Universiteit Leiden
The Netherlands

Opleiding Informatica

Detecting Phase Transitions

using Forward-Forward Neural Networks

Jelle van Cappelle

Supervisors:
Evert van Nieuwenburg & Felix Frohnert

BACHELOR THESIS

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

05/08/2023

Abstract

Phase transitions in physical systems are an ongoing field of research in which machine learning has proved increasingly useful over the past decade. Many different machine learning methods of studying phase transitions have been explored, including neural networks. All major neural network approaches so far have used backpropagation, however, recently a new training procedure called the *Forward-Forward* algorithm (FFA) has been developed. In this thesis, two approaches using FFA for training are demonstrated. A supervised classifier is able to learn the phase transition of the Ising model. An unsupervised discriminative network is shown to correctly identify the phase transition, and to learn the order parameter of the Ising, q-state Potts, and transverse-field Ising models. Experiments on the many-body localization model demonstrate some limitations of the current design of FFA.

Contents

1 Introduction

Phase transitions are an active area of research in condensed matter physics. Many different kinds of phase transitions exist: the well known solid, liquid and gas phases of most matter; (anti)ferromagnetic phases; as well as more exotic quantum mechanical phases, such as superconductivity, superfluidity, and many more. There are still open questions regarding the phase transitions of some quantum mechanical systems. Progress in this area could lead to useful discoveries with relevance to, for example, quantum computing or superconductive materials.

Like many fields over the past decade, condensed matter physics research has experienced an increase in the use of machine learning techniques. Many new techniques of finding phase transitions using machine learning have emerged, both supervised and unsupervised, ranging from 'simple' methods like PCA to more elaborate deep learning techniques. Most of these techniques have been successful, with some unsupervised techniques in particular being able to correctly identify phase transitions without any prior knowledge of the physical model under study.

So far, virtually all research on identifying phase transitions using neural networks has used backpropagation as the training algorithm. Backpropagation solves the credit assignment problem for neural networks by repeatedly applying the chain rule to find the partial derivative of an error function with respect to each of the weights in the network. Recently, a new solution to the credit assignment problem has been developed: the Forward-Forward algorithm [1] (FFA) for training neural networks. FFA solves the credit assignment problem locally to each individual layer of the network, which removes the requirement imposed by backpropagation that the network must be completely differentiable, from input to output.

The goal of this thesis is to investigate if the success of previous machine learning approaches to identify phase transitions can be repeated using the Forward-Forward algorithm. To this end, an FFA-based classifier is demonstrated to learn the phase transition of the Ising model in a supervised manner. An unsupervised FFA-based discriminative network is demonstrated to correctly identify phase transitions in the Ising model, the q-state Potts model and the transverse-field Ising model. Additionally and unrelated to phase transitions, an improvement to the Forward-Forward algorithm is presented in the form of a *peer normalization threshold*.

1.1 Related work

The application of machine learning to phase transitions is by no means a new idea: many previous works have already applied a multitude of techniques, both supervised [2–10], and unsupervised [10–20], to a wide range of physical models. Previously explored methods include: supervised neural networks [2–8], including recurrent networks [6] and convolutional networks [8]; support vector machines [9, 10]; principal component analysis (PCA) [10–14]; diffusion maps [15]; auto-encoders [12, 13, 16]; learning by confusion [17, 18] and prediction-based methods [19, 20].

Currently, the most prevalent supervised method using neural networks consists of classifying samples from a physical system by their phase. The classifier is often trained in well-known regions of the tuning parameter and then used to extrapolate to the unknown regions [2, 3].

Some unsupervised approaches are able to find the critical point without any prior knowledge, other than that there exists a phase transition. The general idea behind such unsupervised neural network approaches is some notion of ‘confusion’ arising from the similarity of samples within the same phase. This is exactly the motivation behind learning by confusion [17, 18] and prediction-based methods [19, 20]. In learning by confusion, a classifier is trained on samples that have been labelled according to some guess for the critical tuning parameter. The closer the guess is to the critical point, the more accurate, i.e. less confused, the network gets. This property can be used to systematically converge to an estimate for the critical point. Predictive methods instead train a network to predict the tuning parameter from any given sample. The physical properties of the samples change much quicker near the critical point compared to elsewhere, and so do the predictions from the trained network, allowing the critical point to be identified.

1.2 Thesis overview

The Forward-Forward algorithm is summarised in section 2. An existing experiment demonstrating the ability to classify handwritten digits is repeated in section 3. Section 4 covers the physical models under study, while section 5 describes the algorithms used to simulate these models to generate a dataset.

The existing approach of learning the phase transition from labelled data [2, 3] is repeated using FFA in section 6. In a paper on prediction-based methods [19], the authors suggest using the hidden representation learned by a predictive model as a nonlinear dimensionality reduction, which can be used as input for clustering - or other methods - to discover phases. Section 7 proposes an unsupervised FFA-based method that is very similar to this idea.

Finally, sections 8, 9 and 10 present the conclusion, suggestions for future work and acknowledgements, respectively. Appendix A derives a definition of the FFA objective function, appendix B describes the implementation of FFA in more detail, and appendix C covers an extensive hyper-parameter study of FFA on MNIST. Finally, appendix D explores how the method proposed in section 7 can be simplified in future work.

2 The Forward-Forward algorithm

The Forward-Forward algorithm (FFA) for training neural networks [1] is a recently developed alternative to backpropagation [21]. Motivated by the idea that global backpropagation of error gradients is not a plausible explanation for the way biological neurons learn [22, 23], it uses training objectives that are local to each individual layer in the network.

Learning is achieved through two forward passes: a *positive* pass and a *negative* pass. During the former, positive i.e. real data is fed through the network, and each layer has the objective of maximizing its *goodness*, i.e. its sum of squared activities. The opposite happens during the negative pass: negative i.e. corrupted or synthesized data is fed through, and each layer has to minimize its goodness. The output activity vector of each layer is normalized w.r.t. its goodness (therefore also w.r.t. its length), such that the next layer cannot derive whether the data was positive or negative from the length of the input vector, but only from its direction.

Intuitively, this means that each layer learns features of its input that are unique to positive data. In a successfully trained network, these features will activate during the positive pass, thus providing a high goodness, but remain deactivated during the negative pass. What exact features a network learns depends on how positive and negative data are generated. Ideally, the negative data will be generated in such a way that it can only be distinguished from positive data through features that are of further interest. For example, consider images of handwritten digits that have a class-label attached. Correctly labelled images could be used as positive data, and incorrectly labelled ones as negative data. To distinguish positive from negative, a Forward-Forward (FF) network would have to learn features related to the classes of handwritten digits in order to know whether they have been labelled correctly. This indeed works in practice, as demonstrated in Geoffrey Hinton's original paper describing FFA [1] and demonstrated again in section 3.

The most important difference with backpropagation is that the per-layer training objective forces each layer to learn non-trivial features. In principle, a network of ten equally sized layers trained with backpropagation could have three layers that learn some non-trivial features, and seven layers that just pass their inputs directly along to their outputs. In a network trained with FFA, this would never be the case, provided that the training was successful.

The flip side to this point is that in order for each layer to learn non-trivial features, the outputs of the previous layer have to be normalized with respect to the goodness. This means that the output of each layer loses a degree of freedom, i.e. some information is lost. The information that remains is how much each output feature contributed to the total goodness compared to its peers. This information is passed to the next layer. Because each layer loses some information, it makes intuitive sense to use the concatenation of the outputs of multiple layers as the output of the network, rather than just the last layer.

The normalization and per-layer nature of the objective is essential to the usefulness of FFA. If a network was trained with backpropagation using the same objective as FFA for its final layer, it might not output useful features. Instead, the hidden layers could learn tot tell positive from negative data, and the final output layer could just maximize all of its outputs for positive data, and minimize them for negative data, without exposing any of the underlying features on which its prediction is based.

2.1 Mathematical description of the Forward-Forward algorithm

The feed-forward operation of a Forward-Forward network is very similar to that of artificial neural networks trained with backpropagation. Given an output of the previous layer $\mathbf{v}_{i-1} \in \mathbb{R}^{n_{i-1}}$, with $\mathbf{v}_0$ being the input to the network, the unnormalized output activities $\mathbf{a}_i \in \mathbb{R}^{n_i}$ of layer i with weights $W_i \in \mathbb{R}^{n_i \times n_i}$, using a ReLU-activation [24] (following the FFA paper), are:

$$\mathbf{a}_i = \max\left(0, W_i \mathbf{v}_{i-1} + \mathbf{b}_i\right) \tag{1}$$

The goodness of layer i is the sum of squares of its unnormalized output, $g_i = \|\mathbf{a}_i\|^2$. The final output $\mathbf{v}_i$ is normalized w.r.t. the mean of its squared elements:

$$\mathbf{v}_i = \sqrt{n_i}\frac{\mathbf{a}_i}{\|\mathbf{a}_i\|} \Rightarrow \frac{\|\mathbf{v}_i\|^2}{n_i} = 1 \tag{2}$$

where $n_i \in \mathbb{N}$ is the number of neural units in layer i. Note that g_i cannot be derived from the normalized output $\mathbf{v}_i$, as intended. Given g_i, a probability of the input to layer i being positive data, p_i, is computed by passing the difference between the goodness and a threshold value through a logistic function:

$$p_i = \sigma\left(g_i - n_i\right), \qquad \sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

The Forward-Forward algorithm performs gradient descent on the weights of each layer, but instead of a global loss function, the gradient of a layer-local objective function, C_i, is used. The exact definition of C_i is discussed in appendix A. Different gradients are defined for the positive and negative passes, $\nabla_{\mathbf{a}_i} C_i^+$ and $\nabla_{\mathbf{a}_i} C_i^-$, respectively:

$$\begin{aligned} \nabla_{\mathbf{a}_i} C_i^+ &= \left(1 - p_i\right)\mathbf{a}_i + \lambda\left(t_i - \mathbf{m}_i\right) \\ \nabla_{\mathbf{a}_i} C_i^- &= -p_i \mathbf{a}_i \end{aligned} \tag{4}$$

where $\lambda\left(t_i - \mathbf{m}_i\right)$ is a peer normalization term applied in the positive pass. Peer normalization is the process of adjusting the weights such that the mean activation of each neural unit in a layer is pulled towards that of its peers. The goal is to prevent units from reaching a state where their value is always zero and no more updates to the weights occur. The hyperparameter λ is the peer normalization weight, $\mathbf{m}_i$ is an exponential moving average of the unnormalized activity of layer i, and t_i is the target mean activity. The per-batch update rules of $\mathbf{m}_i$ and t_i are as follows:

$$\begin{aligned} \mathbf{m}_i' &= \beta_1 \mathbf{m}_i + (1 - \beta_1)\overline{\mathbf{a}_i} \\ t_i &= \max\left(\theta_i,\ \frac{\mathbf{1}^\top \mathbf{m}_i}{n_i}\right) \end{aligned} \tag{5}$$

with β_1 a hyperparameter controlling how quickly the exponential moving average responds to new values. The overline notation on $\overline{\mathbf{a}_i}$ is used to represent the average value of $\mathbf{a}_i$ within the current batch, assuming batch updates are used. A threshold, θ_i, determines a minimum value for the target mean activity. Such a threshold is not used in the original FFA paper [1], which is equivalent to choosing $\theta_i = 0$. Section 3.2 explores the benefit of choosing $\theta_i > 0$.

The gradients $\nabla_{\mathbf{a}_i} C_i^+$ and $\nabla_{\mathbf{a}_i} C_i^-$ are propagated to the weights by the chain rule, as usual:

$$\nabla_{W_i} C_i^\pm = \left(\nabla_{\mathbf{a}_i} C_i^\pm\right)\mathbf{v}_{i-1}^\top \tag{6}$$

The gradient of the ReLU activation function is chosen to always be equal to one. In practice this only has the effect of allowing the peer normalization term of the gradient (4) to have an effect even when the activation of a unit (i.e. its respective element in $\mathbf{a}_i$) is zero. The weights are updated according to the normal procedure for gradient descent with learning rate α and momentum parameter β_2. First, the momentum variables are updated:

$$\begin{aligned}\Delta'_{W_i} &= \beta_2 \Delta_{W_i} + (1-\beta_2)\,\overline{\left(\nabla_{W_i} C_i^+ + \nabla_{W_i} C_i^-\right)} \\ \Delta'_{\mathbf{b}_i} &= \beta_2 \Delta_{\mathbf{b}_i} + (1-\beta_2)\,\overline{\left(\nabla_{\mathbf{a}_i} C_i^+ + \nabla_{\mathbf{a}_i} C_i^-\right)}\end{aligned} \tag{7}$$

and then the weights:

$$\begin{aligned}W'_i &= (1-\alpha\gamma)\left(W_i + \alpha \Delta_{W_i}\right) \\ \mathbf{b}'_i &= \mathbf{b}_i + \alpha \Delta_{\mathbf{b}_i}\end{aligned} \tag{8}$$

These update rules are applied per batch, and the overline notation is once again used to average over the values of the gradients within a batch. A hyperparameter γ is used to enforce weight decay [25], to prevent the development of extremely large weights.

2.2 Implementation in Keras/TensorFlow

The original implementation of FFA [26] is written in MATLAB, without a machine learning framework and without GPU-support. For this reason, as well as personal preference, a custom implementation of FFA has been written in Python, using the Keras [27] framework with TensorFlow [28] as its backend. A detailed explanation of the implementation is given in appendix B.

3 Experiments on MNIST

3.1 Benchmark

To get an idea of how well the FFA implementation works on a well-studied machine learning task, the experiment titled *A simple supervised example of FF* in the FFA paper [1] will be repeated. This experiment involves classifying images from the MNIST dataset of handwritten digits [29] by training a Forward-Forward network on inputs that consist of an image and a one-hot encoded class label. A softmax layer is trained to predict the class labels from the output features of the FF network. In the positive pass, the correct labels are used in the input, while in the negative pass an incorrect label is used. The softmax layer is trained using a neutral label (where each category has the value 0.1), to ensure the classifier has no label information. The output probabilities of the classifier layer are used to draw a false label for the negative pass. This way, the most confusing false labels are chosen more often, thus focusing the training on classes for which the network has not yet learned the right features to correctly classify images.

3.1.1 Methods

A single run of the MNIST classification experiment is conducted to show that the learning procedure in Keras/TensorFlow yields similar results to the MATLAB implementation. Additionally, hyperparameter searches are conducted to study the effects of various hyperparameters. An architecture of three Forward-Forward layers of 1,000 units each is used, of which the normalized outputs of the last two layers are used as input to a single softmax layer of ten units, one for each class.

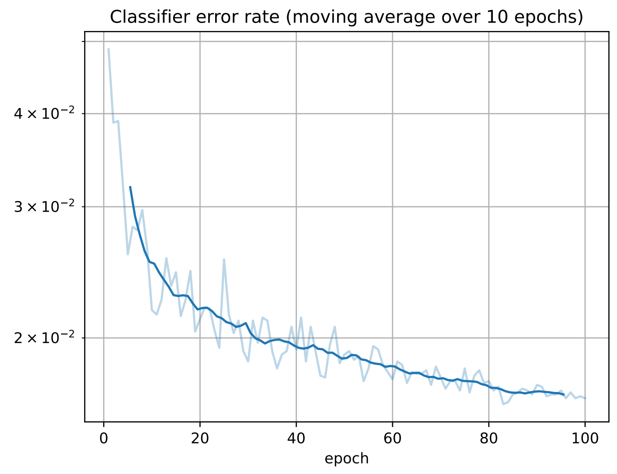
Two main metrics are used: classification error and pairwise errors. The classification error is simply the average number of times the network misclassifies a sample. Pairwise errors are computed per layer, and are defined as the average number of times the layer mislabels a pair of positive and negative inputs. A pair of inputs is considered mislabelled by layer i when p_i (3) is higher for the negative input than it is for the corresponding positive input. Note that this metric is only applicable to situations where negative samples are generated by applying some modification to an existing positive sample.

The default hyperparameters are a learning rate of 0.01 (0.1 for the classifier layer), momentum of 0.9, peer normalization weight and delay parameters of 0.03 and 0.9 respectively, and a weight decay of 0.01.

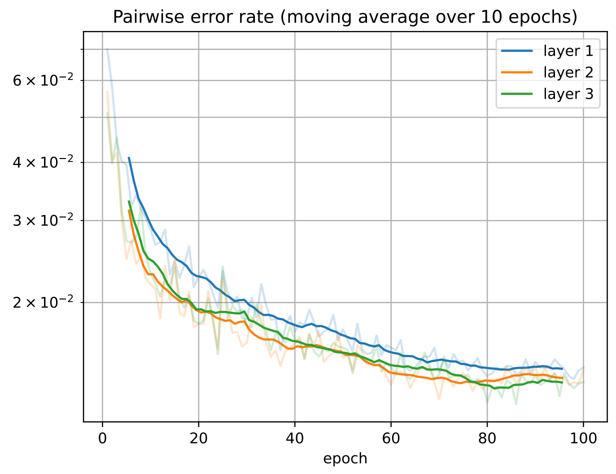
3.1.2 Results

Single run The experiment was run for 100 epochs, and classification- and pairwise errors were recorded. Linear learning rate decay towards zero was used, starting from epoch 51. Training starts at epoch 1.

Figure 1 shows all validation error measurements declining as training progresses. The decline of the pairwise errors in all layers shows that the FF network learns to effectively discriminate between positive and negative data. Furthermore, the decline of the classification error over time shows that the learned features are indeed useful for classification. The classification error at the end of



(a) Classification error rate.

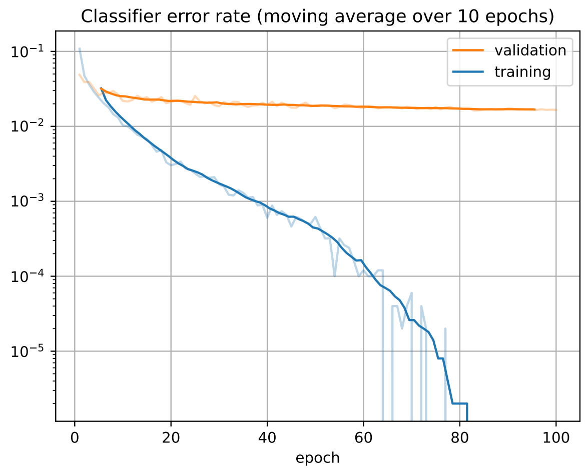


(b) Pairwise error rate in each layer.

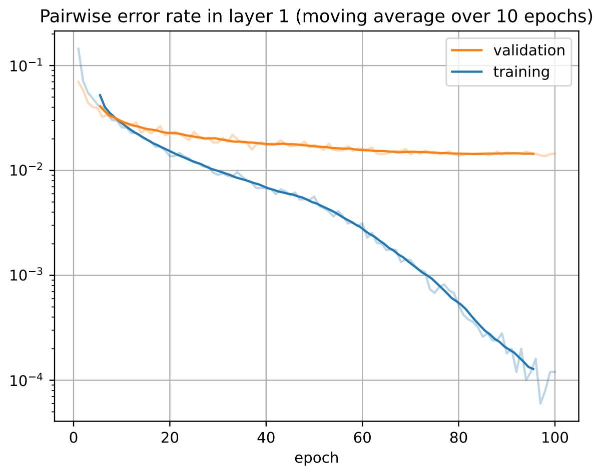
Figure 1: Validation errors from a single run of the MNIST experiment. A moving average over 10 epochs is plotted. The original data are plotted in the respective lighter colors. Linear learning rate decay towards zero was used, starting from epoch 51.

training is about 1.7%, which is close to the 1.8% validation error achieved by running the official MATLAB code [26]. Note that these numbers both represent a single run, and do not necessarily represent the mean classification error over many runs.

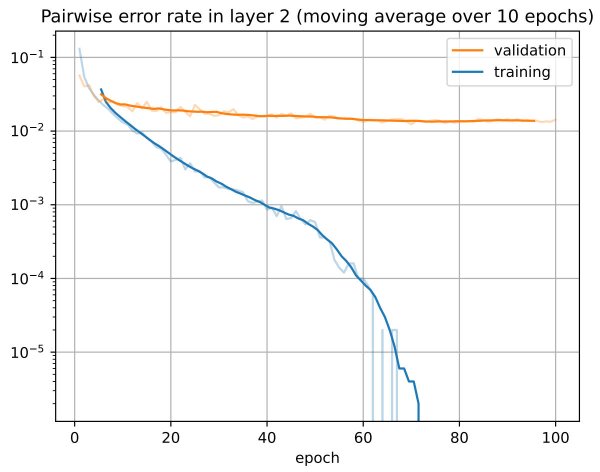
Figure 2 shows a comparison between training and validation metrics. While the validation errors decline steadily, the training errors decline much steeper in comparison, especially once learning rate decay kicks in. The training errors eventually reach zero out of 50,000 training samples in all metrics except for the pairwise errors in layer 1, while the validation error rates remain in the order of 10^{-2}. In other words, the network eventually memorises all (difficult) training samples, i.e. it overfits. This suggests that the network might perform much better on the validation set if a larger dataset were used for training, or that a network with less parameters might perform equally as well.



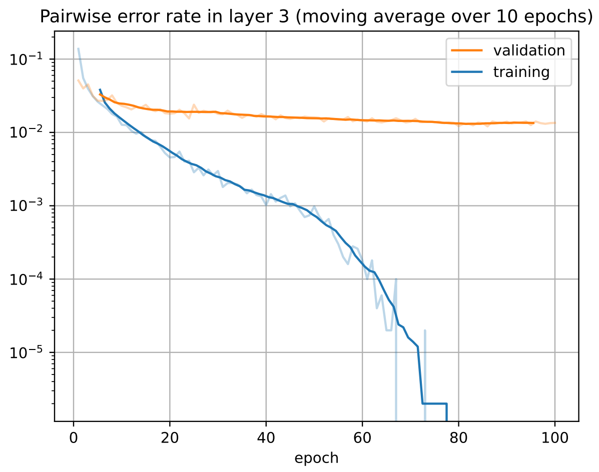
(a) Classification error.



(b) Pairwise errors in layer 1.



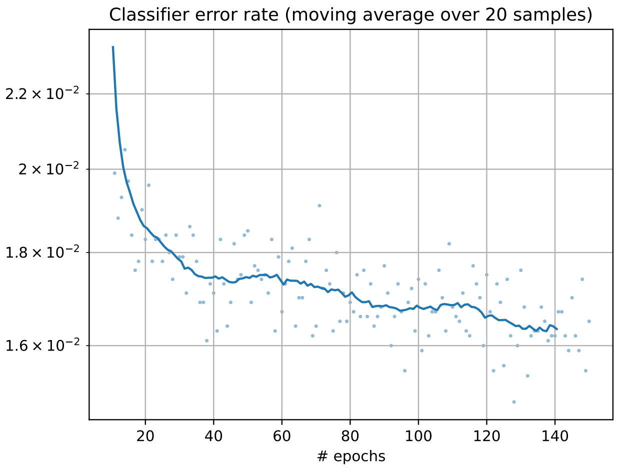
(c) Pairwise errors in layer 2.



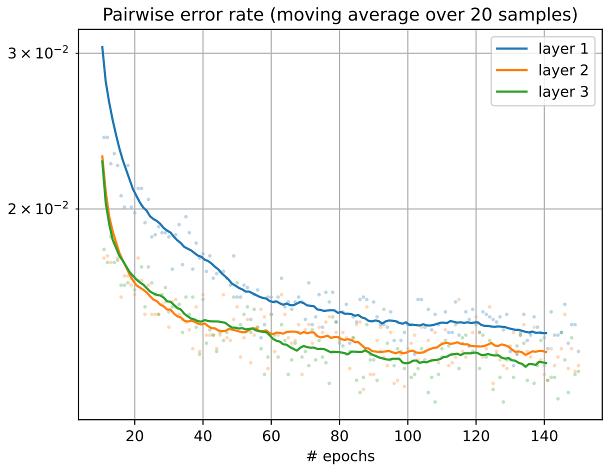
(d) Pairwise errors in layer 3.

Figure 2: (a) Classifier training and validation error. (b, c, d) Pairwise training and validation errors for layers 1, 2, and 3 respectively. A moving average over 10 epochs is plotted in all figures. The original data are plotted in the respective lighter colors. Linear learning rate decay towards zero was used, starting from epoch 51.

Varying the number of training epochs The model was repeatedly trained for each number of epochs in the range $\{1, \ldots, 150\}$. A batch size of 200 was used. Linear learning rate decay towards zero was used, starting when half the total number of epochs has elapsed.



(a) Classification error.



(b) Pairwise errors.

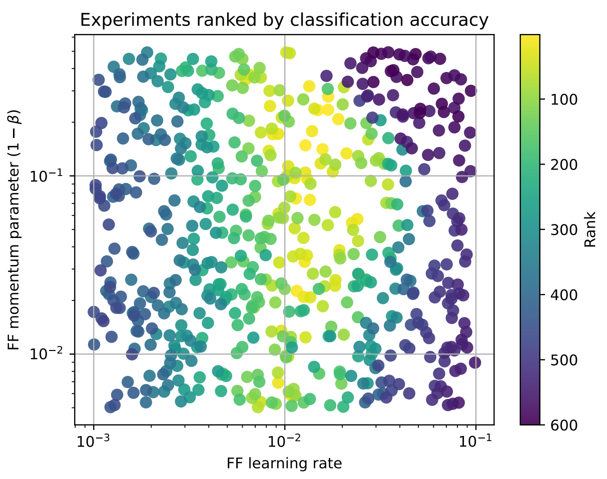
Figure 3: Validation errors as a function of the number of epochs. A moving average over 20 samples is plotted, the original data are scatter-plotted in the associated color. The first ten samples are omitted in the scatter plot for readability.

As shown in figure 3, the classification accuracy and pairwise accuracy improve with longer training. However, there are diminishing returns when training longer than about 40 to 60 epochs: the difference in accuracy on a log-scale between training for 60 or 140 epochs is about the same or less than the difference between training for 20 or 60 epochs. For this reason, the hyperparameter searches will only train each model for 60 epochs.

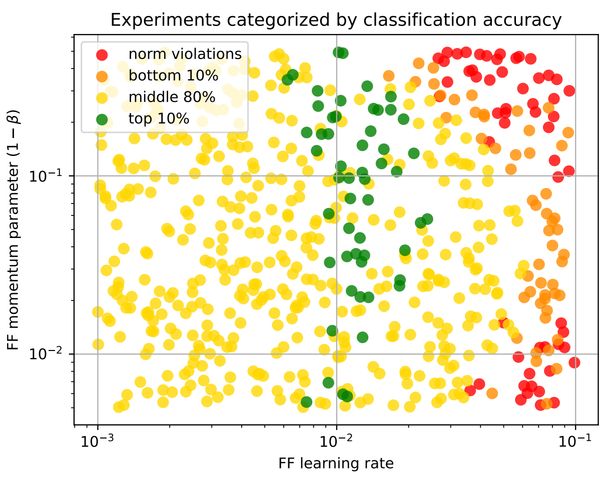
This experiment also provides a margin of error for the performance near 100 epochs: between 90 and 110 epochs the classification error ranges between 1.55% and 1.82%, which falls between the result of 1.8% obtained with the original MATLAB code and the 1.4% result mentioned in the original FFA paper for a network with 2,000 hidden units - twice as many - per Forward-Forward layer.

Hyperparameters A random hyperparameter search was conducted over different sets of hyperparameters. A random search was chosen as opposed to a grid search, because a random search finds good hyperparameters more efficiently [30] while searching uniformly through the space of hyperparameters. Each model in the hyperparameter search was trained for 60 epochs, using a batch size of 200. Linear learning rate decay towards zero was used, starting from epoch 31.

Figure 4 clearly shows an optimal learning rate, whereas the momentum parameter seems to make comparatively less impact on performance. The optimal parameters seem to be a learning rate in the range [0.01, 0.02], with a momentum parameter β between 0.8 and 0.9. Strangely, for learning rates $\alpha > 0.02$ there are many cases where the supposedly normalized outputs of the FF network



(a) Ranked from lowest classification error (yellow) to highest error (dark purple).



(b) Top 10% and bottom 10% categories, as well as a norm-violation category.

Figure 4: Results from a random hyperparameter search over the learning rate and momentum parameter of the Forward-Forward network. The Y-axes represent $1 - \beta$, with β the momentum parameter, to facilitate log-scaling. Metrics are taken from the validation step of the final epoch. 'Norm-violations' indicate cases where the mean of squares of the FF network output was not equal to one.

do not have a mean of squared elements equal to one. This happens due to a division by zero that occurs when the outputs are all zero, which is supposed to be prevented through peer normalization (described in 2.1). The explanation lies in the fact that a threshold of $\theta_i = 0$ is used for the target mean activity (5). Because of this, the target mean activity is always equal to the mean activity of all units, which might eventually become zero. If this happens, there is no way for the network to recover, because no units will ever receive a non-zero gradient. The solution is to choose some $\theta_i > 0$, which will be explored in section 3.2.

Further results of hyperparameter searches are omitted here for brevity, and can be found in appendix C.

3.1.3 Conclusion

The custom Keras implementation has been shown to perform as expected on MNIST: it learns to discriminate between positive and negative data; the learned features are useful for classification; and the final classification performance is within margin of error of the performance of the original MATLAB code [26] supplied with the paper describing FFA [1].

Furthermore, results from a hyperparameter search show that the algorithm is comparatively much more sensitive to the learning rate than to the momentum parameter. Results also show that with high learning rates the FF network suffers from the *dying ReLU problem* [31]: neural units which always have an activation value of zero, stopping their weights from receiving updates. The proposed solution is to use a positive threshold $\theta_i > 0$ for peer normalization.

3.2 Peer normalization threshold

Results from the hyperparameter search in section 3.1.2 indicate that for high learning rates, a Forward-Forward network can sometimes reach a state where all units in a layer are permanently deactivated. The proposed solution is to choose a positive minimum value $\theta_i > 0$ for the target mean activity (5), which will be investigated in this section.

First, it is useful to think about what a good minimum value should be from a mathematical point of view. Because peer normalization is only applied in the positive pass, we should choose θ_i as high as possible, to encourage high activities. At the same time, we should also constrain the activities, $\mathbf{a}_i$, and thus the learned features, as little as necessary. Therefore, we should choose θ_i such that the estimated probability of the data being positive, p_i, is able to exceed 50% even when the mean activity is as low as allowed by θ_i. From the definition (3) of p_i we can see that, given a layer of n_i units:

$$p_i \geq 0.5 \Leftrightarrow \|\mathbf{a}_i\|^2 \geq n_i \tag{9}$$

However, we cannot control the sum of squared activities, $\|\mathbf{a}_i\|^2$, directly. But, in positive passes, assuming that the peer normalization weight is sufficiently high, we can steer the sum of activities, $\mathbf{1}^\top \mathbf{a}_i$, towards:

$$\mathbf{1}^\top \mathbf{a}_i \geq n_i \theta_i \tag{10}$$

We can view this as a soft constraint on $\mathbf{a}_i$. The threshold only takes effect when the mean activity is less than θ_i. Therefore, in cases where it acts as a constraint, such as when all units are permanently off, we can assume that we steer the activities towards the lowest mean activity that satisfies the constraint. This is a worst-case scenario. It would be better if the activities were higher, but in those cases a threshold would not be necessary, nor would it have any effect. In short, for practical purposes we can assume that we steer the activities towards $\mathbf{1}^\top \mathbf{a}_i = n_i\theta_i$. We can now compute an upper bound for $\|\mathbf{a}_i\|^2$, using the fact that the sum of squared activities is at most the square of the sum of activities, which it is when all activities are zero except for one. This gives us the following upper bound:

$$\mathbf{1}^\top \mathbf{a}_i = n_i\theta_i \Rightarrow \|\mathbf{a}_i\|^2 \leq (n_i\theta_i)^2 \tag{11}$$

So, to be able to satisfy eq. (9) in the worst case, when $\mathbf{1}^\top \mathbf{a}_i = n_i\theta_i$, we have to choose θ_i to be at least:

$$\theta_i \geq \frac{1}{\sqrt{n_i}} \tag{12}$$

It is always safe to choose this exact value for θ_i, because an FF layer with a mean activity below this value will never be able to satisfy the requirement of eq. (9). Choosing a higher value will further constrain the activities, $\mathbf{a}_i$, which may not be desirable because it consequently also restricts which features the layer is able to learn. For example, a layer might learn n_i features, each of which is active $\frac{1}{n_i}$ of the time (in the positive pass, that is), with a sum of squared activities near $\|\mathbf{a}_i\|^2 \approx n_i$. If a threshold $\theta_i > \frac{1}{\sqrt{n_i}}$ is used, the layer can't exist in this state. It would have to either scale its activities linearly with θ_i (which will be punished in the negative pass if the layer isn't perfectly accurate), or learn different features that are active more often.

Nonetheless, it might actually be desirable for layers to learn multiple redundant features, or for features to be more active than the minimum to achieve $p_i \geq 0.5$. For this reason, different values for θ_i will also be investigated in the following subsections.

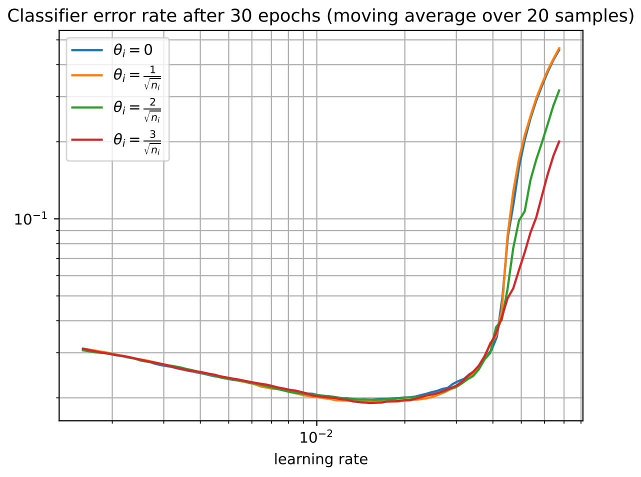
3.2.1 Methods

To test the effectiveness of the peer normalization threshold, the MNIST classifier from section 3 is trained repeatedly for a range of learning rates, and with different momentum and threshold parameters. The network is trained for 60 epochs each time, with linear learning rate decay towards zero starting from epoch 31.

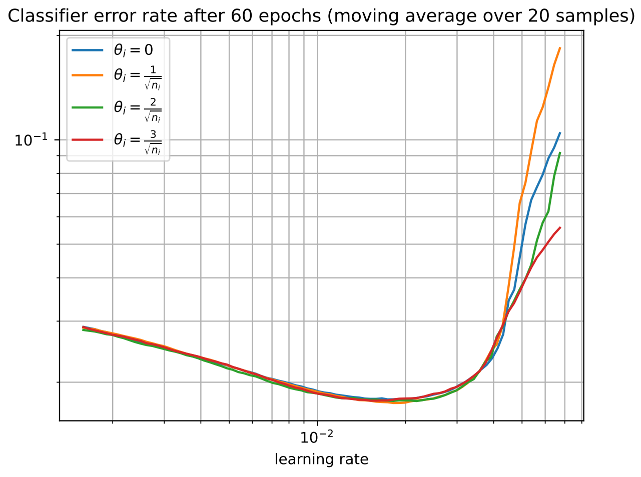
Because peer normalization is supposed to prevent layers from becoming permanently deactivated, it is to be expected that the impact of a threshold $\theta_i > 0$ is strongest in those regions of the hyperparameter space where norm violations were observed, i.e. the areas containing red points in figure 4b. Two values for the momentum parameter are tested: $\beta = 0.9$ and $\beta = 0.5$. Near the horizontal line for $\beta = 0.9 \Rightarrow 1 - \beta = 0.1$ in figure 4b, there are some norm violations at high learning rates, while there are comparatively many more in a wider region near the horizontal line for $\beta = 1 - \beta = 0.5$. The hypothesis is that for $\beta = 0.9$, the peer normalization threshold has some small positive impact on the classification accuracy for very high learning rates, and a bigger impact for $\beta = 0.5$ at a wider range of learning rates.

3.2.2 Results

Figures 5 and 6 show the results for $\beta = 0.9$ and $\beta = 0.5$ respectively. Four values of θ_i were tested: 0, $\frac{1}{\sqrt{n_i}}$, $\frac{2}{\sqrt{n_i}}$ and $\frac{3}{\sqrt{n_i}}$.



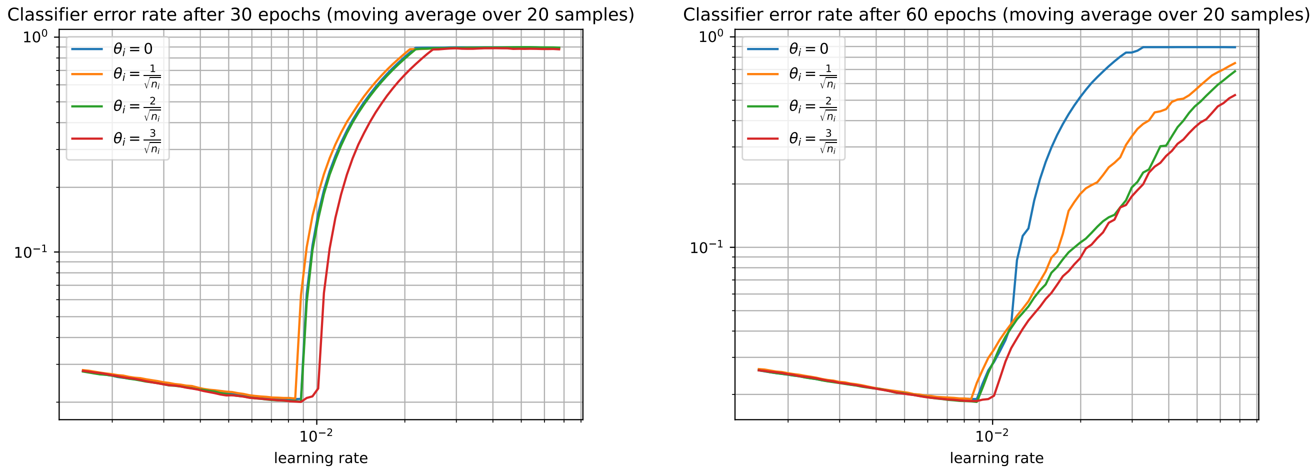
(a) Classification error after 30 epochs.



(b) Classification error after 60 epochs.

Figure 5: Classification error on the validation set as a function of learning rate, after (a) 30 and (b) 60 epochs. Results are smoothed using a moving average over 20 samples, over a total of 103 samples spaced evenly on a log-scale. A momentum parameter of $\beta = 0.9$ was used.

The results align almost perfectly with the hypothesis. At $\beta = 0.9$, the choice of θ_i (within the range of selected values) has no impact on the final classification error up to a learning rate of 0.04, after which the highest threshold, $\theta_i = \frac{3}{\sqrt{n_i}}$, performs best. At high learning rates, a threshold of $\theta_i = \frac{1}{\sqrt{n_i}}$ performs worse than using no threshold, it is unclear why this is the case. At $\beta = 0.5$, the benefits of using a peer normalization threshold are already noticeable starting from a learning rate of 0.01. The highest choice again performs best.



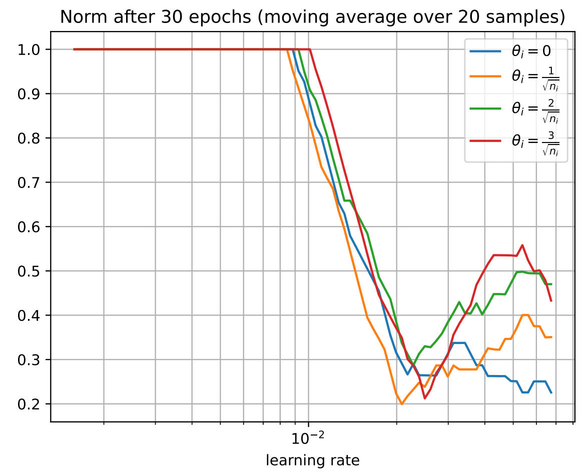
(a) Classification error after 30 epochs.

(b) Classification error after 60 epochs.

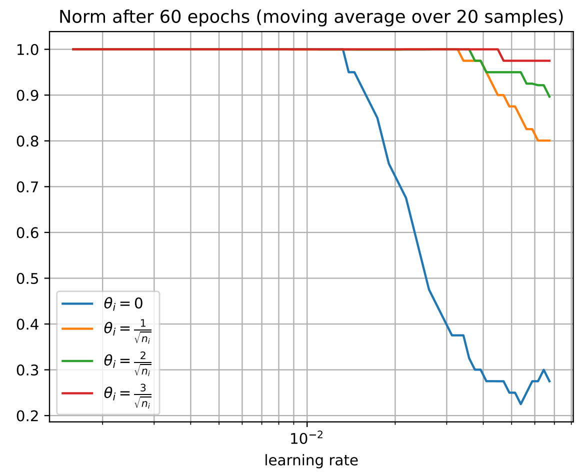
Figure 6: Classification error on the validation set as a function of learning rate, after (a) 30 and (b) 60 epochs. Results are smoothed using a moving average over 20 samples, over a total of 103 samples spaced evenly on a log-scale. A momentum parameter of $\beta = 0.5$ was used.

It is interesting to note that, for $\beta = 0.5$ and learning rates above 0.03, up to epoch 30 the network has an error rate of 90%, which on MNIST is equivalent to random guessing. In other words, the network doesn't learn, regardless of the threshold. However, due to learning rate decay, the network is able to improve again in the epochs 31 − 60 if the threshold was $\theta_i > 0$. This is a sign that the peer normalization threshold indeed stops the layers from becoming permanently deactivated.

Figure 7 shows this more clearly by plotting the mean squared value (from now on referred to as the norm) of the normalized output of the network as a function of the learning rate, when using $\beta = 0.5$. Since the output is normalized, the norm should always be equal to one, except in cases where the activities are all zero. In that case a division by zero using the TensorFlow function `tf.math.divide_no_nan` returns zero, causing the norm to be zero. At learning rates below 0.008 the norm is indeed always equal to one. In fact, the onset of norm violations in figure 7a coincides with the sharp increases in classification error in figure 6. At higher learning rates there are always norm violations after 30 epochs, but when the threshold is $\theta_i > 0$, the network is able to recover in epochs 31 − 60 due to learning rate decay. Again, $\theta_i = \frac{3}{\sqrt{n_i}}$ shows the best improvement.



(a) Norm after 30 epochs.



(b) Norm error after 60 epochs.

Figure 7: Mean squared value (norm) of normalized outputs on the training set as a function of learning rate, after (a) 30 and (b) 60 epochs. Results are smoothed using a moving average over 20 samples, over a total of 103 samples spaced evenly on a log-scale. A momentum parameter of $\beta = 0.5$ was used.

3.2.3 Conclusion

The usage of a peer normalization threshold $\theta_i > 0$ does not fully prevent performance degradation at high learning rates due to norm violations, but it does prevent layers from becoming permanently deactivated and unable to learn. There are no observed benefits or downsides to using such a threshold at lower learning rates. It is clear that for MNIST, in the cases where it matters, the higher the threshold, the better, within the range of tested thresholds. However, because a high threshold value can potentially constrain a network on a different learning task, and to avoid having to pick a new threshold for each task, a threshold of only $\frac{2}{\sqrt{n_i}}$ is chosen to be used in all experiments from this point on.

4 Physical Models

4.1 Ising model

One of the simplest models that exhibits a phase transition is the Ising model of ferromagnetism [32, 33]. The model consists of particles placed on a lattice, with bonds between neighbors. At each lattice site i there is a particle which has a spin σ_i, which can be up ($\sigma_i = 1$) or down ($\sigma_i = -1$). Each directly neighboring site j also has a particle with a spin. If the spins σ_i and σ_j of the neighboring particles are equal (aligned), their bond is in a low-energy state. If the spins are not aligned, the bond is in a high-energy state. The energy of the complete model is described by its Hamiltonian, which is the sum of the energy contributions of every bond:

$$H_{\text{Ising}} = -\sum_{\langle i,j \rangle} \sigma_i \sigma_j \tag{13}$$

where $\langle i, j \rangle$ is meant to represent all unique pairs of neighboring sites i and j, with $i, j \in \{1, \ldots, N\}$ for a total number of sites N. This thesis covers the square-lattice version of the Ising model, which means the particles are placed on an $L \times L$ square lattice, so $N = L^2$ with $L \in \mathbb{N}$. Periodic boundary conditions are applied, i.e. sites on one edge of the lattice are neighbors with sites on the opposing edge, such that every site has four neighbors.

The model behaves according to Boltzmann statistics, which is to say that the relative likelihood of finding the system in state X vs. state Y depends on their respective Hamiltonians:

$$\frac{p_X}{p_Y} = \exp\left(\frac{H_Y - H_X}{k_B T}\right) \tag{14}$$

with T the temperature and k_B the Boltzmann constant.

The lower the Hamiltonian energy of a state, the higher its individual probability. However, the total number of states that have a certain energy increases rapidly as the energy increases from the minimal value to higher values. Consider for example a system of three particles, with spins σ_1, σ_2 and σ_3. There are only two states where all spins are aligned: $\sigma_1 = \sigma_2 = \sigma_3 = 1$ and $\sigma_1 = \sigma_2 = \sigma_3 = -1$. However, there are six states where the spins are not all aligned: there are three different ways to choose a particle to have a different spin from the rest, and two ways to choose its sign. As the number of particles increases, this difference in the number of low-energy and high-energy states only grows.

The temperature controls how sensitive the statistical distribution of states is to the energy difference between states. As T approaches zero, only the states with the lowest energy are probable. On the other hand, as T approaches infinity, every state becomes equally likely and high-energy states will dominate since there are simply more of them.

Returning to the Ising model, Boltzmann statistics imply that at low temperatures we are likely to find states where many neighboring spins are aligned, while at high temperatures more neighbor pairs will be unaligned. This leads to a phenomenon called *spontaneous magnetization*: below a certain critical temperature, T_c, the system switches from a disordered phase to an ordered phase. In the disordered phase, spin-up and spin-down particles are equally common. In the ordered phase, this symmetry is broken and the system becomes dominated by one of the two spin values. When this happens, i.e. when most particles have the same spin, the model is said to be ferromagnetic. The magnetization M is given by:

$$M = \frac{1}{N} \sum_i \sigma_i \tag{15}$$

The exact transition point of the Ising model is given by:

$$T_c = \frac{2}{k_B \ln\left(1+\sqrt{2}\right)} \tag{16}$$

In this thesis $k_B = 1$ is used, so $T_c \approx 2.269$.

4.2 Potts model

The q-state Potts model [34] is a generalization of the Ising model. It allows lattice sites to have $q \geq 2$ different spin directions, with $q \in \mathbb{N}$. Similar to the Ising model, neighbors with equal (aligned) spins have a low-energy bond, whereas neighbors with unaligned spins have a high-energy bond. This is reflected in the Hamiltonian of the Potts model:

$$H_{\text{Potts}} = -J \sum_{\langle i,j \rangle} \delta\left(s_i, s_j\right) \tag{17}$$

Here, J is a coupling constant, δ is the Kronecker-delta and $s_i \in \{1, \ldots, q\}$ is the spin value of site i. The magnetization of the Potts model has a magnitude and direction, given by $M_{\mathbb{C}} \in \mathbb{C}$:

$$M_{\mathbb{C}} = \frac{1}{N} \sum_j \exp\left(\frac{-2\pi i s_j}{q}\right), \qquad M^2 = |M_{\mathbb{C}}|^2 \tag{18}$$

where in the context of $M_{\mathbb{C}}$, i is the imaginary unit. The exact transition point is given by:

$$T_c = \frac{J}{k_B \ln\left(1+\sqrt{q}\right)} \tag{19}$$

To show that this is indeed a generalization of the Ising model, we rewrite the Ising Hamiltonian (13):

$$H_{\text{Ising}} = -\sum_{\langle i,j \rangle} \sigma_i \sigma_j = N - 2 \sum_{\langle i,j \rangle} \delta\left(\sigma_i, \sigma_j\right) \tag{20}$$

Because the Boltzmann distribution (14) only deals with relative Hamiltonians, we can ignore the constant N. We can also switch to a different representation of the spin value at site i, using $s_i \in \{1, 2\}$ instead of $\sigma_i \in \{-1, 1\}$, because all that matters is whether two spins are equal. This allows us to write:

$$H_{\text{Ising}} = -J \sum_{\langle i,j \rangle} \delta\left(s_i, s_j\right), \quad J = 2 \tag{21}$$

which shows that the Ising model is equivalent to the 2-state Potts model with coupling constant $J = 2$. By substituting $q = 2$, $k_B = 1$ and $J = 2$ into the equation (19) for the critical temperature of the Potts model, we get:

$$T_c = \frac{2}{\ln\left(1+\sqrt{2}\right)} \approx 2.269 \tag{22}$$

which is indeed the transition point of the Ising model. While the coupling constant is $J = 2$ for the Ising model, for the Potts model $J = 1$ is generally used, so that will be the convention in this thesis, along with $k_B = 1$.

4.3 Transverse-field Ising model

The transverse-field Ising model (TFIM) [35, 36] is similar to the classical Ising model, with the addition of an external magnetic field orthogonal to the neighbor-neighbor alignment axis. Specifically, the neighbor-neighbor interactions from the Ising model apply to the z-components of the spins, while the external magnetic field is applied to the x-components. Because quantum mechanics forbids the simultaneous measurement of the spin of the same particle along both the x- and z-axes, this model can't be described by a classical Hamiltonian. This means that the TFIM Hamiltonian isn't a function of the system state, but rather a quantum operator that can be used to extract the expectation value of the total energy from a wave function. A wave function is a mapping $s \mapsto \psi(s)$ of a spin-state s to a probability amplitude $\psi(s) \in \mathbb{C}$, such that the probability density of finding state s is $|\psi(s)|^2$. A wave function can represent a single state or a superposition of multiple states. Wave functions have to be normalized such that:

$$\sum_s |\psi(s)|^2 = 1 \tag{23}$$

Given a wave function ψ, the expectation value of the Hamiltonian energy is defined by a linear product (in Dirac notation):

$$\langle \hat{H} \rangle = \langle \psi | \hat{H} | \psi \rangle \tag{24}$$

For a TFIM with N spin sites, ψ can be any superposition of 2^N basis states - one for each unique configuration of z-spins. This means the wave function can be represented by a vector $\psi \in \mathbb{C}^{2^N}$, and the Hamiltonian operator by a matrix $\hat{H} \in \mathbb{C}^{2^N \times 2^N}$. Then, the product $\langle \psi | \hat{H} | \psi \rangle$ can be written as a matrix multiplication:

$$\langle \psi | \hat{H} | \psi \rangle = \psi^* \hat{H} \psi \tag{25}$$

where ψ^* is the conjugate transpose of ψ. The Hamiltonian operator for the transverse-field Ising model looks very similar to the Hamiltonian for the classical Ising model, with an extra term for the external field:

$$\hat{H}_{\text{TFIM}} = -\sum_{\langle i,j \rangle} \hat{\sigma}_i^z \hat{\sigma}_j^z - g \sum_i \hat{\sigma}_i^x \tag{26}$$

Here, operators $\hat{\sigma}_i^x, \hat{\sigma}_i^z \in \mathbb{C}^{2^N \times 2^N}$ are Pauli spin matrices, which on their own can be used to extract the expectation values of the individual spins of each site $i \in \{1, \ldots, N\}$, along the x- and z-axes respectively. The strength of the external field is $g \in \mathbb{R}$. This thesis covers the 1D version of the model, so the spin sites are placed on a chain rather than a 2D lattice. Periodic boundary conditions are applied, meaning that each site has exactly two neighbors.

The $-\sum_{\langle i,j\rangle} \hat{\sigma}_i^z \hat{\sigma}_j^z$ term in the TFIM Hamiltonian gives neighboring spins that are aligned on the z-axis a lower energy compared to anti-aligned spins, while the $-g\sum_i \hat{\sigma}_i^x$ term gives spins that are aligned with the external magnetic field g on the x-axis a lower energy compared to spins that are opposite to the external field. Like the Ising and Potts models, TFIM states with a low energy are statistically favored over states with a high energy.

In this thesis, the TFIM at temperature $T = 0$ (i.e. the ground state) is studied, and the spins are measured along the z-axis. The ground state of the TFIM has a phase transition at $|g| = 1$. At $|g| < 1$, the system is in the ordered phase where most spins are aligned on the z-axis, i.e. the model is ferromagnetic. At $|g| > 1$, the system is in the disordered phase, where most spins are aligned with the external field, but unaligned on the z-axis. The magnetization of the TFIM is given by the following operator:

$$\hat{M} = \frac{1}{N}\sum_i \hat{\sigma}_i^z \tag{27}$$

4.4 Many-body localization

The models described in the previous subsections all have a ferromagnetic phase, and therefore have an order parameter that can easily be found using PCA [11]. This isn't the case for the many-body localization (MBL) model [17, 37–39]. MBL is a quantum-mechanical model where a fixed number of fermions are placed on a chain. Fermions are particles that (as opposed to Bosons) have the property that only one of them can occupy the same site at a time, due to the Pauli exclusion principle. Some examples of fermions are electrons, protons and neutrons. In this thesis a model of N fermionic particles on a chain of $L = 2N$ sites is used. The particles have nearest neighbor interactions that cause them to repel each other. Furthermore, the model has random per-site potentials, making some sites more energetically favorable than others. If this random field is strong enough, it causes the particles to *localize* to a specific configuration.

The MBL Hamiltonian is defined in second quantized form; meaning in terms of creation ($\hat{c}_i^*$), annihilation ($\hat{c}_i$) and number ($\hat{n}_i = \hat{c}_i^*\hat{c}_i$) operators. Applying the creation or annihilation operator to the wave function ψ_s of a state s yields the wave function $\psi_{s'}$ of a new state s' with a particle added or removed, respectively, at site $i \in \{1,\ldots,L\}$. Each site can be occupied by at most one particle at a time, so applying $\hat{c}_i^*$ or $\hat{c}_i$ will yield zero for any (superposition of) state(s) for which their respective actions would be invalid. The number operator $\hat{n}_i$ extracts the expectation value for the number of particles at site i.

The Hamiltonian of the MBL model used in this thesis is:

$$\hat{H}_{\text{MBL}} = -t\sum_{\langle i,j\rangle}\left(\hat{c}_i^*\hat{c}_j + \hat{c}_j^*\hat{c}_i\right) + J\sum_{\langle i,j\rangle}\hat{n}_i\hat{n}_j + \sum_i h_i\hat{n}_i \tag{28}$$

with $\hat{H}_{\text{MBL}} \in \mathbb{C}^{S\times S}$, where $S = \frac{L!}{N!(L-N)!} = \frac{(2N)!}{(N!)^2}$ is the size of the allowed configuration space. The per-site potentials $h_i \in \mathbb{R}$ are randomly chosen from a uniform distribution $[-W, W]$, with W the tuning parameter. The term $J\sum_{\langle i,j\rangle}\hat{n}_i\hat{n}_j$, with $J = 1$, causes neighboring particles to repel each other. Meanwhile, the term $-t\sum_{\langle i,j\rangle}\left(\hat{c}_i^*\hat{c}_j + \hat{c}_j^*\hat{c}_i\right)$, with $t = 2$, allows the movement of particles (also known as *hopping* or *tunneling*) between sites. The $\langle i,j\rangle$ notation once again indicates a sum over all unique pairs of neighboring sites. Periodic boundary conditions are not applied.

In this thesis, the eigenfunction $\psi_{1/2}$ in the middle of the energy spectrum is considered. That is, the wave function $\psi_{1/2}$ is chosen such that when the eigenfunctions (i.e. eigenvectors) of $\hat{H}_{\text{MBL}}$ are sorted by their energy (i.e. eigenvalue), $\psi_{1/2}$ is in the middle of the spectrum. The model has an ergodic phase at low W and a localized phase at high W. In the ergodic phase, every configuration of particles is approximately equally likely. In the localized phase however, only one or a few similar configurations are probable, dictated by the random field potentials. The exact transition point and its mechanics are unknown, but the transition is generally estimated to occur somewhere between $W = 1$ and $W = 5$, depending on other parameters such as system size and energy [37–39].

5 Simulation methods

To generate datasets for the different physical models, Monte Carlo methods are used to simulate systems at a specific temperature or other tuning parameter. These algorithms generally perform random perturbations of the system within certain constraints derived from the underlying physics. In the case of the Ising and Potts models, the Wolff cluster algorithm [40] is used. For TFIM, the simulation of a system involves learning and sampling a ground-state wave function, whereas for MBL a wave function is found through exact diagonalization of the Hamiltonian. The exact methods of simulation used for each model are explained in further detail in the sections below.

5.1 Wolff algorithm for the Ising & Potts models

The main idea behind Monte Carlo cluster algorithms [40–42] is that instead of randomly flipping the spin of one lattice site at a time, some systems can also be simulated by flipping clusters of many sites at once. These algorithms have the favorable property that they are *rejection free*, meaning that a potential update to the state is never rejected, a property that doesn't hold for e.g. the Metropolis-Hastings algorithm [43, 44].

The cluster algorithm used here is the Wolff algorithm [40], which operates as follows, given a system at temperature T:

1. Choose a random site on the lattice to be the first site of a new cluster. Push it to a stack of sites to consider.
2. Check if the stack is empty. If so, the cluster is finished, go to step 5.
3. Pop a site off the stack. Consider adding each of its neighbors to the cluster if the neighbor has an equal spin and if the pair of sites (the bond between neighbors) has not been considered yet. If the neighbor site meets these requirements and has not been added to the cluster yet, add it to the cluster with a probability of $1 - \exp\left(\frac{-J}{k_B T}\right)$. A coupling constant of $J = 2$ is used for the Ising model, while $J = 1$ is used for the Potts model. The convention for the Boltzmann constant is to use $k_B = 1$.
4. Push every site added in the previous step to the stack. Go back to step 2.
5. By definition, all sites in the cluster have the same spin. Pick a new spin state that is different, and assign it to all sites in the cluster. For the Ising model there is only one option, so the spin is flipped, but for the q-state Potts model one of the $q - 1$ remaining options is randomly chosen.

Snapshots of the system state are recorded for use in a dataset. For a lattice of N sites, the steps above are repeated until at least N sites have been flipped (changed spins) between each recorded snapshot. Given that each cluster always contains at least one site regardless of the temperature, at most N clusters have to be flipped in order to flip at least N sites.

5.1.1 Implementation

A custom implementation of the Wolff algorithm has been written in C++, based on an existing implementation in Python [45]. The custom implementation has been designed to avoid memory allocations and set-lookups as much as possible, using the following data structure for lattice sites (cells):

```
struct Cell
{
        uint32_t state;
        uint32_t flipped;
        uint32_t left_considered;
        uint32_t down_considered;
};
```

The cells themselves are stored as one array, and two stacks (also stored as single arrays) are used to keep track of indices of cells to consider and cells in the cluster. The algorithm uses a unique ID of type `uint32_t` for each simulation step (each time a cluster is created and flipped). Cells are marked with this unique ID in the fields `flipped`, `left_considered` and `down_considered` whenever they are added to the stack of cells in the cluster, and when their left or downward bond is considered for the cluster, respectively. This allows the algorithm to avoid adding or considering cell bonds twice in the same step, without keeping track of a set of each visited cell or bond, thus avoiding all associated overhead. Finally, the custom implementation uses multithreading to generate samples in parallel, allowing it to take advantage of multiple CPU cores.

Time complexity Pushing to or popping from the stack is a constant time operation, as well as marking and checking visited cells and bonds. Therefore, the running time scales approximately linearly with the number of times step 3 from section 5.1 is repeated, which is once for each site flipped. To flip N sites between each snapshot, the algorithm takes $O(N)$ time, or $O(L^2)$ for an $L \times L$ lattice.

Performance The custom implementation in C++ is a lot faster than the Python implementation: generating 5,000 samples (snapshots) of an Ising model with lattice size $L = 30$ at temperature $T = 2.27$ takes 71.3s using the single-threaded Python implementation, averaged over 10 runs, on a laptop equipped with an AMD 5800H processor. Meanwhile, the C++ implementation only takes 186ms, using eight threads on the same laptop, again averaged over 10 runs. This is a speedup of 383×. Even without multithreading, a speedup of 67× is achieved.

Further improvements In hindsight, it is possible to simplify the algorithm even further using a trick mentioned by Erik Luijten [42, p. 22]. By flipping sites directly instead of adding them to a stack and flipping them later, there is no more need to keep track of which sites have already been added to the cluster. This works because in the case of the Ising and Potts models, only sites aligned to the original spin have a non-zero probability of being added to the cluster, so flipped sites have zero probability of being flipped again via the same cluster.

5.1.2 Dataset

Using the implementation discussed in the previous subsection, a dataset is generated for the Ising model and q-state Potts models with $q \in \{3, 4, 5\}$, at various lattice sizes. For the Ising model, samples are generated for 1,000 evenly spaced values of the temperature T, with $T \in [0.05, 1.95] \cdot 2.27$.

The same is done for the Potts models with $T \in [0,2] \cdot T_c$ which is equal to $T \in [0,2] \cdot \ln\left(1+\sqrt{q}\right)^{-1}$. Table 1 shows the total number of configurations generated for each model.

model \ L (lattice size)	10	20	30	40	60	80	100
Ising	10^6	10^6	10^6	10^6	$2 \cdot 10^5$	$2 \cdot 10^5$	$2 \cdot 10^5$
Potts ($q \in 3,4,5$)	$5 \cdot 10^5$	$5 \cdot 10^5$	$5 \cdot 10^5$	$5 \cdot 10^5$	n/a	n/a	n/a

Table 1: Number of samples for each model and lattice size.

5.2 Transverse-field Ising model

The matrix representation for the TFIM Hamiltonian (26) has a $2^N \times 2^N$ shape, for a total of 2^{2N} elements for a system of N spin sites. For a system of $N = 20$ sites, this amounts to terabytes of storage if the Hamiltonian were to be stored as a dense matrix. Sparse matrices are more efficient, but still scale proportional to the number of basis states, which in 2^N. To store the Hamiltonian in an efficient format, the NetKet [46,47] toolbox is used, which offers a `LocalOperator` class to represent operators acting on a limited number of sites. NetKet comes with standard `LocalOperators` for $\hat{\sigma}_i^x$ and $\hat{\sigma}_i^z$, acting on the x- and z-spins of the particle at site i, respectively. The full Hamiltonian can then be constructed by summing and multiplying $\hat{\sigma}_i^x$ and $\hat{\sigma}_i^z$ according to the TFIM Hamiltonian:

$$\hat{H}_{\mathrm{TFIM}} = -\sum_{\langle i,j\rangle} \hat{\sigma}_i^z \hat{\sigma}_j^z - g \sum_i \hat{\sigma}_i^x \tag{29}$$

where g is the strength of the external magnetic field. The memory needed to store the TFIM Hamiltonian using `LocalOperators` only grows linearly with the number of sites, which is a big improvement over exponential growth.

Storing the Hamiltonian is only part of the equation, however. Before the ground state of the system can be sampled, it has to be computed first. NetKet has tools for this task, too. Instead of storing the wave function, ψ, as a vector of 2^N elements, a neural network is used to represent the wave function. That is, a neural network is used to represent the mapping $s \mapsto \psi(s)$ of a spin-state s to the probability amplitude $\psi(s) \in \mathbb{C}$ of finding state s. To find the wave function of the ground state, the network is then trained to minimize the expectation value of the energy, which is given by [46, p. 17]:

$$\langle \hat{H} \rangle = \frac{1}{\langle \psi | \psi \rangle} \sum_s |\psi(s)|^2 \tilde{H}(s), \qquad \langle \psi | \psi \rangle = \sum_s |\psi(s)|^2 \tag{30}$$

A division by $\langle \psi | \psi \rangle$ is applied to normalize the neural network wave function to satisfy eq. (23). Here, $\tilde{H}$ is the *local energy* defined as:

$$\tilde{H}(s) = \sum_{s'} \frac{\psi(s')}{\psi(s)} \langle s' | \hat{H} | s \rangle \tag{31}$$

which can be computed by exploiting the sparsity of $\hat{H}$ and by applying the aforementioned `Localoperators`. Even though the sum in eq. (30) runs over all possible states, NetKet can still efficiently approximate it. The expectation value is approximated by sampling a limited number of states, drawn according to the probabilities encoded by the neural network wave function, ψ. A gradient of the energy with respect to the network parameters can then be computed and used for gradient descent. This is possible because the operators that make up $\hat{H}$ are differentiable, allowing partial derivatives w.r.t. $\psi(s)$ and subsequently the network parameters to be computed by repeatedly applying the chain rule.

Once the ground state (which is a superposition of basis states) has been learned, individual spin configurations can be drawn from its distribution to generate a dataset. This process of learning and sampling the ground state has to be repeated for each sampled value of the external field strength g, since the TFIM Hamiltonian (29) - and therefore its ground state - depends on g.

Using this process, a dataset has been generated for 101 evenly spaced values of $g \in [0, 2]$, with 1,000 samples per value, for a total of 101,000 samples.

5.3 Many-body localization

The MBL dataset was generated using exact diagonalization of the Hamiltonian (28). First, all possible configurations of N particles on $L = 2N$ sites are generated, and each is assigned an index in the vector ψ representing the wave function. Once the index of every state is known, a sparse matrix representation of the MBL Hamiltonian can be constructed.

The matrix $\hat{H}_{\text{MBL}}$ is defined as:

$$\hat{H}_{\text{MBL}} = -t \sum_{\langle i,j \rangle} \left(\hat{c}_i^* \hat{c}_j + \hat{c}_j^* \hat{c}_i \right) + J \sum_{\langle i,j \rangle} \hat{n}_i \hat{n}_j + \sum_i h_i \hat{n}_i \tag{32}$$

with $\hat{H}_{\text{MBL}} \in \mathbb{C}^{S \times S}$ and $S = \frac{(2N)!}{(N!)^2}$ the number of unique configurations. The operator $\hat{n}_i$ is a sparse matrix with ones on the diagonal for any index corresponding to a state with a particle at site i, and zeros everywhere else. Each term $\hat{c}_i^* \hat{c}_j + \hat{c}_j^* \hat{c}_i$ is also represented by a sparse matrix. In this case, the matrix has ones at every combination of indices such that each of the corresponding states can be obtained from the other by moving a particle from occupied site i to empty site j, or vice-versa.

Once constructed, the eigenvectors of the Hamiltonian matrix are computed using the `scipy.sparse` Python package. The eigenvectors (wave functions) are sorted by eigenvalue (energy), and the middle one, $\psi_{1/2}$, is chosen. From this wave function, configurations can be sampled with probability $|\psi_{1/2}(s)|^2$ of sampling state s. This process has to be repeated for every random field that is sampled.

5.3.1 Datasets

Because the random field is not known to the network in advance, it's nearly impossible to tell from just one sample whether the system is in the ergodic or localized phase. If the same system is sampled many times however, there will be a clear difference in the distribution of samples. This is a result of the difference in the wave functions for both phases, which will be relatively 'flat' for the ergodic phase, but have strong peaks for only one or a few configurations in the localized phase. Because the difference between the phases is not really noticeable from single samples, a few different datasets have been constructed to explore different ways to overcome this problem. It is worth noting that other machine learning approaches to MBL or similar models have used various different types of inputs for their dataset, such as the wave function [9] itself, the entanglement spectrum [7, 17], or a time series [8] of multiple samples.

All datasets used in this thesis have been generated by sampling many different random fields for each value of W, constructing the corresponding Hamiltonian and finding the wave function in the middle of the energy spectrum. The datasets differ in how the input vectors[1]have been obtained from the wave function. The following four different methods were used:

- **Inputs of single samples**. For each random field the wave function is sampled once.
- **Inputs of 20 samples**. For each random field the wave function is sampled 20 times. The 20 samples are concatenated to form an input vector.
- **Correlations between samples as inputs**. The 20 samples from the previous method are used to generate features that tell how strongly the occupancy of each site is correlated between all 20 samples. If n_i is the number of times out of 20 that site i is occupied, then the elements x_i of the new input vector are:

  $$x_i = \left(\frac{n_i - 10}{10}\right)^2 \tag{33}$$

  Note that an FF layer would not be able to learn these features from the original 20-sample input. This is due to the fact that each feature x_i has a high value when the relevant input elements (sites) are either all high (occupied) or all low (unoccupied), but a low value otherwise. Learning the correlation features is similar to learning the XOR-problem, which is impossible to learn in a single layer and therefore impossible to learn with FFA.
- **Wave function**. For each random field the wave function is converted to probability densities, which are used as inputs.

Each dataset has been generated from 1,001 evenly spaced values of $W \in [0, 10]$, with 100 random fields sampled for each value of W, for a total of 100,100 input samples per dataset. The dataset of 20-sample inputs is further augmented to 1,001,000 input samples by taking 10 random permutations of the 20 samples for each random field. This is possible because the samples are not chronological, therefore their order within the input vector doesn't matter.

[1]In the context of MBL, the term 'sample' is used to refer to data obtained from a system whereas the terms 'input', 'input vector' or 'input sample' refer to data that is fed into a neural network.

6 Classifying phases

Inspired by existing supervised approaches [2–4, 6] using backpropagation, a classifier is trained using FFA. The network is trained to classify the phases of samples from the Ising model that are far removed from the transition-point. The goal is to then apply the trained network to samples near the transition, and to find at which temperature the network flips from predicting $T < T_c$ to predicting $T > T_c$. It is expected that this class-assignment flips at the critical temperature.

6.1 Methods

A classifier is trained using the same method as described in section 3. Samples with the correct phase-label are used as positive data, while incorrectly labelled samples are used as negative data. A softmax layer is trained on the output features from the Forward-Forward (FF) network.

The training samples are chosen to be at least a certain *margin* away from a guessed transition point, $T_{c(\text{guess})}$, and at most a certain *limit* away from this point. The dataset is truncated to ensure the training set contains an equal number of samples on both sides of $T_{c(\text{guess})}$, and that these samples are equally far removed from $T_{c(\text{guess})}$, to prevent bias. After training, the classifier is applied to samples within the selected margin from $T_{c(\text{guess})}$, generating assignment-curves for the $T < T_c$ and $T > T_c$ classes. The curves are then fitted to a logistic/sigmoid function. The temperatures at which these functions cross probability $p(\text{class}|T) = 0.5$ are averaged and used as the estimate for the critical temperature, $T_{c(\text{estimate})}$.

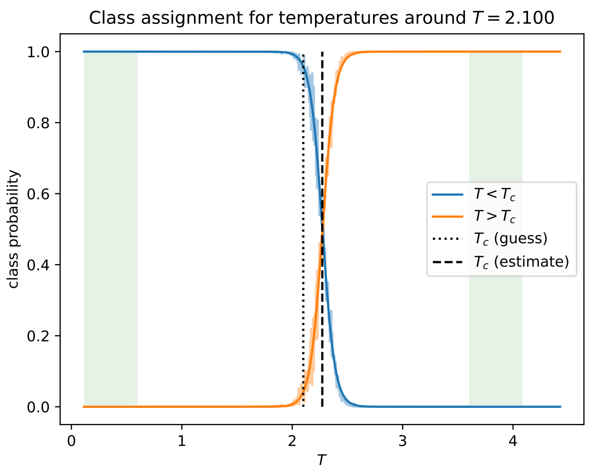
This process is then repeated for a range of temperatures surrounding the known transition-point, $T_c = \frac{2}{\ln(1+\sqrt{2})} \approx 2.269$, and the results are plotted to observe the effect of $T_{c(\text{guess})}$ on the resulting $T_{c(\text{estimate})}$.

The network architecture consists of three FF layers of 200 units each, and a softmax layer with two outputs (one for each class label). The outputs of all but the first FF layer are fed into the softmax layer. A learning rate of 0.01, a momentum parameter of 0.9 and a peer normalization weight of 0.03 were used for training the Forward-Forward networks.

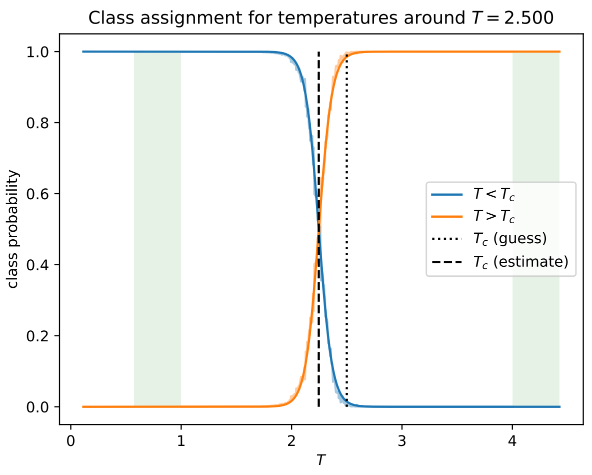
6.2 Results

Single run Figure 8 shows the class-assignment curves for a network trained on Ising configurations, for low and high initial guesses. The estimated transition points are located in approximately the same spot both times, when compared to the distance between the two initial guesses. This shows that the crossing of the class-assignment curves is pulled towards the actual transition point, rather than towards the center of the selected training data.

Scan across a range of temperatures Figure 9a shows $T_{c(\text{estimate})}$ as a function of $T_{c(\text{guess})}$ for various lattice sizes of the Ising model. The curves are relatively flat compared to the dotted line plotting $T_{c(\text{estimate})} = T_{c(\text{guess})}$, indicating that the estimated transition point is mostly dependent on the actual transition point, and comparatively little on the initial guess. For $L = 10$, the curve is steeper than it is for other lattice sizes.

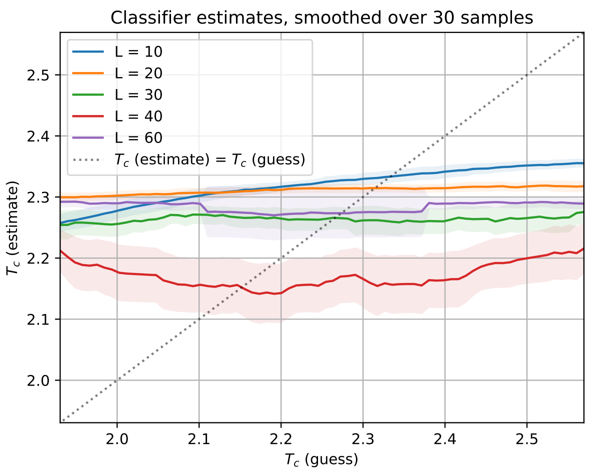


(a) Low guess: $T_{c(\text{guess})} = 2.1$, $T_{c(\text{estimate})} = 2.27$.

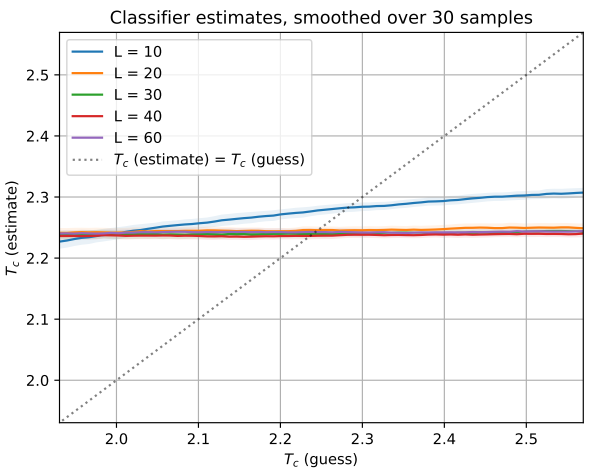


(b) High guess: $T_{c(\text{guess})} = 2.5$, $T_{c(\text{estimate})} = 2.25$.

Figure 8: Fitted class-assignment curves for lattice size $L = 30$, after training for ten epochs. Settings `margin` $= 1.5$ and `limit` $= 2$ were used. Training samples were only drawn from the areas colored in green, for a total of 184,000 training samples.



(a) Results using FFA.



(b) Results with the same network architecture, using backpropagation.

Figure 9: $T_{c(\text{estimate})}$ as a function of $T_{c(\text{guess})}$, using settings `margin` $= 1.5$ and `limit` $= 2$, trained for ten epochs. A moving average over 30 data points is plotted, and a confidence interval of three standard errors is plotted in a lighter color. The number of samples used in each experiment ranged between 70,000 near the edges of the dataset to 184,000 in the area where $T_{c(\text{guess})} \in [2.1, 2.43]$.

Figure 9b shows the same curves, except backpropagation has been used instead of FFA. Most of the curves have a much lower standard error than their FFA counterpart, indicating that results obtained with backpropagation are a lot more consistent.

The results in figure 8 show that the estimated critical temperature $T_{c(\text{estimate})}$ is generally somewhere near the true critical temperature, T_c, but it is possible that there exists a small bias towards $T_{c(\text{guess})}$. It is therefore assumed that the following relations hold:

$$T_{c(\text{estimate})} < T_{c(\text{guess})} \Rightarrow T_c < T_{c(\text{guess})}$$

$$T_{c(\text{estimate})} > T_{c(\text{guess})} \Rightarrow T_c > T_{c(\text{guess})}$$

From these relations it follows that the true critical temperature, T_c, is the point at which the curves cross $T_{c(\text{estimate})} = T_{c(\text{guess})}$. Table 2 lists those intersection points. There does not seem to be

L	10	20	30	40	60
T_c	2.333	2.315	2.266	2.152	2.273

Table 2: Intersection points where $T_{c(\text{estimate})} = T_{c(\text{guess})}$ for various model sizes, using the FFA-based classifier.

any particular order in which the curves cross this line as a function of lattice size, as would be predicted by finite-size scaling [48]. Finite-size scaling predicts the critical temperature observed for finite-sized systems to be higher than the theoretical value for infinitely-sized systems. The critical temperature is predicted to decline asymptotically towards the theoretical value as the system size increases.

6.3 Conclusion

The FFA-based classifier shows somewhat promising results. Different values of $T_{c(\text{guess})}$ yield similar values of $T_{c(\text{estimate})}$ (figures 8 and 9a), with a comparatively small bias towards the initial guess. However, the results do not obey the finite-size scaling expectation of asymptotically decreasing towards T_c as L increases. The results obtained with FFA also exhibit greater variance compared to those obtained with backpropagation, see figure 9. It stands to reason that backpropagation is better suited for supervised classification than FFA, because backpropagation optimizes the classification accuracy directly, whereas FFA optimizes different layer-local objective functions. FFA might be better suited for unsupervised learning, however, since the algorithm doesn't necessarily require supervision as long as there is a good source of negative data.

7 Discriminating by tuning parameter

7.1 Motivation

While it is possible to learn the phase transition of the Ising model using a Forward-Forward classifier, this approach has a few drawbacks. It requires labels, and blanks out a part of the dataset around the transition point, which means the transition point has to be known - to some limited degree of accuracy - in the first place. An unsupervised approach might be better suited, especially if the method is to be used in situations where the existence of a phase transition is unknown or where there is no good approximation of the transition point.

In order for an unsupervised approach to work, a source of negative data is needed. Previously, false class labels were used to generate negative examples. Instead of class labels for $T < T_c$ and $T > T_c$, the temperature T (or other tuning parameter) can be used directly as the label, using a *twohot* encoding (also used in reinforment learning [49, p. 6–7]). A twohot encoding maps T to a number of buckets covering the domain of T in the dataset. The encoding is a vector containing a number for each bucket that is equal to one if T is exactly equal to the center of the bucket, and drops off linearly till zero when T is half a bucket-width (a bucket-radius) away from the center. The buckets are spaced such that their values always add up to one.

When training a Forward-Forward network this way, labelling samples with the true value of T in the positive pass, and with a random value of T in the negative pass, the network should learn features that enable it to discriminate between samples from different temperatures. In other words, it will learn features that are unique to specific ranges of temperature. An intuitive way to understand why this should be interesting for finding phase transitions is to think of a glass of water. With the naked eye, it's impossible to tell a glass of 10°C water apart from a glass of 30°C water. But, it would be very easy to tell a glass at -10°C apart from a glass at 10°C, because the first would be frozen while the latter would be liquid. It is hypothesized that an unsupervised FF network trained as described above would learn to focus on macroscopic properties defining phase transitions, simply because this is the easiest way to achieve its learning goal, just as it's easier to tell glasses of water in different phases apart.

7.2 Methods

A Forward-Forward network with three layers of 100 hidden units each is trained as described in 7.1, using 10 input units (buckets) for the twohot-encoded temperature (or other tuning parameter). The normalized outputs from the last two layers are used as feature vectors. In the negative pass, false labels are generated by choosing a random value, T_{random}. However, this might accidentally result in choosing a value very close to the original value, T_{original}. To prevent this from happening, T_{random} is chosen at least some offset ε away from T_{original}, i.e. $|T_{\text{random}} - T_{\text{original}}| \geq \varepsilon$.

For the transverse-field Ising and many-body localization models, the approach is exactly the same, except that rather than the temperature, the field strengths g and W are used as tuning parameter.

Once trained, the network is then run on a validation set, using a neutral temperature label (i.e. each bucket gets a value of $\frac{1}{\#\text{ buckets}}$, which is 0.1 in this case). The outputs of the network are recorded for each sample. Three ways of determining T_c from the learned features are described in the following subsection.

The implementation of this training procedure is described in appendix B.3. Table 3 lists the hyperparameters that were used for each physical model. The learning rates used for the Ising

	Ising	Potts	TFIM	MBL
learning rate	$0.1/L$	$0.1/L$	0.01	0.01
momentum	0.9	0.9	0.9	0.9
peer normalization	0.03	0.03	0.03	0.03
peer normalization delay	0.9	0.9	0.9	0.9
weight decay	0.01	0.01	0.01	0.01
offset ϵ	0.1	0.05	0.05	0.2
training epochs	10	10	4	6

Table 3: Hyperparameters used for each model.

and Potts models are scaled down as the system size L increases, because the network has been observed to be unable to learn on large systems otherwise. Secondly, the minimum offset ϵ has been adjusted per model to reflect the scale of the tuning parameter domain. Finally, the training is shorter for TFIM and MBL compared to the Ising and Potts models, because the risk of overfitting is higher considering that the datasets are smaller.

7.2.1 Finding the critical point

Class-based The two phases form two classes in the dataset. Samples from the same class should be similar, and samples from different classes should be dissimilar. It is hypothesized that the output features of a trained network can be used to determine how similar any two samples are.

The mean feature (i.e. output) vector, $\mathbf{v}_T$ is computed for each temperature T (or other tuning parameter) in the dataset:

$$\mathbf{v}_T = \frac{1}{|X_T|} \sum_{\mathbf{x} \in X_T} \mathbf{F}(\mathbf{x}) \tag{34}$$

where X_T is the set of samples at temperature T, and $\mathbf{F}(\mathbf{x})$ is the function representing the neural network, mapping an input $\mathbf{x}$ to a feature vector. A measure for how similar these feature vectors are to each other is then computed. The similarity between any two feature vectors $\mathbf{v}$ and $\mathbf{w}$ is defined as a function $s(\mathbf{v}, \mathbf{w})$, equal to the dot product of $\mathbf{v}$ and $\mathbf{w}$:

$$s(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} \tag{35}$$

The transition point, T_c, is then defined as the point where the mutual similarity of the feature vectors $\mathbf{v}_T$ within the classes $T < T_c$ and $T > T_c$ is maximized, and the similarity of any two vectors from different classes is minimized. The exact heuristic to maximize is as follows:

$$h(T_c) = \frac{1}{2} S_{T_1<T_c, T_2<T_c} + \frac{1}{2} S_{T_1>T_c, T_2>T_c} - S_{T_1<T_c, T_2>T_c} \tag{36}$$

Where $S_{\mathrm{cond}(T_1,T_2)}$ is the mean similarity of all $\mathbf{v}_{T_1}$ and $\mathbf{v}_{T_2}$ for which $\mathrm{cond}(T_1, T_2)$ holds true:

$$S_{\mathrm{cond}(T_1,T_2)} = \left(\sum_{\mathrm{cond}(T_1,T_2)} s(\mathbf{v}_{T_1}, \mathbf{v}_{T_2}) \right) \left(\sum_{\mathrm{cond}(T_1,T_2)} 1 \right)^{-1} \tag{37}$$

Note that in eq. (36), $S_{T_1<T_c,T_2>T_c}$ is the same as $S_{T_1>T_c,T_2<T_c}$, since T_1 and T_2 are interchangeable.

Similarity-based At the critical temperature, the system shows properties of both the ordered and unordered phase. Therefore, the mean feature vector of the critical temperature should be the one that is (closest to) equally similar to all other feature vectors. The corresponding heuristic to maximize is:

$$h(T_c) = -\mathrm{std}\left\{ s\left(\mathbf{v}_{T_c}, \mathbf{v}_T\right) \mid T \neq T_c \right\} \tag{38}$$

Where $\mathbf{v}_T$ represents the mean feature vector (34) of temperature T and $s(\mathbf{v}, \mathbf{w})$ the similarity function (35), both as defined previously.

Order parameter As described in subsection 7.1, the network is expected to learn features related to the macroscopic properties defining the phase transition. An order parameter, such as the squared magnetization M^2 for ferromagnetic models, is such a property. This property can be extracted by finding the corresponding axis in the space of feature vectors. Let $\mathbf{F}(\mathbf{x})$ be the function representing the neural network, mapping an input Ising configuration $\mathbf{x}$ to the corresponding feature vector. Furthermore, let $\mathbf{x}^-$ and $\mathbf{x}^+$ be the completely negatively and positively magnetized configurations, respectively, and R a set of random configurations. We can then define the vectors representing the most ordered and most disordered samples, $\mathbf{v}_o$ and $\mathbf{v}_d$ respectively:

$$\begin{aligned} \mathbf{v}_o &= \frac{1}{2}\left(\mathbf{F}(\mathbf{x}^-) + \mathbf{F}(\mathbf{x}^+)\right) \\ \mathbf{v}_d &= \frac{1}{|R|}\sum_{\mathbf{x}\in R} \mathbf{F}(\mathbf{x}) \end{aligned} \tag{39}$$

For practical purposes, in this work R will be a set of 1,000 random configurations. When applying this method to models other than the Ising model, the definition of $\mathbf{v}_o$ is changed appropriately. The vector $\mathbf{v}_o$ is always constructed by averaging feature vectors from all maximally ordered samples. From $\mathbf{v}_o$ and $\mathbf{v}_d$ we can compute a vector $\mathbf{a}$ representing the 'order axis', which can be rescaled to arrive at an order operator $\mathbf{o}$ such that $\mathbf{o} \cdot \mathbf{v}_o = 1$ and $\mathbf{o} \cdot \mathbf{v}_d = 0$:

$$\mathbf{a} = \mathbf{v}_o - \mathbf{v}_d \frac{\mathbf{v}_o \cdot \mathbf{v}_d}{\|\mathbf{v}_d\|^2}, \qquad \mathbf{o} = \frac{\mathbf{a}}{\mathbf{a} \cdot \mathbf{v}_o} \tag{40}$$

Finally, the learned order parameter of a sample $\mathbf{x}$ is $\mathbf{o} \cdot \mathbf{F}(\mathbf{x})$. The order parameter can now be computed as a function of temperature, given the set X_T of samples at temperature T in the dataset:

$$o(T) = \frac{1}{|X_T|} \sum_{\mathbf{x}\in X_T} \mathbf{o} \cdot \mathbf{F}(\mathbf{x}) \tag{41}$$

The transition point is defined as the temperature at which $o(T)$ changes the fastest, i.e. the heuristic to maximize is:

$$h(T_c) = |o'(T_c)| \tag{42}$$

7.2.2 Variations

The following paragraphs describe two variations on the methods from the previous subsection, that are also tested.

Normalized similarity For the class-based and similarity-based methods, a normalized version of $s(\mathbf{v}, \mathbf{w})$ is tested as well, which replaces eq. (35) with:

$$s(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\| \cdot \|\mathbf{w}\|} \tag{43}$$

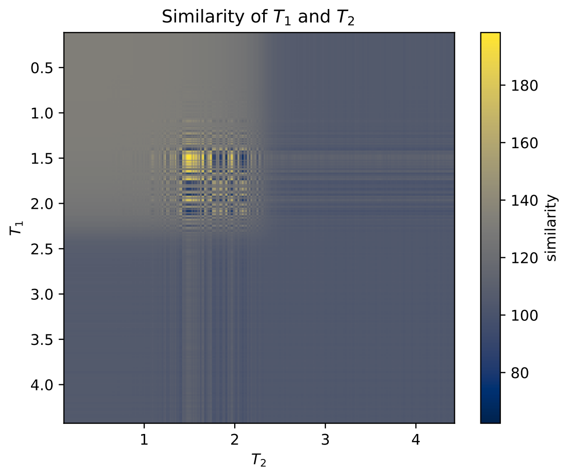
This calculates the similarity as the cosine of the angle between two feature vectors, leaving the length of the vectors out of the equation. Calculating similarity this way ensures that mean feature vectors with a higher magnitude aren't counted as being more similar to all other vectors. However, it could be argued that the original definition of $s(\mathbf{v}, \mathbf{w})$ is better, since the feature vectors - the outputs of the FF network - are already normalized before being averaged to produce the mean feature vector for a specific temperature. Thus, mean feature vectors with a lower magnitude can only occur if the samples at that temperature produce feature vectors that point in very different directions, meaning they are already not similar to each other. It therefore makes sense to just use the dot product between two mean feature vectors $\mathbf{v}_{T_1}$ and $\mathbf{v}_{T_2}$, which gives the mean similarity of any combination of samples at T_1 and T_2.

Using true labels All methods are also tested using true labels instead of neutral labels as input when generating feature vectors. Ideally, neutral labels should be used because they guarantee that the output features are only a function of the input sample, which is a nice property to have. This is especially true because the method of measuring similarities only makes sense if identical samples produce identical feature vectors regardless of the temperature at which they occur. Nonetheless, the network is only trained using true or false labels, not neutral ones, so it's worth investigating if there is a benefit to using true labels.

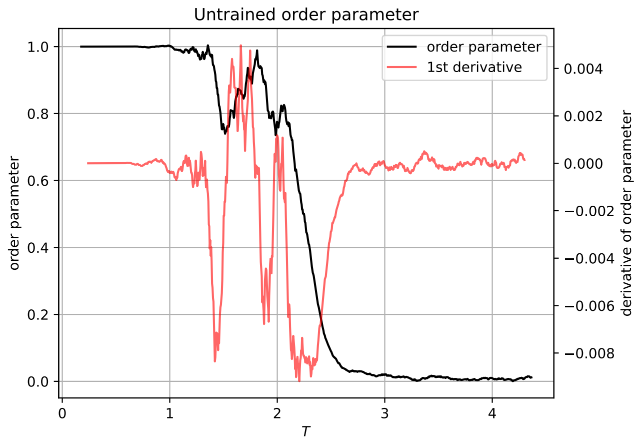
7.3 Results

7.3.1 Ising model

Without training To asses whether the Forward-Forward network learns useful features, it is instrumental to compare results from a trained network with those from one that has not been trained. Figure 10 plots the similarity matrix and calculated order parameter (41) obtained from an untrained network. It is interesting to note that both subfigures already show some phase-related structure: the similarity matrix has a bright spot representing the ordered phase, and the 'order parameter' shows a steep decline from $o(T) \approx 1$ to $o(T) \approx 0$ around the critical temperature, albeit with a lot of additional noise. Because the ordered phase consists of states where most spins (therefore most inputs to the network) have the same value, it makes sense that the network outputs - despite not being trained - show high mutual similarity within the ordered phase. The same doesn't happen in the disordered phase, because the samples have uncorrelated spins/inputs. Consequently, because there are still characteristic features associated with the ordered phase, the calculated order parameter already behaves somewhat as expected, declining around the critical point.



(a) Similarity matrix.



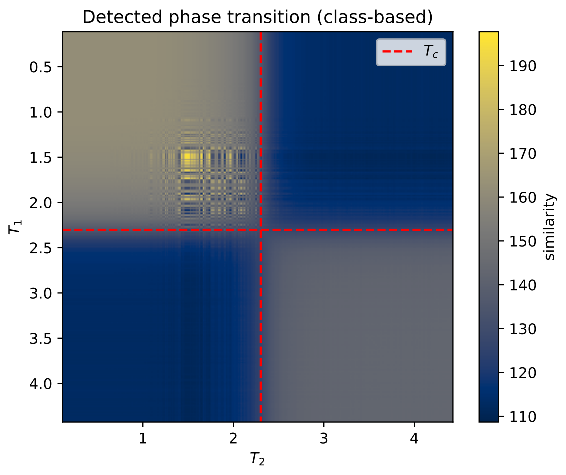
(b) Order parameter.

Figure 10: Results from an untrained Forward-Forward network, on Ising configurations with lattice size $L = 30$. (a) Heatmap of $s(\mathbf{v}_{T_1}, \mathbf{v}_{T_1})$ for all combinations of T_1 and T_2, from now on referred to as 'similarity matrix'. (b) Order parameter as computed by eq. (41). The graph has been smoothed using a moving average over 30 data points. The derivative was calculated after smoothing, and then smoothed again. The resulting graph is still very noisy, regardless.

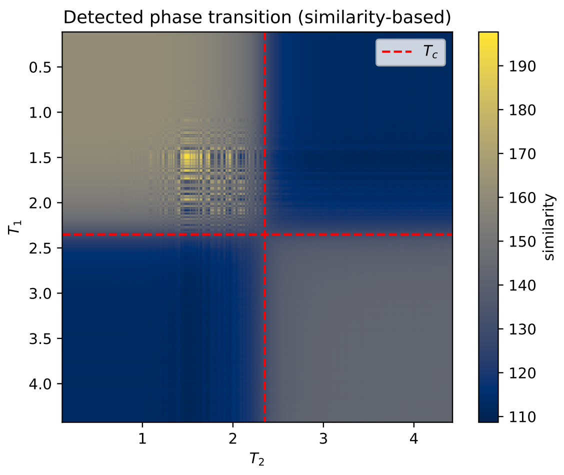
This puts into perspective any results obtained using neural networks on the Ising model. In fact, it can be argued that while simple physical models are a good exercise to use as proof of a concept, neural networks are overkill for any model on which PCA is already effective. After all, there is no need for a complicated nonlinear machine learning model when a simple linear dimensionality reduction suffices.

Single run Moving on, figure 11 shows the similarity matrix after training, as well as the detected critical temperature T_c for the class-based and similarity-based methods. It is clear from the figure that the network has learned something: there are distinct bright patches representing the two phases, and dark patches showing low similarity between samples from different phases. The detected critical temperatures are in the vicinity of the known value $T_c \approx 2.269$. Figure 12a shows the learned order parameter, which has a lot less noise after training. In fact, the learned order parameter closely resembles the squared magnetization M^2, plotted in subfigure 12b.

Statistics over many runs Figure 13 shows statistics from many runs. All methods converge to a value, but only the order parameter method seems to asymptotically decline towards the theoretical value for T_c. Using the similarity-based and order parameter method, predictions of $T_c = 2.273 \pm 0.002$ and respectively $T_c = 2.272 \pm 0.002$ are obtained at $L = 100$, which is very close to the theoretical value. This is to be expected for the order parameter method, because it closely resembles the method of empirically calculating T_c from the derivative of M^2. The fact that the class-based method converges to a different value does not necessarily invalidate this method. Rather, it uses a slightly different definition of what exactly marks the critical point, which can be hard to pin down in the case of a continuous phase transition such as that of the Ising model.

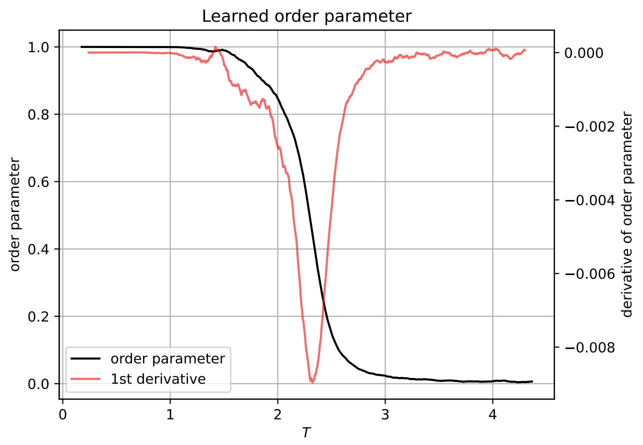


(a) Class-based method: $T_c = 2.30$.

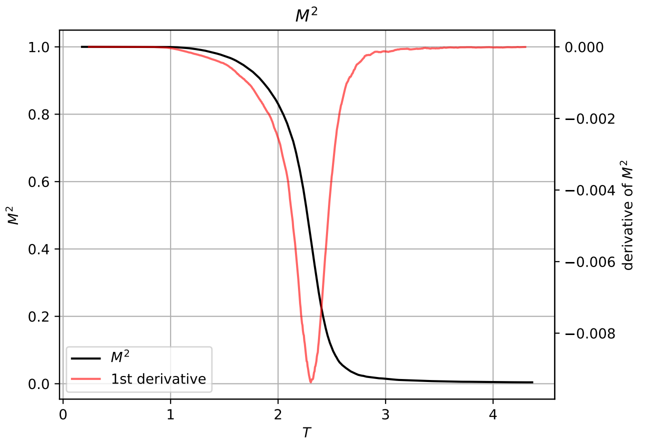


(b) Similarity-based method: $T_c = 2.35$.

Figure 11: Two identical similarity matrices from a network trained on Ising configurations with $L = 30$. The dashed red line shows the detected phase transition for (a) the class-based method and (b) the similarity-based method.

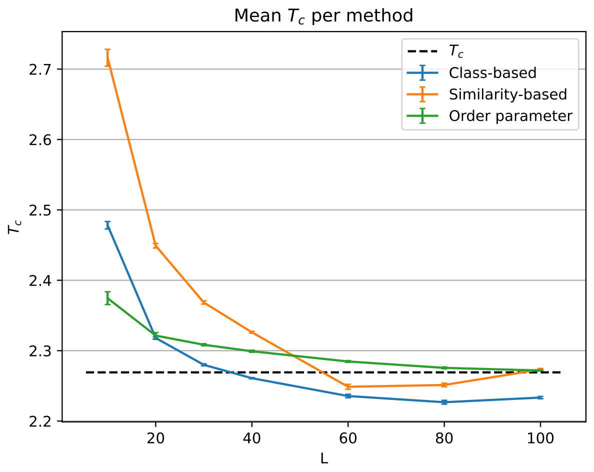


(a) Learned order parameter.

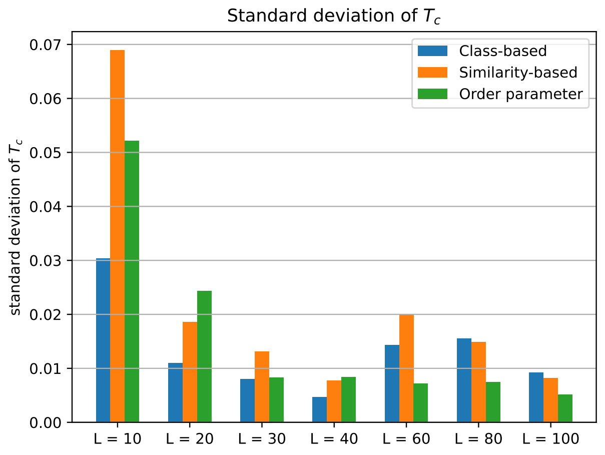


(b) M^2 calculated directly from samples.

Figure 12: (a) The learned order parameter after training the network on Ising configurations with $L = 30$. (b) The mean squared magnetization M^2 of those same configurations. Both graphs have been smoothed using a moving average over 30 data points. The derivatives were calculated after smoothing, and then smoothed again to prevent noise from being amplified in the derivatives.



(a) Mean T_c obtained with each method, as a function of L.



(b) Standard deviation of T_c for each method, as a function of L.

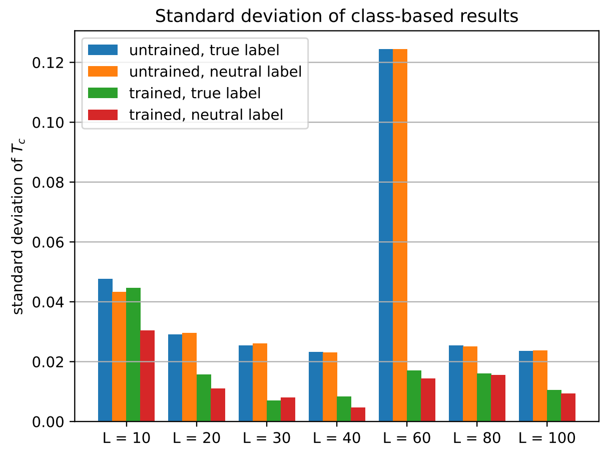
Figure 13: Results for different values of L. Each data point represents the mean or standard deviation of 300 runs. Error bars of three standard errors are plotted in (a). The dashed line represents the theoretical critical temperature.

Subfigure 13b shows the standard deviation of the predicted T_c. There is no clear 'winner' here; all methods have approximately similar standard deviations. It is worth noting that both the predictions for T_c and their standard deviations decline as the lattice size L increases. This is due to finite-size scaling [48] and the fact that the order parameter of larger systems is less sensitive to fluctuations caused by random spin flips, which allows the critical temperature to be determined with higher accuracy.

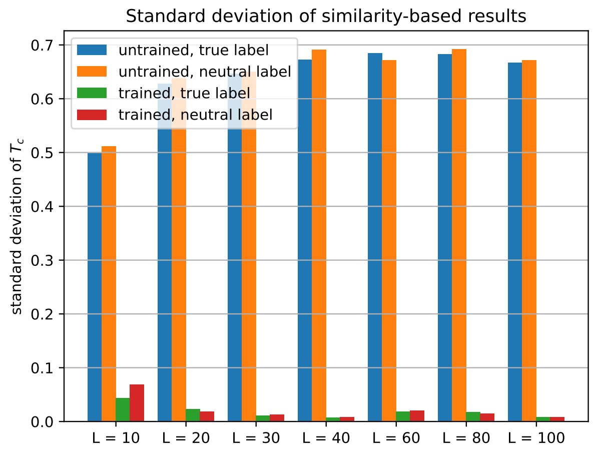
It is once again good to see what happens when the network isn't trained. Figure 14 shows the effect of training and the choice of neutral or true labels on the standard deviation of the predicted T_c. It is clear that the similarity-based approach only works after training, and that training generally reduces the standard deviation of the result. This demonstrates that even though the phase transition is still detected without training (using the class-based and order parameter methods), the network does really learn and improve through training.

There does not seem to be any clear benefit to using true labels, so from now on only neutral labels will be used since they have the favorable property that the network output is only a function of the input sample, as discussed in 7.2.2.

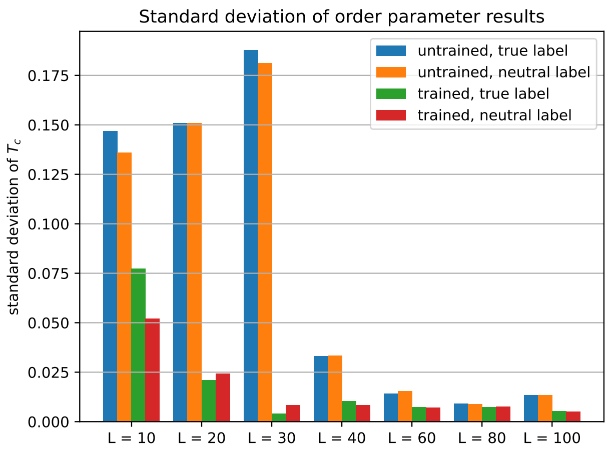
Finally, figure 15 compares results with and without normalized similarity (7.2.2). For the similarity-based method, using normalized similarity seems to be problematic at lattice sizes $L \geq 80$, while at small values of L it seems beneficial to use normalization. Other than that, there isn't any clear advantage to either option as far as the class-based method is concerned.



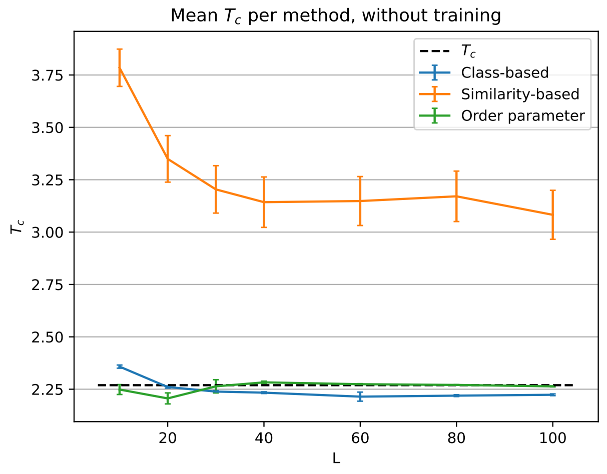
(a) Class-based.



(b) Similarity-based.

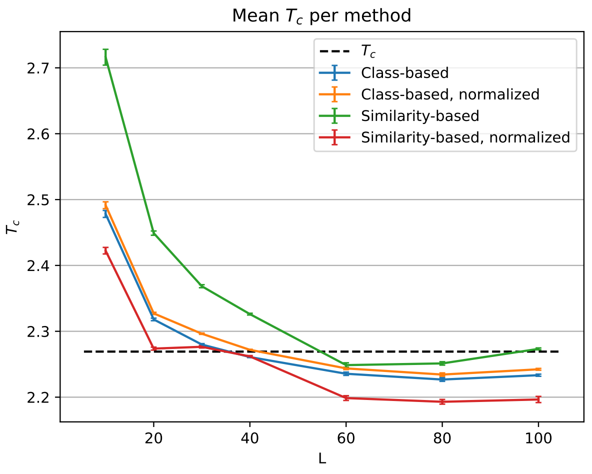


(c) Order parameter.

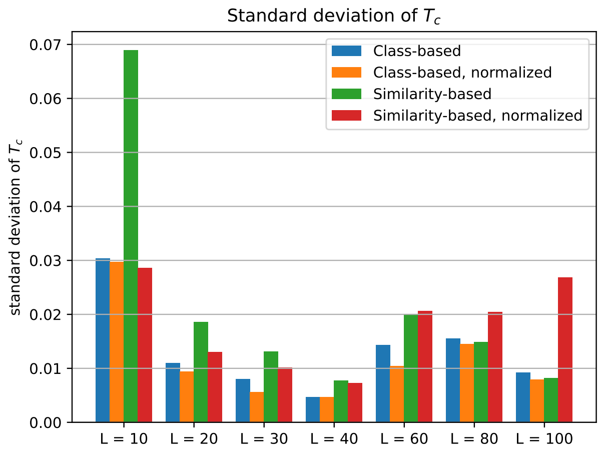


(d) Results without training.

Figure 14: (a-c)Standard deviation of T_c as a function of L for each method, for any choice of training or not training, and using neutral or true labels. (d) T_c as a function of L for each method, if the network is not trained. Error bars of three standard errors are plotted in (d).



(a) Mean T_c obtained with each method, as a function of L.

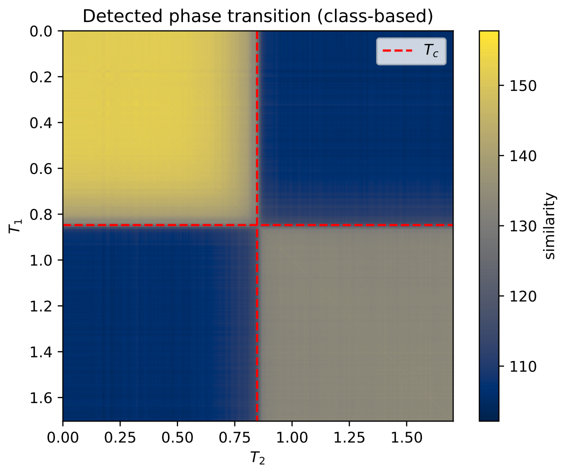


(b) Standard deviation of T_c for each method, as a function of L.

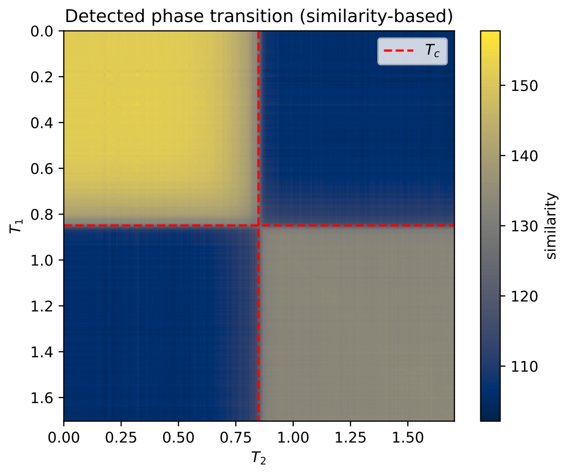
Figure 15: Results for different values of L. Each data point represents the mean or standard deviation of 300 runs. Error bars of three standard errors are plotted in (a).

7.3.2 Potts model

The experiments from the previous subsection have been repeated with the Potts model. Results from a single run are plotted in figures 16 and 17, while results from many runs over all values of $q \in \{3, 4, 5\}$ and $L \in \{10, 20, 30, 40\}$ have been summarized in figure 18.



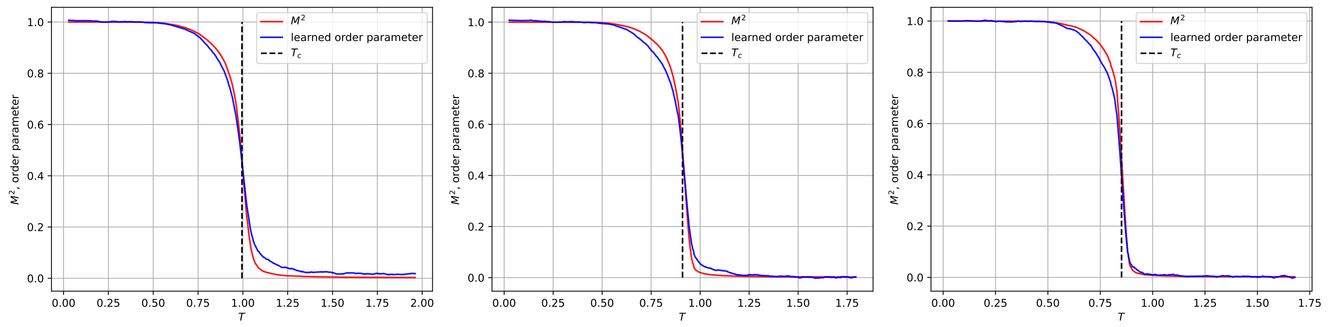
(a) Class-based method: $T_c = 0.847$.



(b) Similarity-based method: $T_c = 0.849$.

Figure 16: Two identical similarity matrices from a network trained on Potts configurations with $q = 5$ and $L = 30$. The dashed red line shows the detected phase transition for (a) the class-based method and (b) the similarity-based method.

Figure 16 shows the similarity matrix obtained from a network trained on the 5-state Potts model, which has $T_c \approx 0.852$. Similar to the results from the Ising model, there is a chessboard pattern showing that samples at $T < T_c$ are mutually similar, and that the same applies to samples at $T > T_c$. The square for $T < T_c$ is brighter than the one for $T > T_c$, which can be explained by the fact that in the ordered phase most spins are aligned to one of q distinct values. This means the network gets a lot of very similar inputs at $T < T_c$, causing the outputs to also be more similar compared to $T > T_c$.



(a) $q = 3$, prediction: $T_c = 0.999$, theoretical value: $T_c = 0.995$.

(b) $q = 4$, prediction: $T_c = 0.921$, theoretical value: $T_c = 0.910$.

(c) $q = 5$, prediction: $T_c = 0.855$, theoretical value: $T_c = 0.852$.

Figure 17: Learned order parameter (blue), squared magnetization (red) and theoretical value for T_c (dashed line) for different values of q at $L = 30$. The predictions for T_c using the first derivative of the learned order parameter are given in the subfigure captions. All curves are smoothed using a moving average over 30 data points.

The learned order parameter for each $q \in \{3, 4, 5\}$ is plotted in figure 17, alongside the squared magnetization, M^2. Both the learned order parameter and the squared magnetization decline sharply around the critical temperature, as expected. Again, this sharp decline allows the critical temperature to be determined accurately using the first derivative of the learned order parameter.

The Potts model experiment has been repeated many times in order to collect statistics. The mean and standard error of these results are plotted in figure 18. Like with the Ising model, the results converge to a value near the true critical temperature as the system size increases. The order parameter method again gets closest to the true value, for the same reason as stated in 7.3.1, being that it closely resembles the method of empirically calculating T_c using the first derivative of M^2.

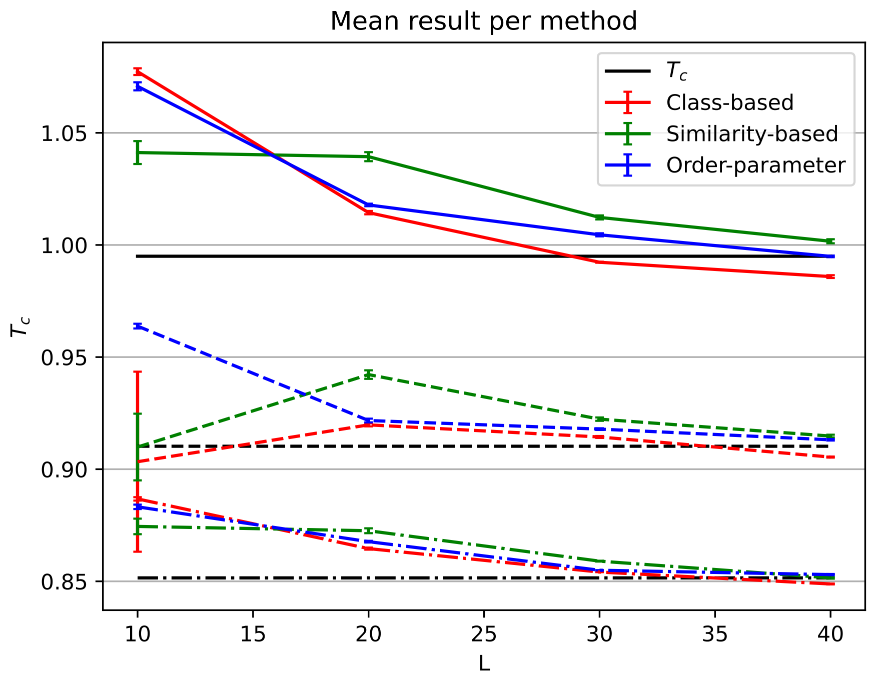
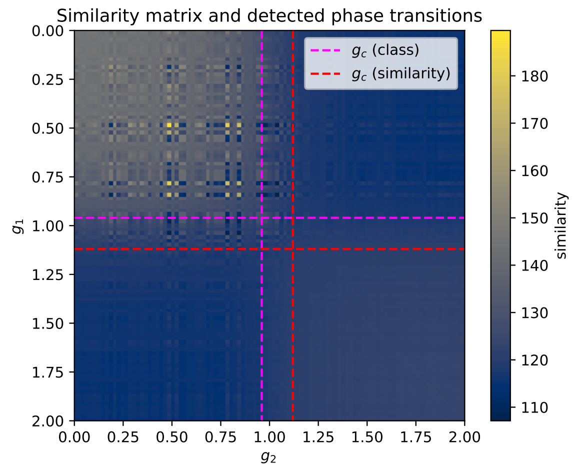


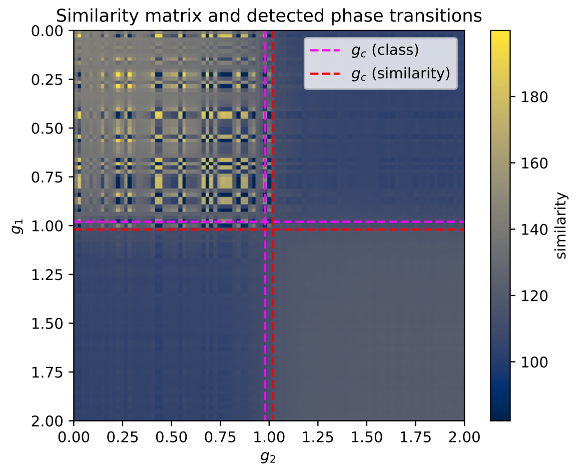
Figure 18: Results for different values of L. Results for $q = 3, 4, 5$ are indicated by the solid, dashed and dash-dotted lines respectively. The theoretical values of T_c are plotted in black for each q. Each data point represents the mean and confidence interval of three standard errors over 300 runs.

7.3.3 Transverse-field Ising model

The discriminative experiment has been repeated with a dataset of TFIM chains of $N \in \{20, 40\}$ particles.



(a) $N = 20$.



(b) $N = 40$.

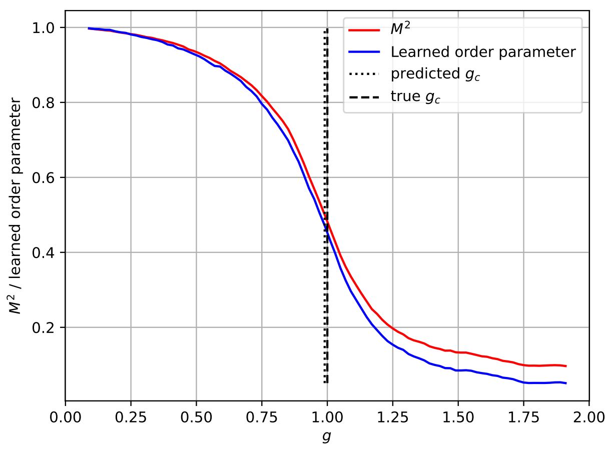
Figure 19: Similarity matrices and detected phase transitions for the class-based and similarity-based methods, for TFIM chains of length $N \in \{20, 40\}$.

Figure 19 shows similarity matrices from a single run. The chessboard pattern is much less clear - and also noisier - compared to the results for the Ising and Potts models. This can likely be attributed to a combination of the following factors:

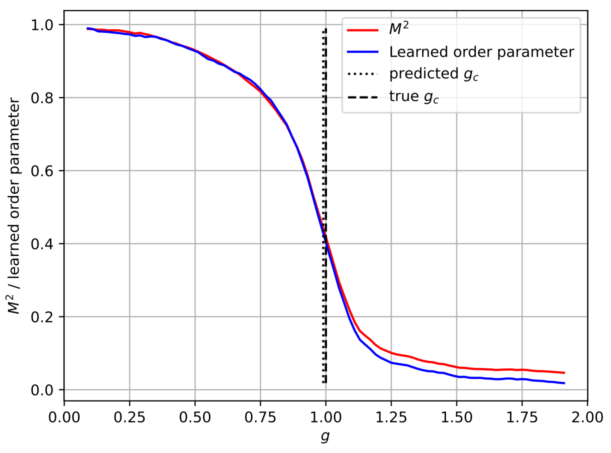
- The dataset is smaller and less granular, with only 101,000 samples spread over 101 values of g, as opposed to the datasets for the Ising and Potts models, which have more samples spread over 1,000 values of T. This makes the TFIM dataset more susceptible to noise.
- The samples consist of only 20 or 40 spin values, rather than at least 100 for the Ising and Potts models with $L = 10$. This makes individual samples within the TFIM dataset more susceptible to noise, i.e. a single spin-flip has a much bigger impact on the total magnetization.
- The dataset is generated using an approximation of the wave function, rather than an exact method. This means the data source might not perfectly simulate the underlying physics, whereas for the Ising and Potts models the generated samples are distributed exactly according to the Boltzmann distribution.

Despite these challenges, the phase transition is still detected by both the class-based and similarity-based methods. The learned order parameter plotted in figure 20 also closely matches M^2, especially for $N = 40$. The critical tuning parameter obtained from the first derivative of the learned order parameter is very close to the theoretical value for g_c.

Table 4 lists the mean results obtained from many runs. All methods predict a critical point near the known theoretical value, and interestingly, all predictions for $N = 40$ are closer to the known



(a) $N = 20$.



(b) $N = 40$.

Figure 20: Learned order parameter (blue), squared magnetization (red), and theoretical value and prediction for g_c (dashed and dotted lines respectively) for TFIM chains of length $N \in \{20, 40\}$. All curves are smoothed using a moving average over 10 data points.

value than their $N = 20$ counterpart. Similar to the results from the Ising and Potts models, the predictions seem to get closer to the true value as the system size increases.

	$N = 20$	$N = 40$
Class-based	0.948 ± 0.002	0.971 ± 0.002
Similarity-based	1.052 ± 0.005	1.022 ± 0.001
Order parameter	0.959 ± 0.002	0.973 ± 0.002

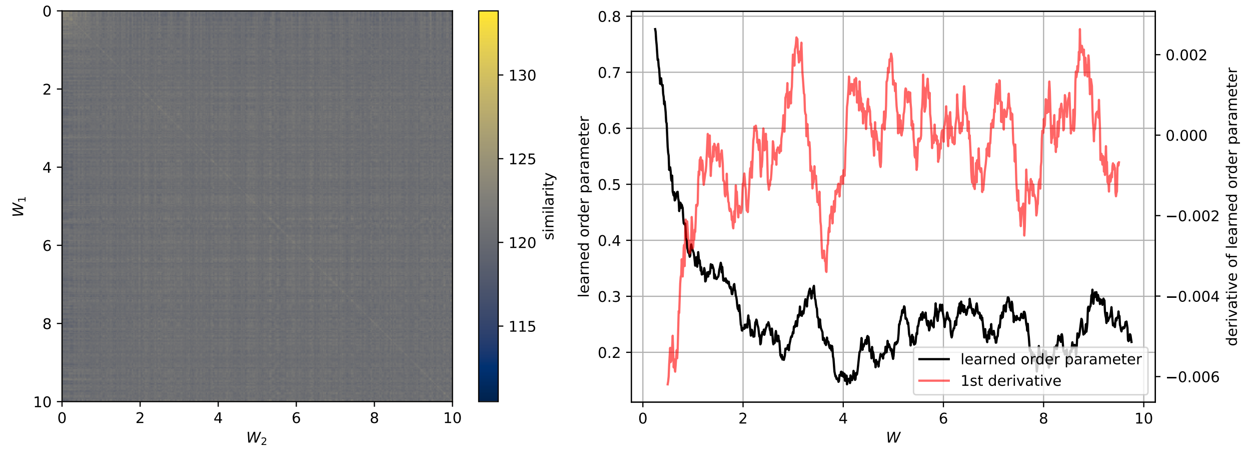
Table 4: Mean predictions for g_c with confidence intervals of three standard errors for all methods and system sizes, obtained over 500 runs.

It stands out from the results on all models covered thus far that the class-based method tends to underestimate the position of the critical point. This is explained by the fact that the ferromagnetic models under study are highly ordered at low values of tuning parameter and highly disordered at high values, with a small grey area near the critical point. Generally, feature vectors of samples from the ordered phase have a higher mutual similarity. This causes the class-based heuristic (36) to favor a lower critical point, since putting it at the edge of the highly ordered area increases the average mutual similarity of the ordered class. It also causes the disordered class to contain part of the grey area, lowering its average mutual similarity. However, the heuristical gain from lowering the critical point to the edge of the highly ordered area outweighs the loss from including the grey area in the disordered class. In conclusion, the class-based method tends to find a lower bound for the critical tuning parameter in the ferromagnetic models covered so far, but this will not necessarily be the case for non-ferromagnetic models.

7.3.4 Many-body localization

For the MBL model there isn't a clear definition of 'most ordered' and 'most disordered' samples like there is for the previous three models. Instead, the learned order parameter is computed as described in section 7.2.1 by using the input samples[2]at $W = 0$ and $W = 10$ to calculate $\mathbf{v}_o$ and $\mathbf{v}_d$ respectively. As such, the term 'order parameter' is used loosely in the context of MBL, and can more accurately be thought of as an indicator of the ergodic phase.

Figure 21 shows the results obtained with the single sample input dataset. As expected, there is no clear distinction between samples from different values of W in the similarity matrix. The learned order parameter is very noisy even after smoothing. It also doesn't drop sharply in between relatively flat parts, which would be expected for a phase transition. As expected, using single samples as input doesn't provide much insight.



(a) Similarity matrix.

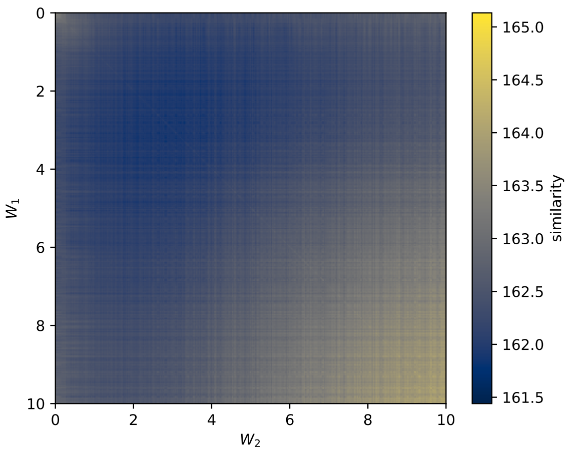
(b) Learned order parameter and its first derivative.

Figure 21: Results from the single sample input dataset. The learned order parameter has been smoothed using a moving average over 50 data points. Its derivative was calculated after smoothing, and then smoothed again using the same method.

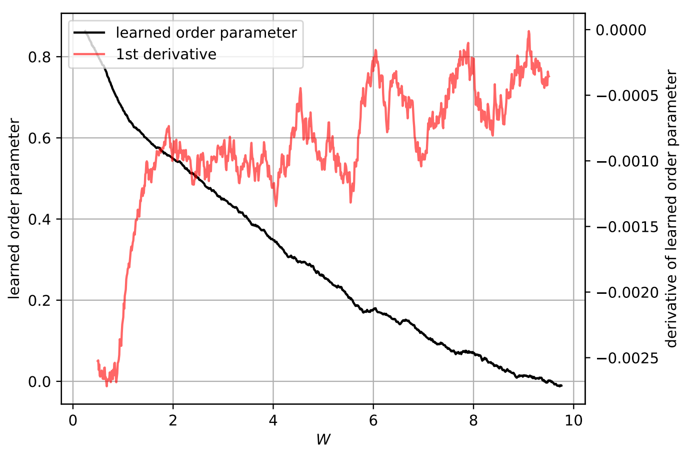
Figure 22 shows the results from the 20-sample input dataset. The similarity matrix again doesn't show the chessboard pattern that would be expected for a phase transition. Note also that the colorbar only covers a very small domain compared to other similarity matrices such as in e.g. figure 11. This means that although there seem to be brighter and darker areas, the difference is minimal. This is even true when compared to the results for TFIM (figure 19). The learned order parameter is less noisy compared to figure 21b, but again doesn't have the shape that would be expected for a phase transition.

Figure 23 shows the results from the correlation features dataset. The similarity matrix resembles a very blurry chessboard pattern. If this is the case, then visually it seems like the critical point is approximately $W_c \approx 4.5$, which is also near where the similarity-based method estimates it to be.

[2]In the context of MBL, the term 'sample' is used to refer to data obtained from a system whereas the terms 'input', 'input vector' or 'input sample' refer to data that is fed into a neural network.



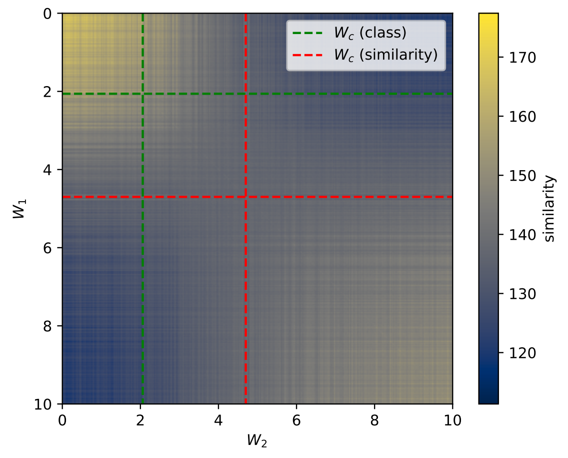
(a) Similarity matrix.



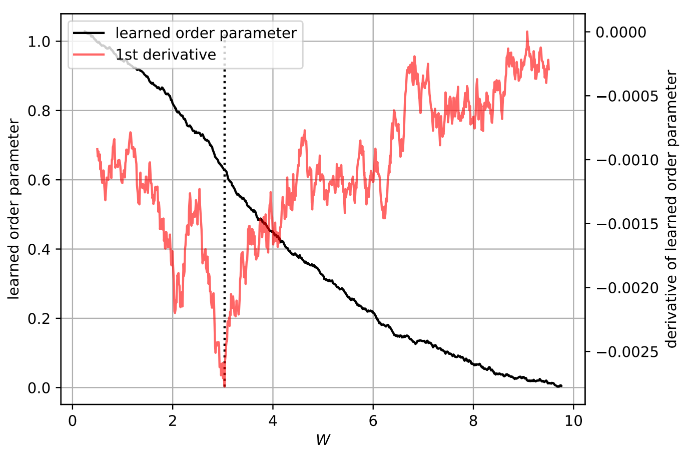
(b) Learned order parameter and its first derivative.

Figure 22: Results from the 20-sample input dataset. The learned order parameter has been smoothed using a moving average over 50 data points. Its derivative was calculated after smoothing, and then smoothed again using the same method.

The derivative of the learned order parameter is very noisy even after smoothing, but does seem to have a dip near $W \approx 3$ indicating a possible phase transition. Although the results in figure 23 seem to hint at a phase transition, they are not convincing enough to draw any conclusions.



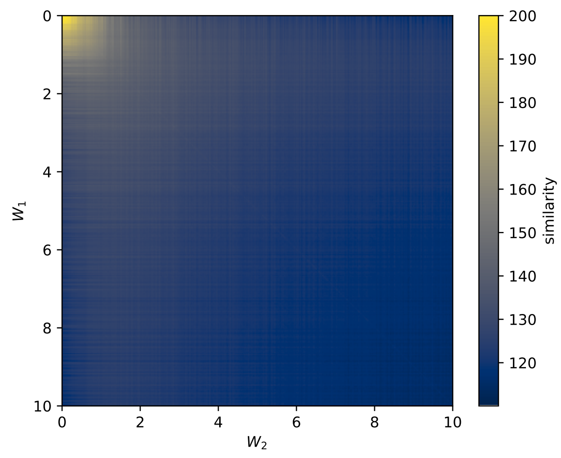
(a) Similarity matrix.



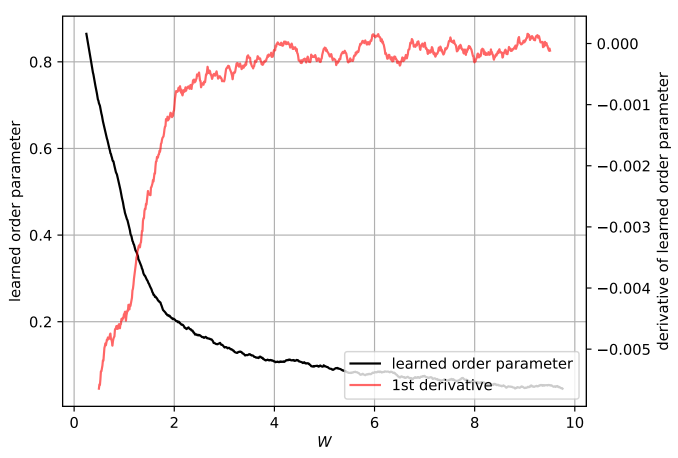
(b) Learned order parameter and its first derivative.

Figure 23: Results from the correlation features dataset. Critical points indicated by each method are marked in dashed (a) and dotted (b) lines. The learned order parameter has been smoothed using a moving average over 50 data points. Its derivative was calculated after smoothing, and then smoothed again using the same method.

Figure 24 shows the results from the wave function dataset. From the similarity matrix it is clear that the wave functions near $W = 0$ are all very similar, as expected. There is again no chessboard pattern indicating a phase transition. The learned order parameter behaves similar to the one from the 20-sample input dataset, but it declines steeper at low values of of W. Again, the characteristic shape that would be expected for a phase transition is not present.



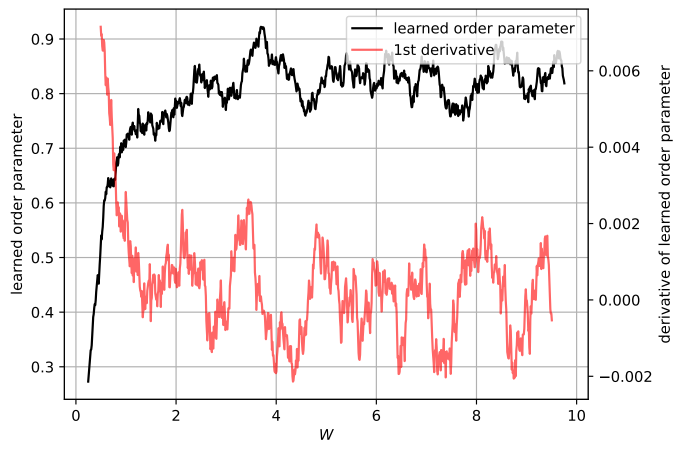
(a) Similarity matrix.



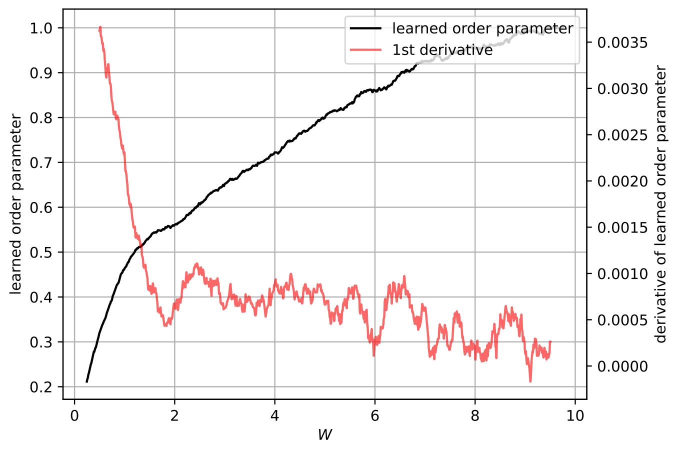
(b) Learned order parameter and its first derivative.

Figure 24: Results from the wave function dataset. The learned order parameter has been smoothed using a moving average over 50 data points. Its derivative was calculated after smoothing, and then smoothed again using the same method.

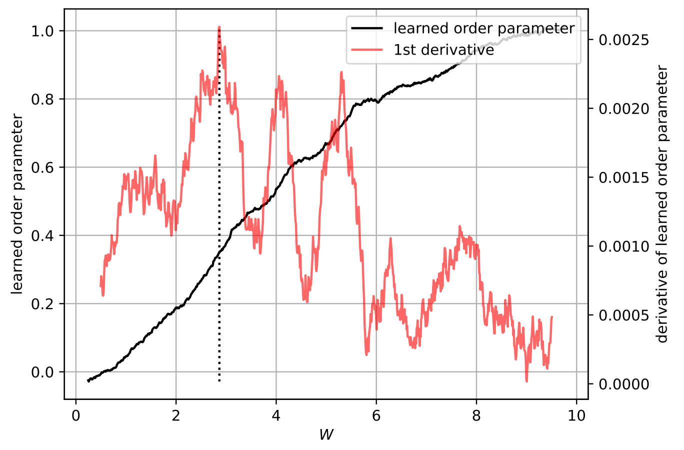
Finally, the question remains whether the results from the learned order parameters would be more helpful if the opposite choice was made for the 'most ordered' and 'most disordered' input samples. Figure 25 plots the learned order parameters that result from choosing inputs at $W = 10$ to calculate $\mathbf{v}_o$ and inputs at $W = 0$ to calculate $\mathbf{v}_d$. These result are almost exactly a vertically mirrored version of the previous learned order parameters. Subfigure 25c again seems to have a peak around $W \approx 3$, but the derivative is again too noisy to draw a conclusion about the existence and location of a critical point.



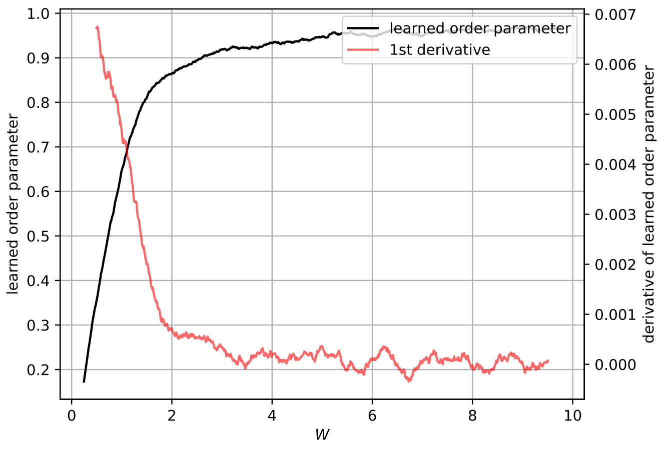
(a) Single sample input dataset.



(b) 20-sample input dataset.



(c) Correlation features dataset.



(d) Wave function dataset.

Figure 25: Learned order parameter results for each dataset when assuming the order to ascend with W, rather than descend. The learned order parameters have been smoothed using a moving average over 50 data points. Their derivatives were calculated after smoothing, and then smoothed again using the same method. The dotted line in (c) marks the peak in the first derivative.

7.4 Conclusion

Results show that the discriminative approach efficiently finds the phase transition of the Ising, Potts and transverse-field Ising models. It is also able to learn an order parameter, without any prior knowledge about the physical model. Reconstructing the order parameter from the learned features requires prior knowledge that there exists an order parameter, but the network would have learned the same features regardless of this prior knowledge. While the ferromagnetic models are too easy to be a meaningful test of the full capabilities of the algorithm, the results do demonstrate the ability to find the critical point in a model-agnostic manner.

On the other end of the spectrum, the MBL model seems to be too complex to be studied using FFA. Some results seem to point to a phase transition in MBL, but those results were obtained from manually engineered input features that can not be learned in a single layer. Aside from MBL being too complex, another possible cause for the inability to identify a phase transition is that the selected system size might be too small. A previous study of MBL [37] indicates that the phase transition becomes more sharply defined as the system size increases.

The main questions that remain are whether FFA might be useful on models that are less complex than MBL, but nonetheless can't be studied using PCA; and whether the limitations on the features that FFA is able to learn can be overcome.

8 Conclusion

It has been demonstrated that like backpropagation, the Forward-Forward algorithm (FFA) can be used as a training algorithm in neural network approaches to the study of phase transitions in physical systems. A supervised classifying method has been shown to work to a limited degree of success. An unsupervised discriminative method has also been shown to work, showing promising results on the Ising, q-state Potts and transverse-field Ising models. Results on the MBL model datasets demonstrate some limitations of FFA, which are inherent to the per-layer nature of its objective function.

Along the way, there have been some auxiliary results. FFA has been implemented in Keras/Tensorflow, and a new peer normalization threshold has been shown to improve its performance. A bug in the original implementation of the pairwise error metric has been discovered and corrected, as discussed in appendix B. Finally, an efficient implementation of the Wolff Monte Carlo cluster algorithm has been produced, allowing the creation of a large dataset of samples from the Ising and Potts models in mere minutes on a moderate laptop.

8.1 Comparison of the discriminative method to related work

The proposed discriminative method is similar to techniques such as PCA or variational autoencoders (VAE) in the sense that it is an unsupervised way to learn features of its input. However, it is very different in the sense that the phases are linearly separable in the learned feature space. In a paper about unsupervised learning of phases using PCA [11], it is noted that the phases of the Ising and COP Ising model can be separated by some nonlinear transformation of one or more of the leading principal components. Similarly, in a paper about unsupervised learning of phases using VAE [12], either the absolute value or L^2-norm of the latent parameters are used to separate the phases of the Ising and XY models. In both papers, the exact transformation used depends on the physical model. This is in contrast to the learned order parameter obtained using the discriminative method, which is a linear transformation of the learned features that is automatically obtained using the same procedure for each model. Moreover, the class-based and similarity-based methods also automatically estimate the critical point using the exact same procedure for each model. This is possible because - unlike PCA or VAE - the discriminative learning procedure includes information about the tuning parameter, allowing the trained network to recognize indicators of different regimes of tuning parameter, i.e. indicators of different phases.

It has to be noted that the order parameter method does require some examples of 'maximally ordered' and 'maximally disordered' samples to compute the order operator (40). However, these may also be obtained without prior knowledge by assuming the order to be maximized and minimized at respectively the lowest and highest values of tuning parameter. In any case, computing an order operator is only a way of extracting phase indicator features that the network has already learned.

The inclusion of the tuning parameter in the training procedure makes the discriminative method conceptually similar to prediction-based methods [19, 20]. In both approaches a neural network is trained to learn a relation between input samples and the corresponding tuning parameter, allowing the identification of a critical point without prior knowledge in a model-agnostic manner. Looking at the big picture, it seems clear that in general, learning a relation between the state of a physical system and its tuning parameter(s) is a very promising area of research.

9 Future work

There are two main directions that further research can take. On one hand, there are opportunities for more investigation into the application of machine learning to phase transitions, both using FFA and other learning techniques. On the other hand, FFA itself is relatively new, and can be developed much further.

9.1 Studying phase transitions through machine learning

9.1.1 The discriminative method

A potential area of future research is further development of the proposed discriminative method. So far, the method has only successfully been applied to systems with a ferromagnetic phase. It would be interesting to look at completely different systems, preferably ones that have a well known phase transition that cannot be detected using PCA, to find out if and what kind of challenges arise. Testing on larger MBL chains can determine whether the FF network in section 7.3.4 was mainly limited by its capabilities as suspected, or by the length of the chain.

While the learned order parameter method generally finds the critical point most accurately of all methods tested, it has the downside that it requires some prior assumptions about the physical system. Appendix D explores the viability of simply using PCA to find an axis in feature space that separates the phases, instead of computing an order operator (40). The results using a simplified architecture with one output layer of only five units are promising, and could be an interesting area of further research.

Another avenue for further research is the application of the discriminative method to experimental data, or data with simulated noise or other limitations. It is currently unknown how well this method handles imperfect data, and how large a dataset needs to be for the method to work. Aside from noise in the samples, in reality the tuning parameter itself can often not be controlled with perfect accuracy either, so the susceptibility of the method to noise in the tuning parameter is a potential area for future research too.

Finally, while heuristics have been developed to indicate where a phase transition would be if there was one, it would be helpful to also have some way of telling whether there is a phase transition (or at least some division into classes) in the first place, and if so, how many there are. Clustering methods might help with this, but special care should be taken to ensure that the detected clusters form contiguous regions in tuning parameter space.

9.1.2 Other methods

The discriminative method proposed in this thesis is one unsupervised way to learn features from a dataset of samples, but there are other ways. Some research has already been done in the direction of (variational) auto-encoders [12, 13, 16], but unlike the discriminative method, these approaches do not learn a relation between the samples and tuning parameter. The hidden representation of a predictive model, as suggested by ref. [19], does have this property. A third option would be to combine the two approaches: train an encoder-decoder model, where the input to the encoder is a sample from the system under study, and the decoder has to reconstruct not only the original sample, but also the corresponding tuning parameter(s). Some weighting of the reconstruction loss might be necessary to control how much influence the two objectives have relative to each other. The model would presumably be able to more efficiently encode and reconstruct the sample and tuning parameter if it learns about the underlying physical relation between the two. The advantage of such an approach over the FFA-based discriminative method is that by using backpropagation, it may be possible to learn features that can't be learned with FFA. For example, features that take multiple layers to compute and whose intermediate products are by themselves not useful contributions towards the per-layer objective of FFA.

One disadvantage that auto-encoders will always have compared to discriminative and predictive methods is that, by design, they are incapable of ignoring irrelevant parts of the input. This means that all input should be relevant in order to learn the most useful features, which may be difficult in cases where it's unknown which parts of the available data might be important. Such challenges may be overcome by first training a discriminative or predictive model to find out which inputs the learned features depend on most strongly, and are therefore most relevant. An auto-encoder could subsequently be trained on only the relevant parts of the data.

9.2 The Forward-Forward algorithm

As stated by Geoffrey Hinton in his FFA paper [1], there are many directions for further development of FFA. This includes testing different activation functions, as well as different definitions of goodness. In particular, Hinton mentions that the activities could be minimized in the positive pass and maximized in the negative pass. Another option is to have a combination of units that maximize and units that minimize during the positive pass. Units that minimize their activity in the positive pass are essentially supposed to find features of their input data that sum to zero for positive data, but not for negative data. In other words, having minimizing units allows the network to learn constraints that apply to the data, which might be useful for applications in physics.

Besides what has already been stated in the FFA paper, it would be interesting to see the effect of different learning rate decay schedules on the performance of FFA, since learning rate decay has been shown to have a great impact on networks trained with backpropagation [50, 51]. It would also be worth investigating the idea of a per-layer learning rate, since figure 27 suggests that at least on MNIST, different layers perform best at different values of the learning rate.

Finally, it might be possible to develop a version of FFA that is able to learn more complex features than those that can be learned in a single layer. For example, the FFA objective could be applied to stacked modules of two or more layers. The number of layers per module should be kept at a minimum to avoid the potential problem of hiding important learned features. Another option would be to enable individual neural units to learn more complex functions of their inputs (compared to eq. (1), that is).

10 Acknowledgements

I would like to thank Evert van Nieuwenburg for guidance and feedback. Furthermore, I want to thank Felix Frohnert for feedback on the thesis as a whole, as well as Erik Baalbergen for feedback on section 4. Finally, I want to thank Tjeerd & Finn van Cappelle for proofreading.

This work was performed using the compute resources from the Academic Leiden Interdisciplinary Cluster Environment (ALICE) provided by Leiden University.

References

[1] G. Hinton, "The forward-forward algorithm: Some preliminary investigations," 2022.

[2] J. Carrasquilla and R. G. Melko, "Machine learning phases of matter," *Nature Physics*, vol. 13, pp. 431–434, May 2017.

[3] K. Ch'ng, J. Carrasquilla, R. G. Melko, and E. Khatami, "Machine learning phases of strongly correlated fermions," *Phys. Rev. X*, vol. 7, p. 031038, Aug 2017.

[4] X. Chen, F. Liu, S. Chen, J. Shen, W. Deng, G. Papp, W. Li, and C. Yang, "Study of phase transition of potts model with domain adversarial neural network," *Physica A: Statistical Mechanics and its Applications*, vol. 617, p. 128666, may 2023.

[5] N. Maskara, M. Buchhold, M. Endres, and E. van Nieuwenburg, "Learning algorithm reflecting universal scaling behavior near phase transitions," *Physical Review Research*, vol. 4, may 2022.

[6] E. van Nieuwenburg, E. Bairey, and G. Refael, "Learning phase transitions from dynamics," *Phys. Rev. B*, vol. 98, p. 060301, Aug 2018.

[7] F. Schindler, N. Regnault, and T. Neupert, "Probing many-body localization with neural networks," *Phys. Rev. B*, vol. 95, p. 245134, Jun 2017.

[8] F. Kotthoff, F. Pollmann, and G. De Tomasi, "Distinguishing an anderson insulator from a many-body localized phase through space-time snapshots with neural networks," *Phys. Rev. B*, vol. 104, p. 224307, Dec 2021.

[9] W. Zhang, L. Wang, and Z. Wang, "Interpretable machine learning study of the many-body localization transition in disordered quantum ising spin chains," *Phys. Rev. B*, vol. 99, p. 054208, Feb 2019.

[10] R. M. Woloshyn, "Learning phase transitions: comparing pca and svm," 2019.

[11] L. Wang, "Discovering phase transitions with unsupervised learning," *Physical Review B*, vol. 94, nov 2016.

[12] S. J. Wetzel, "Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders," *Physical Review E*, vol. 96, aug 2017.

[13] W. Hu, R. R. P. Singh, and R. T. Scalettar, "Discovering phases, phase transitions, and crossovers through unsupervised machine learning: A critical examination," *Physical Review E*, vol. 95, jun 2017.

[14] A. Tirelli, D. O. Carvalho, L. A. Oliveira, J. P. de Lima, N. C. Costa, and R. R. dos Santos, "Unsupervised machine learning approaches to the q-state potts model," *The European Physical Journal B*, vol. 95, nov 2022.

[15] A. Lidiak and Z. Gong, "Unsupervised machine learning of quantum phase transitions using diffusion maps," *Phys. Rev. Lett.*, vol. 125, p. 225701, Nov 2020.

[16] C. Alexandrou, A. Athenodorou, C. Chrysostomou, and S. Paul, "The critical temperature of the 2d-ising model through deep learning autoencoders," *The European Physical Journal B*, vol. 93, dec 2020.

[17] E. P. L. van Nieuwenburg, Y.-H. Liu, and S. D. Huber, "Learning phase transitions by confusion," *Nature Physics*, vol. 13, pp. 435–439, feb 2017.

[18] Y.-H. Liu and E. P. L. van Nieuwenburg, "Discriminative cooperative networks for detecting phase transitions," *Phys. Rev. Lett.*, vol. 120, p. 176401, Apr 2018.

[19] F. Schäfer and N. Lörch, "Vector field divergence of predictive model output as indication of phase transitions," *Phys. Rev. E*, vol. 99, p. 062107, Jun 2019.

[20] E. Greplova, A. Valenti, G. Boschung, F. Schäfer, N. Lörch, and S. D. Huber, "Unsupervised identification of topological phase transitions using predictive models," *New Journal of Physics*, vol. 22, p. 045003, apr 2020.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct 1986.

[22] F. Crick, "The recent excitement about neural networks," *Nature*, vol. 337, pp. 129–132, Jan 1989.

[23] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, pp. 335–346, Jun 2020.

[24] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, 11–13 Apr 2011.

[25] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems* (J. Moody, S. Hanson, and R. Lippmann, eds.), vol. 4, Morgan-Kaufmann, 1991.

[26] G. Hinton, "Official matlab implementation of the forward-forward algorithm." https://www.cs.toronto.edu/~hinton/ffcode.zip, 2023. Accessed: 2023-02-21.

[27] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[28] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," 2016.

[29] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[30] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012.

[31] L. Lu, “Dying ReLU and initialization: Theory and numerical examples,” *Communications in Computational Physics*, vol. 28, pp. 1671–1706, jun 2020.

[32] W. Lenz, “Beiträge zum verständnis der magnetischen eigenschaften in festen körpern,” *Physikalische Zeitschrift*, vol. 21, pp. 613–615, 1920.

[33] L. Onsager, “Crystal statistics. i. a two-dimensional model with an order-disorder transition,” *Physical review*, vol. 65, no. 3-4, pp. 117–149, 1944.

[34] R. B. Potts, “Some generalized order-disorder transformations,” *Mathematical proceedings of the Cambridge Philosophical Society*, vol. 48, no. 1, pp. 106–109, 1952.

[35] P. Pfeuty, “The one-dimensional ising model with a transverse field,” *Annals of Physics*, vol. 57, no. 1, pp. 79–90, 1970.

[36] J. Richter, T. Heitmann, and R. Steinigeweg, “Quantum quench dynamics in the transverse-field Ising model: A numerical expansion in linked rectangular clusters,” *SciPost Phys.*, vol. 9, p. 031, 2020.

[37] A. Pal and D. A. Huse, “Many-body localization phase transition,” *Phys. Rev. B*, vol. 82, p. 174411, Nov 2010.

[38] D. J. Luitz, N. Laflorencie, and F. Alet, “Many-body localization edge in the random-field heisenberg chain,” *Phys. Rev. B*, vol. 91, p. 081103, Feb 2015.

[39] T. Chanda, P. Sierant, and J. Zakrzewski, “Many-body localization transition in large quantum spin chains: The mobility edge,” *Phys. Rev. Res.*, vol. 2, p. 032045, Aug 2020.

[40] U. Wolff, “Collective monte carlo updating for spin systems,” *Phys. Rev. Lett.*, vol. 62, pp. 361–364, Jan 1989.

[41] R. H. Swendsen and J.-S. Wang, “Nonuniversal critical dynamics in monte carlo simulations,” *Phys. Rev. Lett.*, vol. 58, pp. 86–88, Jan 1987.

[42] E. Luijten, *Introduction to Cluster Monte Carlo Algorithms*, pp. 13–38. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[43] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” *The Journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[44] W. K. Hastings, “Monte carlo sampling methods using markov chains and their applications,” *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[45] E. van Nieuwenburg and M. Schmitt, “Crc183 github repository, files: `notebooks/isingdata.py`, `notebooks/wolffalgorithm.py`.” `https://github.com/CRC183-summer-school/school_2021`, 2021.

[46] F. Vicentini, D. Hofmann, A. Szabó, D. Wu, C. Roth, C. Giuliani, G. Pescia, J. Nys, V. Vargas-Calderón, N. Astrakhantsev, and G. Carleo, "NetKet 3: Machine Learning Toolbox for Many-Body Quantum Systems," *SciPost Phys. Codebases*, p. 7, 2022.

[47] F. Vicentini, "Ground-state: Ising model." https://netket.readthedocs.io/en/latest/tutorials/gs-ising.html, 2021.

[48] M. E. Fisher and M. N. Barber, "Scaling theory for finite-size effects in the critical region," *Phys. Rev. Lett.*, vol. 28, pp. 1516–1519, Jun 1972.

[49] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse domains through world models," 2023.

[50] K. You, M. Long, J. Wang, and M. I. Jordan, "How does learning rate decay help modern neural networks?," 2019.

[51] A. Lewkowycz, "How to decay your learning rate," 2021.

A Defining the objective function

A definition of the objective function C_i is not given in either the original paper [1] or the MATLAB code [26]. This is most likely because it's not very important which exact objective function is used, as long as high activities are rewarded in the positive pass and punished in the negative pass. Nonetheless, C_i can be found by integrating its gradient (4). The peer normalization gradient is neglected in this process, since it is not part of the main objective function.

Let us start with the gradient for an arbitrary activity vector $\mathbf{x}$ of layer i in the negative pass, $\nabla_{\mathbf{x}} C_i^-(\mathbf{x})$:

$$\nabla_{\mathbf{x}} C_i^-(\mathbf{x}) = -p_i \mathbf{x} = -\sigma\left(\|\mathbf{x}\|^2 - n\right)\mathbf{x} \tag{44}$$

We can rewrite this using $\mathbf{x} = r\hat{\mathbf{x}}$, with $\hat{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$ the unit vector of $\mathbf{x}$:

$$\nabla_{\mathbf{x}} C_i^-(\mathbf{x}) = -\sigma\left(r^2 - n_i\right) r\hat{\mathbf{x}} = \frac{-r\hat{\mathbf{x}}}{1 + e^{n_i - r^2}} \tag{45}$$

This allows us to define the objective C_i^- as a line integral from the origin to $\mathbf{a}_i$:

$$\begin{aligned} C_i^-(\mathbf{a}_i) &= C_i^-(\mathbf{0}) + \int_{\mathbf{0}}^{\mathbf{a}_i} \nabla_{\mathbf{x}} C_i^-(\mathbf{x}) \cdot d\mathbf{x} \\ &= C_i^-(\mathbf{0}) + \int_0^{\|\mathbf{a}_i\|} \frac{-r\hat{\mathbf{x}}}{1 + e^{n_i - r^2}} \cdot \hat{\mathbf{x}} dr \\ &= C_i^-(\mathbf{0}) + \int_0^{\|\mathbf{a}_i\|} \frac{-r}{1 + e^{n_i - r^2}} dr \end{aligned} \tag{46}$$

The integral can be simplified by introducing $u = n_i - r^2$:

$$\begin{aligned} \int \frac{-r}{1 + e^{n_i - r^2}} dr &= \frac{1}{2}\int \frac{1}{1+e^u} du \\ &= \frac{1}{2}\int \left(1 - \frac{e^u}{1+e^u}\right) du \\ &= \frac{u}{2} - \frac{1}{2}\int \frac{e^u}{1+e^u} du \end{aligned} \tag{47}$$

We can then introduce $s = 1 + e^u$:

$$\begin{aligned} \frac{u}{2} - \frac{1}{2}\int \frac{e^u}{1+e^u} du &= \frac{u}{2} - \frac{1}{2}\int \frac{1}{s} ds \\ &= \frac{u}{2} - \frac{\ln(s)}{2} \\ &= \frac{1}{2}\left(n_i - r^2 - \ln(1 + e^{n_i - r^2})\right) \end{aligned} \tag{48}$$

Now $C_i^-(\mathbf{a}_i)$ can be defined:

$$\begin{aligned} C_i^-(\mathbf{a}_i) &= C_i^-(\mathbf{0}) + \int_0^{\|\mathbf{a}_i\|} \frac{-r}{1 + e^{n_i - r^2}} dr \\ &= C_i^-(\mathbf{0}) + \frac{1}{2}\left(-\|\mathbf{a}_i\|^2 - \ln(1 + e^{n_i - \|\mathbf{a}_i\|^2}) + \ln(1 + e^{n_i})\right) \\ &= C_i^-(\mathbf{0}) + \frac{1}{2}\left(-g_i - \ln(1 + e^{n_i - g_i}) + \ln(1 + e^{n_i})\right) \end{aligned} \tag{49}$$

with $g_i = \|\mathbf{a}_i\|^2$ being the goodness. We are free to choose $C_i^-(\mathbf{0})$ such that we get rid of the constant, leaving us with:

$$C_i^-(\mathbf{a}_i) = -\frac{1}{2}\left(g_i + \ln(1 + e^{n_i - g_i})\right) \tag{50}$$

The gradient in the positive pass can be defined partially in terms of the gradient in the negative pass:

$$\nabla_{\mathbf{x}} C_i^+(\mathbf{x}) = (1 - p_i)\mathbf{x} = \mathbf{x} + \nabla_{\mathbf{x}} C_i^-(\mathbf{x}) \tag{51}$$

which yields the following integral:

$$\begin{aligned} C_i^+(\mathbf{a}_i) &= C_i^+(\mathbf{0}) + \int_{\mathbf{0}}^{\mathbf{a}_i} \left(\mathbf{x} + \nabla_{\mathbf{x}} C_i^-(\mathbf{x})\right) \cdot d\mathbf{x} \\ &= C_i^+(\mathbf{0}) + \frac{1}{2}\|\mathbf{a}_i\|^2 + \int_{\mathbf{0}}^{\mathbf{a}_i} \nabla_{\mathbf{x}} C_i^-(\mathbf{x}) \cdot d\mathbf{x} \\ &= C_i^+(\mathbf{0}) + \frac{g_i}{2} + C_i^-(\mathbf{a}_i) - C_i^-(\mathbf{0}) \end{aligned} \tag{52}$$

allowing us to define C_i^+ as:

$$C_i^+(\mathbf{a}_i) = -\frac{1}{2}\ln(1 + e^{n_i - g_i}) \tag{53}$$

We can now make some approximations for large layers with $n_i \gg 1$, using the shorthand $C_i^- = C_i^-(\mathbf{a}_i)$ and $C_i^+ = C_i^+(\mathbf{a}_i)$:

$$\begin{aligned} g_i \gg n_i \Rightarrow & \ln(1 + e^{n_i - g_i}) \approx 0, \\ & C_i^- \approx -\frac{g_i}{2}, \\ & C_i^+ \approx 0 \\ g_i \ll n_i \Rightarrow & \ln(1 + e^{n_i - g_i}) \approx n_i - g_i \\ & C_i^- \approx -\frac{n_i}{2} \\ & C_i^+ \approx \frac{g_i - n_i}{2} \end{aligned} \tag{54}$$

Indeed, the objective function for the negative pass is maximized at $g_i \ll n_i$, while the objective for the positive pass is maximized at $g_i \gg n_i$. Note also that the objective functions are approximately constant in their respective maximized domains, and sloped w.r.t. g_i everywhere else.

B FFA implementation and other code

Code for the implementation of FFA; the generation of the data; as well as the code used to detect phase transitions in the various models, is available on GitHub: https://github.com/JellevanCappelle/Detecting-phases-using-FFA. The FFA implementation is discussed further in the following subsections.

B.1 The `FFLayer` and `FFModel` classes

The Forward-Forward algorithm is implemented in two classes: `FFLayer` represents a single layer, whereas `FFModel` overrides the necessary functions of the `tf.keras.Model` class to implement the training procedure.

`FFLayer` overrides the `call` function of the base class (`tf.keras.layers.Layer`) to propagate inputs. It also implements a function to propagate inputs while calculating gradients and metrics (`calc_state_and_gradients`) and one to propagate inputs while only calculating metrics (`calc_pairwise_err`), used in testing. Equations (1 – 6) are all implemented in (functions called by) `calc_state_and_gradients`.

`FFModel` overrides the `call` and `train_step` functions of the base class (`tf.keras.Model`) to implement the feed-forward operation, and the training procedure, respectively. This class also implements an `update_metrics` function which is used in the `test_step` of the `FFClassifier` and `FFDiscriminator` classes.

It is important to note that the pairwise error metric is always calculated by comparing the goodness in the positive and negative passes, rather than by comparing the predicted probabilities, which is done in the original MATLAB code [26]. The reason for this choice is that two different values of goodness can lead to the same 32-bit float result for the probability, because large positive/negative values of goodness yield probabilities that are almost exactly one/zero. This leads to many spurious equalities, which can either inflate or deflate the number of pairwise errors, depending on whether equalities are considered errors. This code counts equalities as errors, but since the goodness values are compared directly, equalities virtually never occur in practice. The difference between comparing goodness values and probabilities can be quite large. For example, the MATLAB code - which doesn't count equalities as errors - incorrectly reports pairwise errors as low as 1.2% for the first layer in the fifth epoch, while the custom implementation reports 4.2% in the same epoch for the same experiment. Note that analytically, the two ways of computing the pairwise error are equivalent, since the predicted probability is a strictly increasing function of the goodness, see equation (3). The difference between counting equalities as errors or not should be negligible, if no rounding occurs. Therefore, the difference in reported pairwise errors can only be due to the rounding of 32-bit floats.

Another difference from the MATLAB implementation is that the SGD optimizer that comes with Keras scales its learning rate differently compared to the update rules in equation (8). The MATLAB code and equation (8) apply the following update rule to momentum variables Δ and weights W:

$$\begin{aligned} \Delta' &= \beta\Delta + (1-\beta)\nabla \\ W' &= W + \alpha\Delta \end{aligned} \tag{55}$$

with α the learning rate, β the momentum parameter and ∇ the gradient. The Keras optimizer doesn't scale the gradients by $1 - \beta$, which results in the following update rule for Δ:

$$\Delta' = \beta\Delta + \nabla \tag{56}$$

which effectively scales all momentum variables and updates by a factor of $\frac{1}{1-\beta}$. In all experiments the learning rate is scaled by $1 - \beta$ to compensate for this fact. There is one aspect in which this still makes a difference, however. In the training procedure, the weight decay parameter is multiplied by the learning rate, which in Keras has to be obtained from the optimizer. Therefore, the weight decay values used in this thesis are not directly comparable to those in the MATLAB implementation. The values discussed in this thesis should be multiplied by $1 - \beta$ (usually 0.1 if $\beta = 0.9$) to obtain their equivalent value in the MATLAB implementation.

B.2 The `FFClassifier` class

The `FFClassifier` class implements a classifier model as described in section 3 in a generalized way, such that it can be used with any labelled dataset. It overrides the `call`, `train_step` and `test_step` functions. It generates negative data by attaching incorrect labels to input samples, and internally uses `FFModel` to implement the training procedure.

B.3 The `FFDiscriminator` class

The `FFDiscriminator` class implements the discriminative training procedure discussed in section 7. Analogous to `FFClassifier`, it generates negative data by attaching false labels to input samples, and uses `FFModel` to implement the rest of the training procedure.

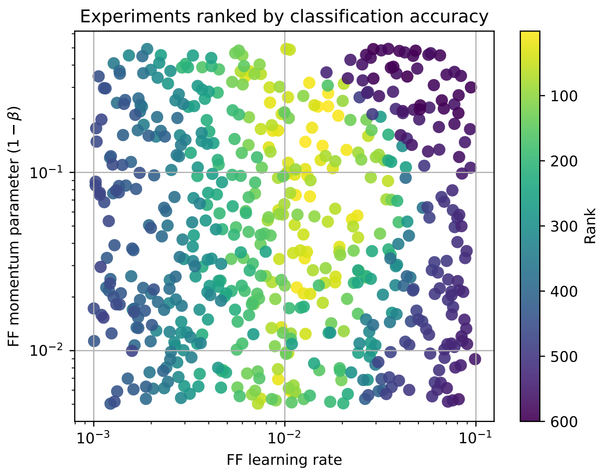
C Hyperparameter study on MNIST

Three separate hyperparameter searches have been conducted on the MNIST experiment, to develop an understanding of the effects of the different available hyperparameters. The following searches have been conducted, using the defaults listed in section 3.1.1:

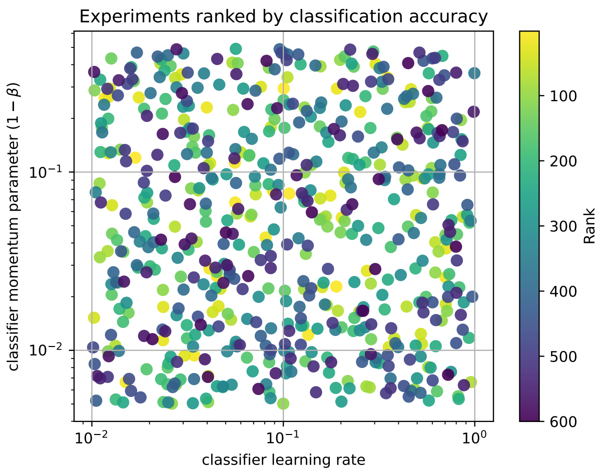
- A search of 600 runs over the learning rates and momentum parameters of both the FF layers and the classification layer. Results are presented in 3.1.2 and continued in C.1.
- A search of 600 runs over the peer normalization weight and delay parameters. Results are presented in C.2.
- A search of 600 runs over the weight decay parameters of both the FF layers and the classification layer. Results are presented in C.3.

C.1 Learning rate and momentum parameter, continued

A hyperparameter search was conducted over the learning rates and momentum parameters of both the FF layers and the classification layer. This choice was made to be able to visualize the relative impact of all four parameters. Figure 26 ranks the results by classification accuracy. It is clear that while the accuracy strongly depends on the parameters of the FF layers, the parameters of the classification layer have an unnoticeable impact by comparison. Figure 27 plots the best and worst pairwise errors in all layers, w.r.t. the FF layer parameters. Note that different layers seem to prefer different parameters, especially different learning rates. This suggest that FFA might benefit from a per-layer learning rate.

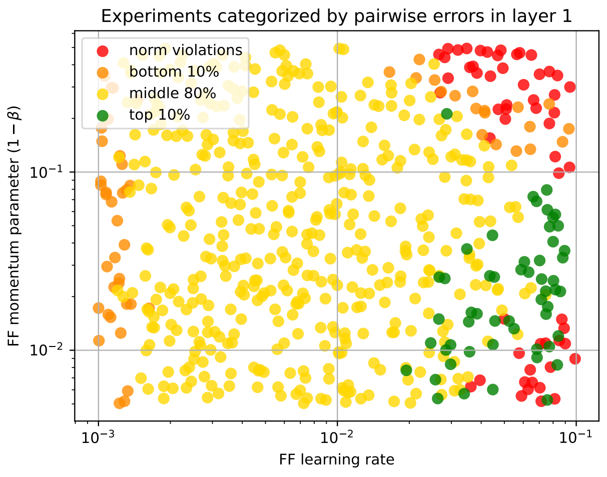


(a) Plotted w.r.t. FF layer parameters.

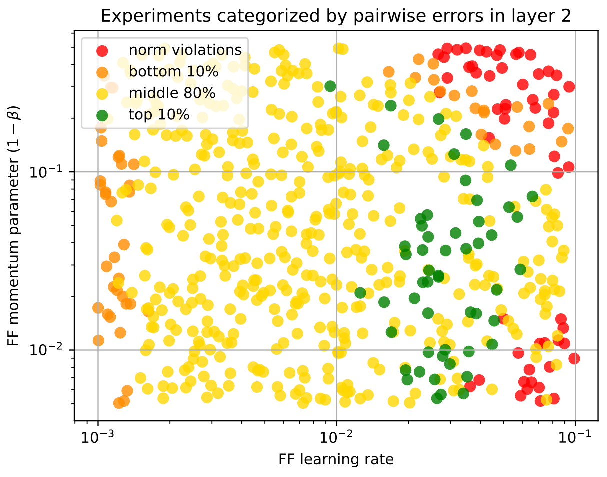


(b) Plotted w.r.t. classification layer parameters.

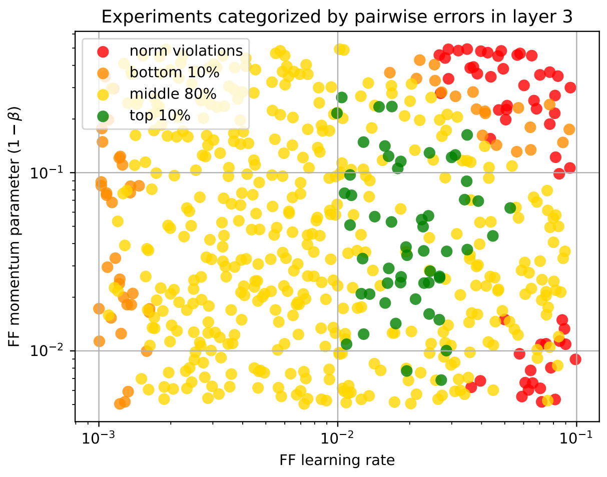
Figure 26: Results from a random hyperparameter search over the learning rates and momentum parameters of the Forward-Forward network and classification layer. The runs are ranked from lowest classification error (yellow) to highest error (dark purple). The Y-axes represent $1-\beta$, with β the momentum parameter, to facilitate log-scaling. Metrics are taken from the validation step of the final epoch.



(a) Layer 1.



(b) Layer 2.

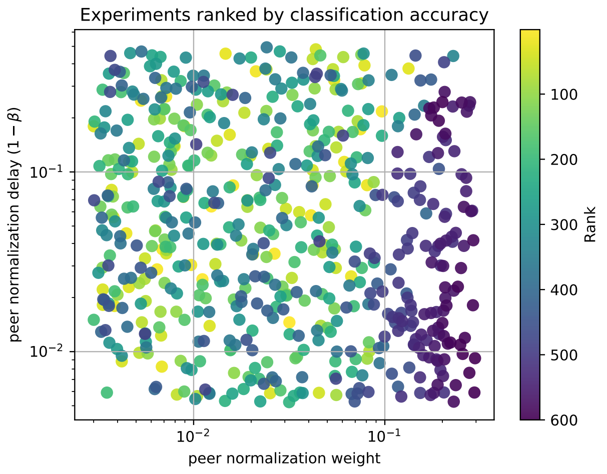


(c) Layer 3.

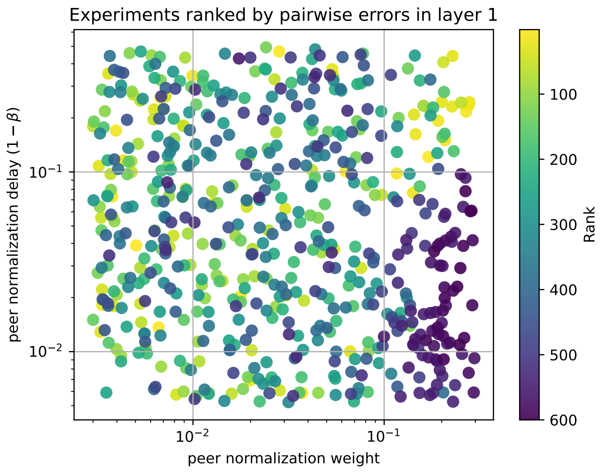
Figure 27: Top 10% and bottom 10% categories of pairwise error, as well as a norm-violation category. 'Norm-violations' indicate cases where the mean of squares of the FF network output was not equal to one.

C.2 Peer normalization

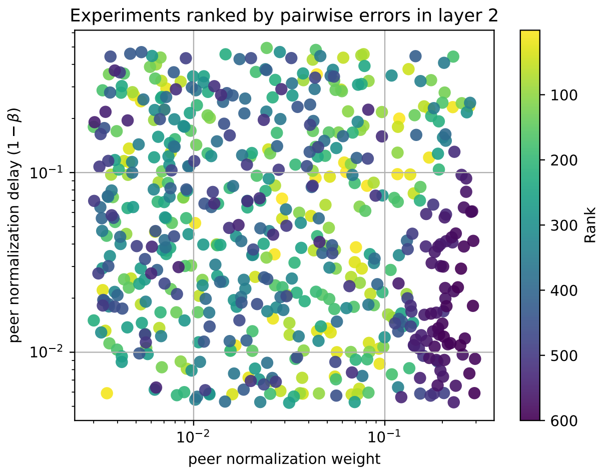
A hyperparameter search was conducted over the peer normalization weight and delay parameters. Figure 28 ranks the results by classification accuracy and pairwise errors for each layer. Something that stands out, is that too much peer normalization is detrimental to performance, especially to the classification accuracy. This makes sense: the stronger the peer normalization, the less freedom the network has to learn features that are more or less active than the average of their peers. This can ultimately result in worse performance. It also seems slightly better to choose a peer normalization delay of $\beta \leq 0.9$, at least considering the pairwise errors in the first two layers.



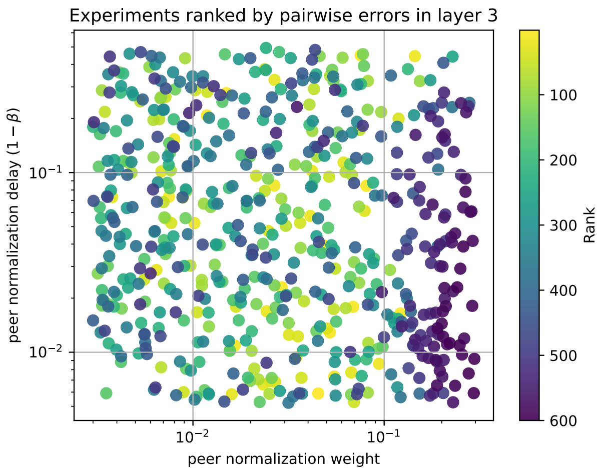
(a) Ranked by classification error.



(b) Ranked by pairwise errors in layer 1.



(c) Ranked by pairwise errors in layer 2.

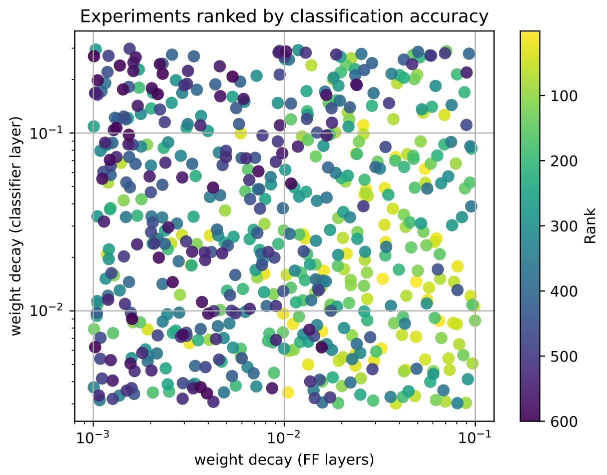


(d) Ranked by pairwise errors in layer 3.

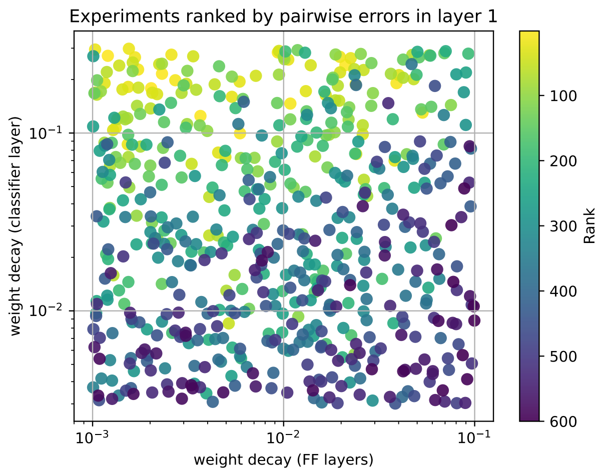
Figure 28: Results from a random hyperparameter search over the peer normalization parameters of the Forward-Forward network. The runs are ranked by (a) classifier validation error; (b, c, d) pairwise validation errors for layers 1, 2, and 3 respectively. Runs are ranked from lowest error (yellow) to highest error (dark purple).

C.3 Weight decay

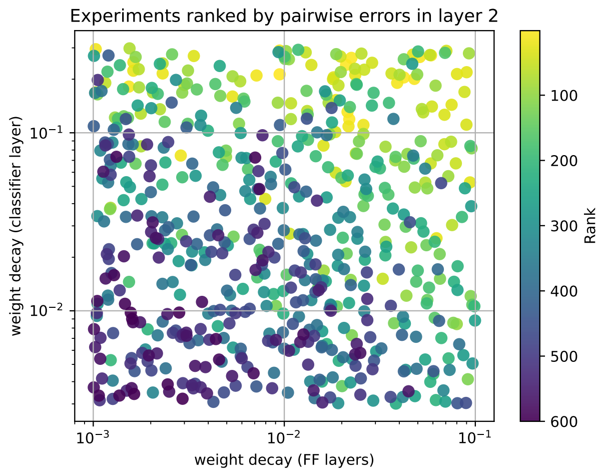
A hyperparameter search was conducted over the weight decay parameters of the FF layers and classification layer. Figure 29 plots a ranking of the results by classification accuracy and pairwise errors for each layer. The weight decay of the FF layers seems to be important for getting a good classification accuracy, with higher weight decay yielding better results. A weight decay above 0.1 in the classification layer seems to lead to slightly worse classification accuracy. However, the pairwise error in the first layer actually benefits from having a high weight decay in the classification layer. In fact, the rankings for the pairwise error in the first layer and the classification accuracy are almost polar opposites. This is most likely due to the fact that as the classification improves, the task of discriminating true labels from false labels becomes increasingly difficult, leading to a worse pairwise error. The first layer therefore benefits from worse classification accuracy when the weight decay in the classification layer is very high. Meanwhile, last two layers seem to benefit from an increased weight decay in the FF layers, in terms of pairwise errors.



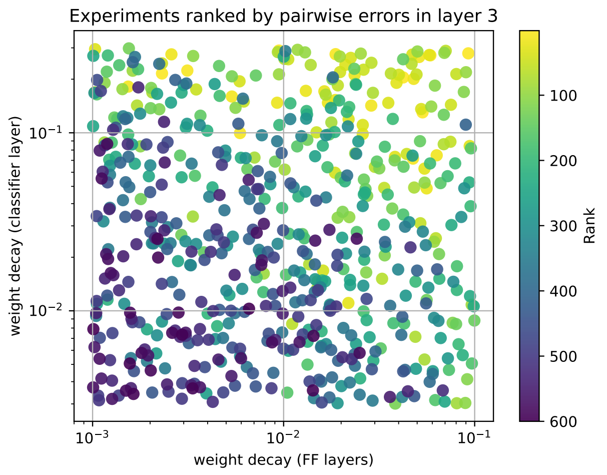
(a) Ranked by classification error.



(b) Ranked by pairwise errors in layer 1.



(c) Ranked by pairwise errors in layer 2.



(d) Ranked by pairwise errors in layer 3.

Figure 29: Results from a random hyperparameter search over the weight decay parameters of the Forward-Forward network and classification layer. The runs are ranked by (a) classifier validation error; (b, c, d) pairwise validation errors for layers 1, 2, and 3 respectively. Runs are ranked from lowest error (yellow) to highest error (dark purple).

D Simplifying the discriminative method

The learned order parameter obtained using the discriminative method shows that for the ferromagnetic models, the phases are linearly separable in the learned feature space. However, obtaining the order operator (40) requires some prior assumptions about the physical model. This appendix explores the viability of a much simpler procedure that separates the phases through PCA of the learned features. The hypothesis is that the phase of the physical system is the most important property for a discriminative network to learn, so therefore it should account for most variance in feature space.

To test this hypothesis, PCA is performed on network outputs of 3,000 samples from the 5-state Potts model, using the same architecture and training procedure that were used in section 7.3.2. The samples are scatter plotted by their two leading principal components in figure 30. In this case, the phases are indeed linearly separable in the first principal component, but there are a total of six clusters. There is one cluster of non-magnetized samples, and one cluster for each direction in which the model can be magnetized. It is not exactly clear why this is the case. The different features could simply be a left-over from the random initialization. After all, with 100 units in each layer, there is no pressure on the network to learn only the most general representation. It could also be the case that the network is fitting to some stochastic biases towards certain spin values for certain temperatures, which are inevitably present in the dataset.

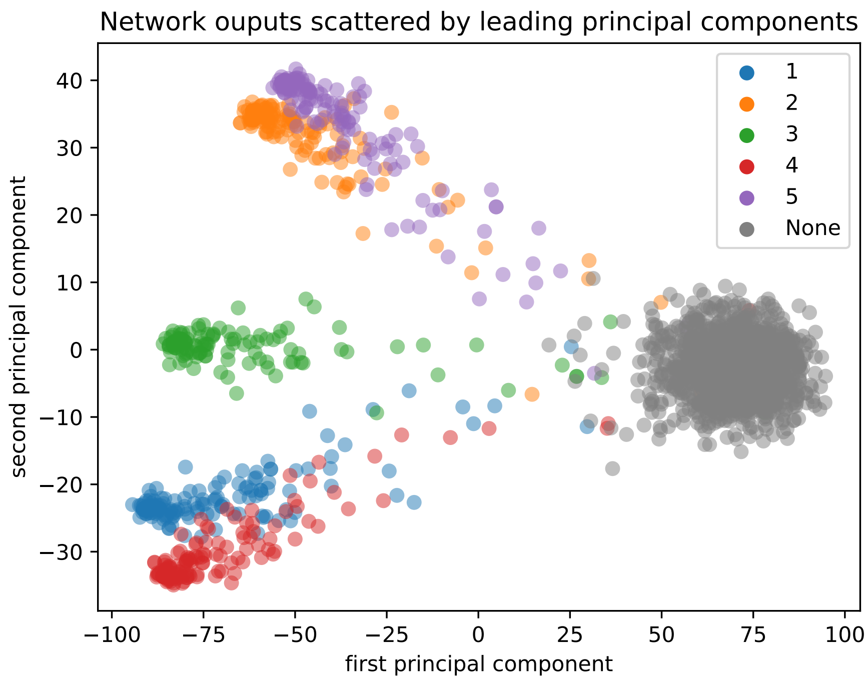


Figure 30: Scatter plots of neural network outputs for the 5-state Potts model. Samples are colored by the dominant spin value, i.e. the spin value that occupies at least half of all sites. The scale of the axes reflect the explained variance of the corresponding principal component. 3,000 randomly selected samples are plotted.

Ideally, the network should not learn different features for samples that belong to the same phase. A good way to achieve this, is to limit the number of units in the final layer such that it can only represent a very limited amount of information about the input sample. This should force the network to learn very general features that are the same regardless of the dominant spin value. To test this, a new simplified network architecture is trained on different models, again using 100 units for the first two layers, but only five for the final layer. The output of the network is just the final layer.

Figure 31 is a repeat of figure 30 using the simplified architecture, showing that this time there are no separate clusters for the different magnetization directions of the Potts model. Outputs of the original and simplified architectures are scatter plotted for all three ferromagnetic models in figure 32. For each model, the original architecture produces a cluster for the disordered phase, as well as one for every direction in which the model can be magnetized. Both leading principal components are significant in terms of explained variance. In fact, both components are required to separate the phases of the Ising model and TFIM, while only those of the 5-state Potts model[3]can be separated using just the first component. The simplified architecture on the other hand only produces one cluster for each phase of each model, and the explained variance of the second component is negligible compared to the first component. Because of this, the phases of each model can be separated using only the first principal component of the simplified network output.

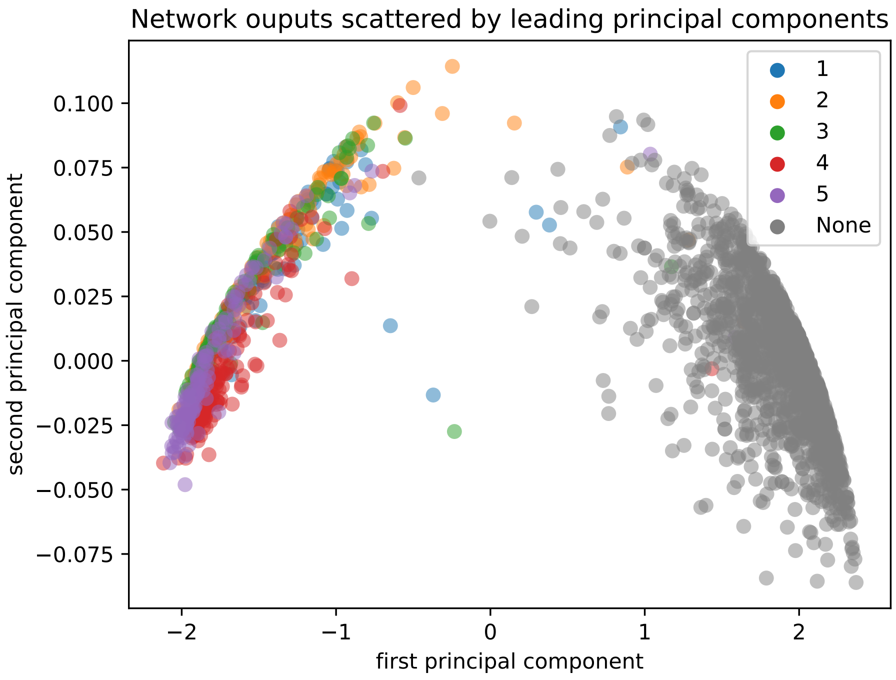


Figure 31: Repeat of figure 30 using the simplified network architecture.

[3]The clusters of the Potts model are in different locations in figures 30 and 31, despite being generated using the same network weights. This is because each random selection of 3,000 samples yields slightly different principal components, meaning a slightly different 2D slice of a 200D feature space.

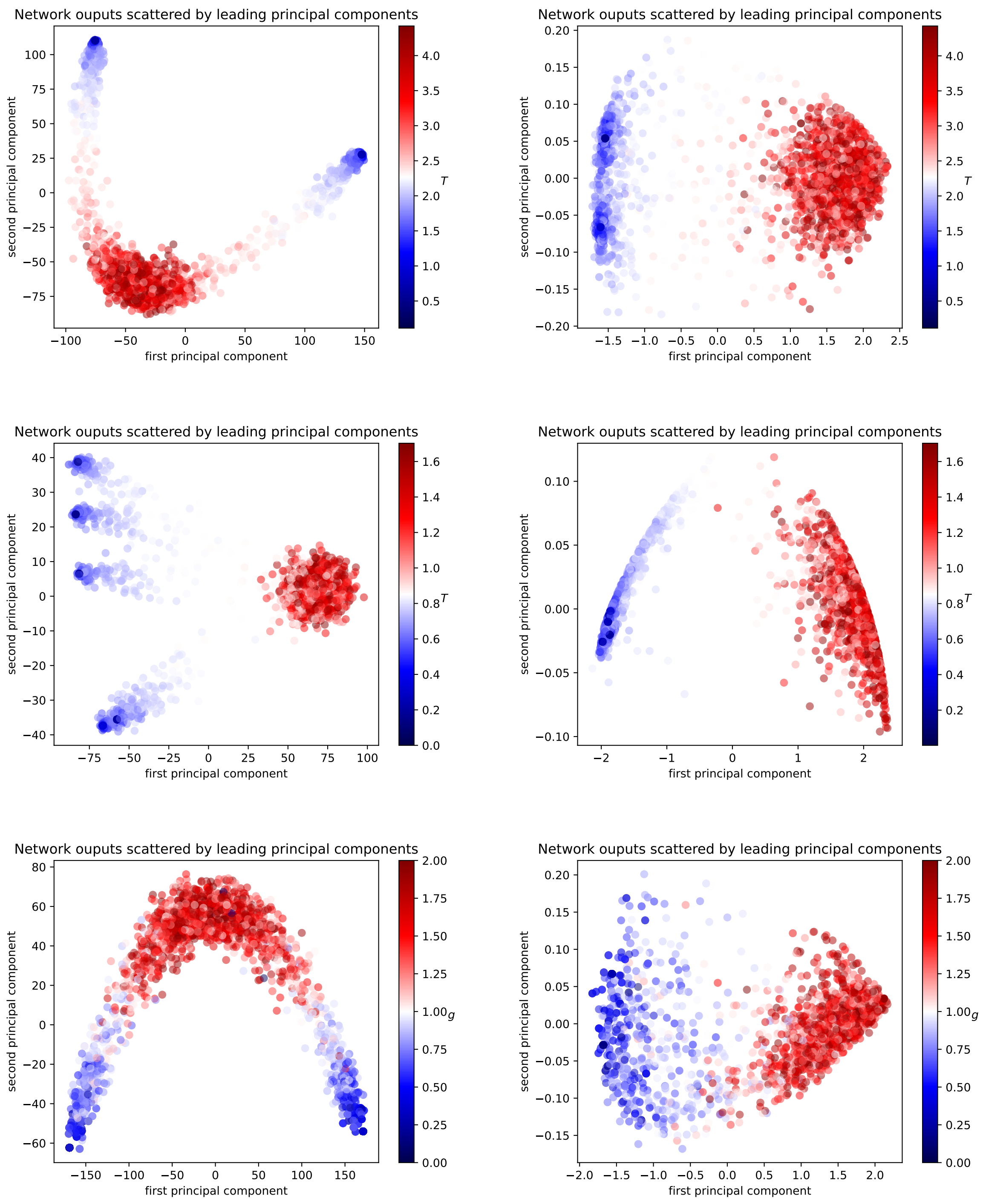


Figure 32: Scatter plots of neural network outputs for different physical models. Left column: using the original architecture of three 100-unit layers, with the final two layers as output. Right column: using the simplified architecture of two 100-unit layers followed by a 5-unit output layer. First row: Ising model with $L = 30$. Second row: 5-state Potts model with $L = 30$. Third row: TFIM with $N = 40$. The scale of the axes reflect the explained variance of the corresponding principal component. 3,000 randomly selected samples are plotted in each figure.