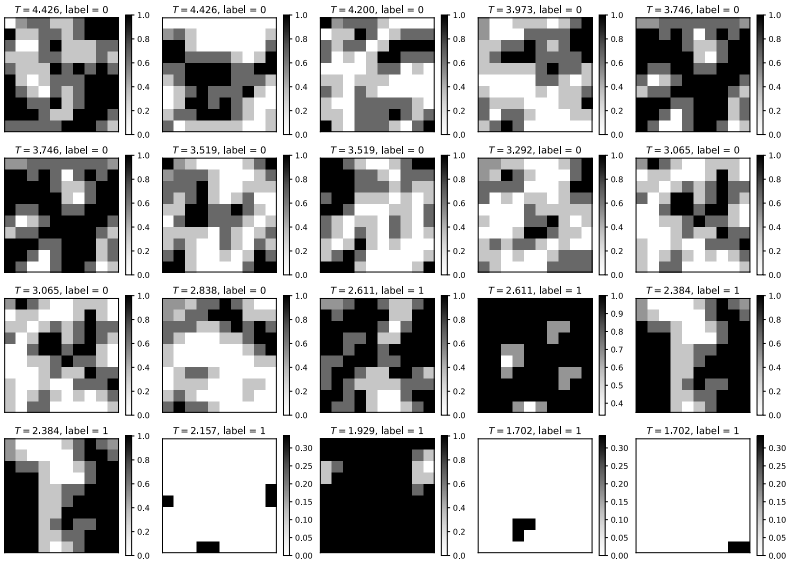


Figure 9: Training samples generated by code from [SvNLW21] after applying average values.

Now that the values are averaged over the neighboring pixels, every pixel can adopt five values instead of two. The samples do not resemble the earlier samples, but still the magnetism of the sample should be the same. These new values of the input data can be interpreted as probabilities, this means the spiketrains should differ more from each other and are based on their neighbours.

	Feed Forward		Rate coded		Latency coded	
	Predicted T_C	Diff to T_C	Predicted T_C	Diff to T_C	Predicted T_C	Diff to T_C
Averaging	2.2102	0.0590	2.2573	0.0119	2.3408	0.0716
No averaging	2.1999	0.0693	2.2824	0.0132	1.9150	0.3542
Increase %		0.149%		0.099%		0.798%

Table 1: Results of predicted T_C with and without averaging, averaged over 10 repeats with L=10, hiddens=200, n_steps=25

The results of the averaging technique can be observed in table 1. These results show that indeed the predictions of the networks are closer to the actual T_C then before. What is interesting is that it almost makes no difference for a rate coded SNN. It is not necessarily strange that the effect is bigger for latency coding since the spike times now have 5 different positions instead of two. The small increase for rate coding can be explained by the fact that the guess of the network already is quite close to the real value for T_C.

Renormalization Group Another solution to the problem would be Renormalization Group [Ash12]. This technique groups the two-dimensional lattice into smaller groups of for example 3 x 3 pixels. Then the dominant value is taken as the new value of the block. The theory behind Renormalization Group is quite complicated and goes beyond the scope of this thesis, however its worth pointing out that due to the scale invariance this method ensures that the temperature of the renormalized image and the original are the same.
The groups can also be converted to probabilities instead of taking the majority of the block. Then the number of positive spins is divided by the total number of spins in the block. This number can then be used as probability for the rate encoding and as spike time for the latency encoding.

4.3 Number of steps

The number of steps for which the forward step is repeated in a SNN has a big impact on the accuracy of the network. To determine how big this impact actually is, the network predicts T_C for Spiking Neural Networks trained with an increasing amount of time steps. The results are averaged over 5 repeats with newly generated 10x10 input data for each repeat.

As can be seen in figure 10 the influence of the number of steps has a much greater impact on a latency coded SNN than on a rate coded one. This effect occurs because the location of the first spike in the latency code is dependent on the length of the number of time steps. If it takes too long for a spike to occur in a spiketrain, the neurons in the hidden layer will stay inactive. The membrane potential of a Leaky Integrate-and-Fire neuron is decaying at every time step. This means that the membrane potential will have reached zero, or almost zero, again before receiving a new spike. Because of this the threshold will never be reached, and there will never be an output spike in the hidden layer. So none of the output classes is activated, resulting in an inability to learn.

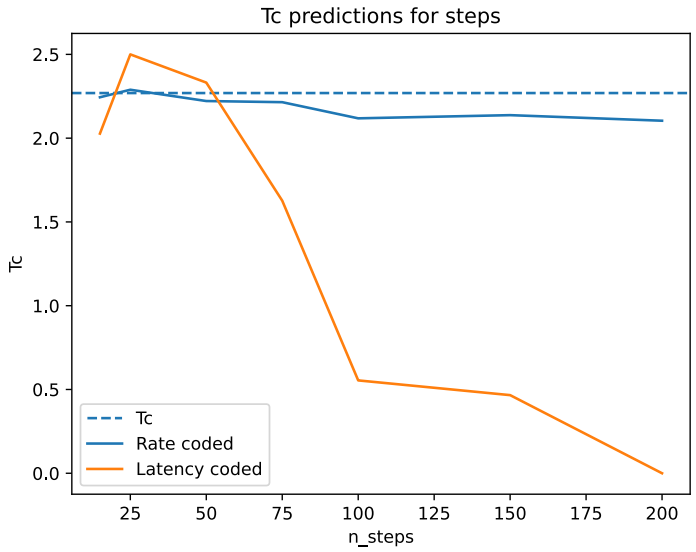


Figure 10: Results of predicted T_C for different spiketrain sizes, averaged over 5 repeats with L=10, hiddens=200

For the rate coded SNN also some diminishing effect can be seen. A reason for this can be that there are too many spikes in the spiketrain since a +1 pixel creates a spiketrain that fires at every time step. This causes the network to overtrain on the input data. And since the input data only contains the extreme samples as explained in section 2.2.2 this occurs a lot for the lower temperatures. This means the SNN is slightly off at predicting the exact T_C for the samples close to T_C with longer spiketrains.

4.4 Lattice size

According to the Finite Size Scaling effect [Pri90] the bigger the input lattice the closer the prediction should be to the actual T_C. To see if this also the case with Spiking Neural Networks a comparison is made between various image sizes. The results in figure 11 show the results of the average of 3 repeats and 4 different lattice sizes.

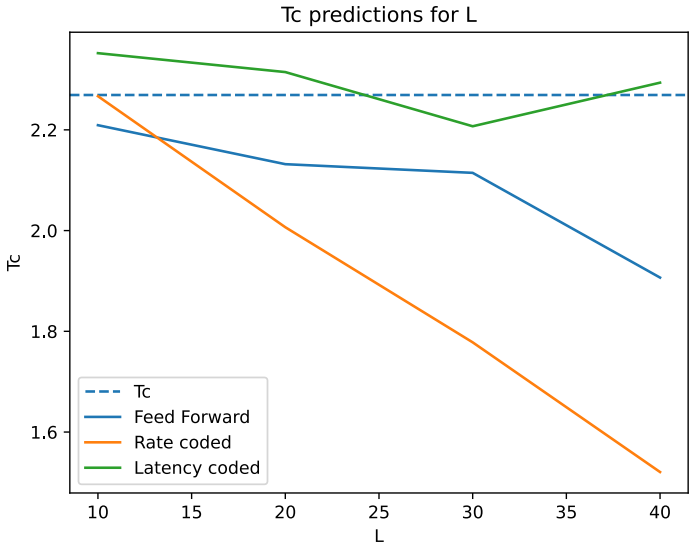


Figure 11: Results of predicted T_C for different lattice sizes, averaged over 3 repeats with hiddens=200

In the results shown in figure 11 the Finite Size Scaling effect can indeed be observed for the latency coded SNN. For the other networks however this effect seems to be reversed. The Finite Size Scaling effect in a latency coded SNN can be explained by the fact that the input lattices are bigger so there is more data. Therefore when there are many spins with a value of 0, the spikes occur in the last time step all at once. These spikes can then still ensure the non-magnetic class will fire once and the magnetic class will fire never. Therefore leading to a correct prediction for the lower temperatures after learning.
The reason this effect is not visible for the rate coded SNN and the Feedforward Neural Network might be that they had an unfortunate set of data samples, since only 3 repeats were possible due to time limitations this bad luck is not compensated with different repeats. This unfortunate data would have a lot of data samples with a lot of zero probability spike trains. This causes very little variance in the spike trains for lower temperatures. So when the lattice is bigger this effect is increased. As a result a lot of neurons in the hidden layer will be inactive and learning difficulty is increased.

5 Related Work

In this section we will discuss previous work related to this thesis. A research on learning the Ising model's critical temperature by guess was conducted by the superviser of this research Evert van Nieuwenburg [vNLH17]. In this research a Feedforward Neural Network is trained to predict the magnetism of generated images of spin lattices. This research then guesses the T_C by guessing the T_C at values known to be inaccurate. For each of these guesses new data is generated using the same Wolff Monte Carlo algorithm as was used in this work. Next the the accuracy on these datasets is plotted. This then creates a nice W shape. On the middle peak of the W the predicted T_C can be found.
The learning method used in this research is a combination of supervised learning and unsupervised learning. The actual T_C is not used for learning however for learning the critical temperature of the data generated at a guess the labels are provided, although these are not the labels corresponding to the actual Ising model.
Another research related to this thesis covers Spiking Neural Networks and how they work [EWN+21]. In this paper the working of SNN's is discussed and the pros and cons of different encoding methods, loss functions and weight update methods are discussed. This is relevant to this thesis because we are also reporting the effectiveness of Spiking Neural Networks.

6 Conclusions and Further Research

In this research we have investigated how efficient Spiking Neural Networks are at identifying the critical temperature T_C of the phase change of the Ising model. Our results have shown that Spiking Neural Networks are more accurate at predicting T_C.
The reason for writing this thesis was the possible positive effect of using Spiking Neural Networks on the Ising model, however we have also found some limitations. We first suspected that the input data of the Ising model would be perfect for use with SNN's. This approach used input data as static images as generated by the Monte Carlo Ising simulator. As it turned out, it is hard for SNN's to make good use of the spiketrains of SNN's when there is a possibility for data to be all zero.

To overcome this issue we suggest investigating the possibility of using Renormalization Group to convert the spin lattices back to probabilities. Originally the Renormalization Group technique chooses the majority of a group as the new value for the Renormalized pixel. However instead of choosing a concrete value a probability can be generated from the values inside a block. So for example if there are 4 pixels with value +1 in a 3x3 block there is a 4/9 chance that the Renormalized block has value +1. Our results have shown that the comparable technique of taking the average over neighbouring spins improves the accuracy of the SNN. Therefore we believe that Renormalization Group is a solution that will have advantages when applied on the input data of an SNN and should be further investigated.

In this thesis all values are clipped to fit between 0 and 1. Another possibility is to clip between a low floating point, like 0.2, and 1. Now the coding for low pixels is not as extreme as it was. So when tested against the averaging method used in this project this might be a better solution for the inactive neuron problem.
Another approach that can be further researched is using the probability of flipping spins from the Wolff Monte Carlo algorithm. If every spin takes on the value of the probability instead of just -1 or +1. These values can be taken as spike times or probabilities for latency and rate coding respectively.

In the end we can conclude that when using Spiking Neural Networks in the common way all Artificial Neural Networks are used leads to some performance gain. However there is more potential in further researching ways in which Spiking Neural Networks can be used to interact as efficient as possible with the properties of the Ising model.

To answer the research question of how efficient SNN's are in comparison to Feedforward Neural Networks, we can say that they are more accurate at identifying the critical temperature of the Ising model. Because there are some drawbacks with respect to the execution time of the forward pass we can not say the SNN is more efficient, at least not in its current implementation. Another item to mention is the importance of choosing the right encoding scheme. Since rate encoding seems to work better for small system sizes and latency coding works better for large system sizes.

Overall Spiking Neural Networks appear to be a good alternative to a Feedforward Neural Network. Depending on the chosen encoding, the amount of time steps within a spiketrain and the size of the input, the SNN even seems to perform better at classification of the Ising model than a Feedforward Network. Because of this and its computational advantages we believe that Spiking Neural Networks will be used in the future for all kinds of tasks currently performed by Feedforward Neural Networks.

The code used in this project can be found on https://github.com/CSteennis/BscThesis

References

[Abb99] L.F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5):303–304, 1999.

[Ash12] Doug Ashton. The renormalisation group, Apr 2012.

[Bri23] Brilliant.org. Bernoulli distribution, 2023.

[Bru67] Stephen G Brush. History of the lenz-ising model. *Reviews of modern physics*, 39(4):883, 1967.

[Cha20] Jeffrey Chang. Notes on statistical mechanics ii (physics 171), Sep 2020.

[Cip87] Barry A Cipra. An introduction to the ising model. *The American Mathematical Monthly*, 94(10):937–959, 1987.

[CM17] Juan Carrasquilla and Roger G. Melko. Machine learning phases of matter. *Nature Physics*, 13(5):431–434, 2017.

[EWN+21] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*, 2021.

[GFES21] Wenzhe Guo, Mohammed E. Fouda, Ahmed M. Eltawil, and Khaled Nabil Salama. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. *Frontiers in Neuroscience*, 15, 2021.

[HH52] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[Isi24] Ernst Ising. *Beitrag zur theorie des ferro-und paramagnetismus*. PhD thesis, Grefe & Tiedemann Hamburg, 1924.

[KMH21] Hiromichi Kamata, Yusuke Mukuta, and Tatsuya Harada. Fully spiking variational autoencoder, 09 2021.

[Kos19] Simeon Kostadinov. Understanding backpropagation algorithm, Aug 2019.

[Kro08] Anders Krogh. What are artificial neural networks? *Nature Biotechnology*, 26(2):195–197, 2008.

[KW41] Hendrick A Kramers and Gregory H Wannier. Statistics of the two-dimensional ferromagnet. part i. *Physical Review*, 60(3):252, 1941.

[Ons44] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3-4):117, 1944.

[Pei36] R. Peierls. On ising's model of ferromagnetism. *Mathematical Proceedings of the Cambridge Philosophical Society*, 32(3):477–481, 1936.

[Pri90] Vladimir Privman. *Finite size scaling and numerical simulation of statistical systems*. World Scientific, 1990.

[Ski06] F. K. Skinner. Conductance-based models. *Scholarpedia*, 1(11):1408, 2006. revision #125663.

[SvNLW21] Markus Schmitt, Evert van Nieuwenburg, and Guiseppe Carleo Lei Wang. Crc 183 summer school "machine learning in condensed matter physics", 2021.

[vNLH17] Evert P. L. van Nieuwenburg, Ye-Hua Liu, and Sebastian D. Huber. Learning phase transitions by confusion. *Nature Physics*, 13(5):435–439, 2017.

[Wan16] Lei Wang. Discovering phase transitions with unsupervised learning. *Physical Review B*, 94(19):195105, 2016.

[Wol89] Ulli Wolff. Collective monte carlo updating for spin systems. *Phys. Rev. Lett.*, 62:361–364, Jan 1989.

[Won17] William Wong. Brainchip enters ai territory with spiking neural network, Sep 2017.