FYSSTK4155

Project 2 - Neural Networks

Erlend Lima, erlenlim

Feed forward multilayered perceptron neural networks are studied by writing an implementation in the Julia language and compared to least squares, Ridge and lasso regression for linear regression, and logistic regression for classification. The data modeled is Ising model in 1D and 2D, and the physical results and choice of hyperparameters are compared to the works of Mehta et al. Our results consistently agree with theirs, with neural nets outperforming standard machine learning methods for classification, but being worse than Ridge and Lasso for regression. As the problems are relatively easy, the optimal results are found using 5 to 10 neurons with ReLU or leaky ReLU and a learning rate $\eta < 0.1$ for classification.

CONTENTS

I. INTRODUCTION

Neural networks have had a renaissance in the recent decades. Their predictive power has fueled progress in a wide variety of fields, from physics to online marketing.
Here we will study neural nets by regression on coupling constants of a 1D Ising model and classifying phases from a 2D Ising model. The results are compared to more standard machine learning techniques, least squares regression, Ridge regression and lasso regression, and logistic regression, respectively.
Comparisons are made to the works of Mehta et al.[3], and much of the data and analyses are inspired by their paper. Besides comparing the results with them, my implementations are checked against `sklearn` and the Julia package `Flux.jl`
The GitHub repository is available here: https://github.com/Caronthir/FYSSTK4155/projects/project2.

II. THEORY

A. The Ising Model

The Ising model is a model for describing binary lattice systems. When cast in the language of physics, the values are regarded as spin, with properties such as energy, magnetization and phase transition being investigated. The phase transition only occur in two or more dimensions. Its analytical description solution was found by Lars Onsager in 1944 [1].
For the one dimensional model the energy is described by

$$E[s^i] = -J\sum_{j=1}^{N} s_j s_{j+1}$$

with J as the coupling constant and s^i as a given spin state.
By lifting the constraint of only summing over nearest neighbors, the model can be recast as a linear regression problem. The energy is now

$$\begin{aligned} E[s^i] &= -\sum_{j,k=1}^{N} J_{j,k} s^i_j s^i_k \\ &= \mathbf{X}^i \cdot \mathbf{J} \end{aligned} \tag{II.1}$$

with $\mathbf{X}^i$ being the matrix over two body interactions and $\mathbf{J}$ the coupling coefficients. The regression can then be performed on the set $\{E[s^i], s^i\}$.
When moving on to two dimensions, the Ising model exhibits phase transitions. Under the critical temperature $T_C/J \approx 2.27$ the system is in an ordered phase with (almost) all spins pointing in the same direction. Above this temperature the system is in an disordered phase with spins randomly oriented. Examples of these are shown in fig. II.1. For practical purposes, ordered phases have $T/J < 2.0$ and disordered $T/J > 2.5$.

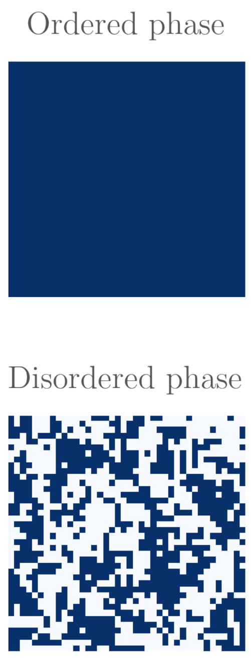

Figure II.1: Examples of ordered and disordered phase systems used in classification.

B. Logistic Regression

In order to classify systems as either ordered or disordered, logistic regression is performed. Classification problems with logistic regression methods have data which is assigned to one of K classes using an indicator response variable $\hat{f}_k = \hat{\beta}_{k0} + \hat{\beta}_k^T x$.
Our case is binary $K = 2$ with

$$y_i \in \{0, 1\} = \{ordered, disordered\}$$

A popular choice for assigned the response to a class is the sigmoid function, giving

$$p(y_i = 0|x_i, \hat{\beta}) = \frac{\exp\left(\hat{\beta}_0 + \hat{\beta}_1 x_i\right)}{1 + \exp\left(\hat{\beta}_0 + \hat{\beta}_1 x_i\right)}$$

$$p(y_i = 1|x_i, \hat{\beta}) = 1 - p(y_i = 0|x_i, \hat{\beta})$$

By minimizing the likelihood we obtain the loss function *cross entropy* [2]

$$\mathcal{C}(\hat{\beta}) = \sum_{i=1}^{n} \left(y_i \log p(y_i = 1|x_i, \hat{\beta}) + (1 - y_i) \log \left[1 - p(y_i = 1|x_i, \hat{\beta}) \right] \right)$$

Classifying binary data is then equivalent to minimizing the cross entropy function. This is achieved using gradient descent.

C. Gradient Descent

Gradient descent is a group of methods to minimize some parameters θ under a function f. The decrease is fastest in the direction of the gradient $-\nabla f(\theta)$. By iteration we then get

$$\theta_{i+1} = \theta_i - \eta \nabla f(\theta_i)$$

where η is the step-size or *learning rate.*
The method has a tendency to slow down when approaching a minima and the gradient gets small. One way to remedy this is to add *momentum*, giving

$$\begin{aligned} v_i &= \mu v_{i-1} + \eta \nabla f(\theta_i) \\ \theta_{i+1} &= \theta_i - v_i \end{aligned}$$

with μ the momentum hyperparameter.
Another benefit is that momentum makes the method more resilient to oscillations.
More can be gained by taking advantage of the fact that the gradient gives more information about a minima the closer it is. By computing the gradient at the *expected* value given the current momentum, we obtain *Nesterov accelerated gradient*:

$$\begin{aligned} v_i &= \mu v_{i-1} + \eta \nabla f(\theta_i + \eta v_{i-1}) \\ \theta_{i+1} &= \theta_i - v_i \end{aligned}$$

This gives faster convergence for a given learning rate.

D. Neural Networks

Neural networks is a machine learning algorithm inspired by biological neurons, and has achieved impressive performance on a wide class of problems. A neuron is modeled by a *perceptron*, with perceptrons stacked together in layers and layers stacked together into a network architecture. The first layer takes an input vector $\mathbf{x}$ of d features and produces an *activation* $a_i(\mathbf{x})$. The activation is feed into the next layer, called *hidden layer*, and so on until the last layer, the *output layer.*

A layer transforms its input but first applying an affine transformation on the form

$$z^{(i)} = \mathbf{W}^{(i)}\mathbf{x} + b^i$$

with $\mathbf{W^i}$ being the *weights* of layer i and b^i its bias. This is followed by a non-linear transformation

$$a_i(\mathbf{x}) = \sigma_i(z^{(i)})$$

The *activation function* σ_i is specific to each layer. The activation function of the output layer determines whether the network is a classifier, in which the function is a sigmoid or softmax function, or regression, where the function are unbounded linear functions.
The combination of linear and non-linear transformations allows a network to approximate any function to arbitrary precision, a result known as the universal approximation theorem[3]. In order to achieve this expressive power, the network has to be trained. This is done by an algorithm known as *backpropagation.*

1. *Backpropagation*

To train a network, a cost function is optimized. For linear regression this is often the mean square error, while for classification this is often the cross entropy. The optimization is done by gradient descent, where an input is fed forward through the network, compared to the expected output, and the error is fed backwards and used to update the weights and biases. This backpropagation of the error is essentially the chain rule from calculus applied recursively.
The four equations of backpropagation are

$$\begin{aligned} \delta_j^l &= \frac{\partial E}{\partial a_j^l} \sigma\prime \left(z_j^l \right) \\ \delta_j^l &= \frac{\partial E}{\partial b_j^l} \\ \delta_j^l &= \left(\sum_k \delta_k^{l+1} w_{kj}^{l+1} \right) \sigma\prime \left(z_j^l \right) \\ \frac{\partial E}{\partial w_{jk}^l} &= \delta_j^l a_k^{l-1} \end{aligned}$$

A deeper explanation of these equation and their derivation can be found in literature, such as [3].

E. Activation Functions

There are a wide arrange of activation functions available, with new ones proposed and their virtues hailed in papers. Here three common activation functions will be studied: the sigmoid, the rectified linear unit (ReLU) and leaky ReLU. These are plotted in fig. II.2.

All are non-linear, as is required for a network to learn; if a network only had a linear transformation, the entire network would be nothing more than linear transformation. The functions are similar in their behavior, with some important differences. The sigmoid suffers from *vanishing derivative*, where the gradient can be vanishingly small and prevent weights from updating. The ReLU rectify this problem as it only saturates in one direction, and the leaky ReLU doesn't have any saturation, and does not suffer from this problem. In addition, ReLU and leaky ReLU have been found to outperform the sigmoid, giving faster learning.[3][2].

III. METHOD

A. Implementation Details

When using linear regression to solve (II.1), methods from `sklearn.linear_models` were used as they are quicker and more versatile than my own code written for the first project.

The logistic regression and neural network were written in Julia. Three versions of gradient descent were written: gradient descent, stochastic gradient descent and Nesterov gradient descent, all with early stopping. All of these were tested, confirmable by the example scripts in the git repository.

Due to my unfamiliarity with the language, I made several critical design flaws. I began writing the code as I would do in C++ or Python, by object orienting it. Julia, however, is not designed for OO-programming, and trying to do so rewards you with inflexible code that is hard to optimize. The type system was abused, so instead of allowing my code to work on many different types, I constantly constrained myself to a small subset.

The result of this mess is that a lot of time was spent hunting down bugs and trying to optimize hard-to-optimize code, wasting time and generating a low quality product. Two major consequences of this is that the neural net is incapable of performing linear regression, only classification, and that the code runs abysmally slow. In order to actually solve the problems of the project, I was forced to use the Julia package `Flux.jl` for the linear regression and grid search over hyperparameters for the classification.

The reader can confirm that my neural net implementation does work on classification problems by running either the example script in the git repository or taking a look in the notebooks.

IV. RESULTS AND DISCUSSION

A. Linear Regression

The coupling coefficients that were found by applying different regression methods to (II.1) can be plotted as $L \times L = 40 \times 40$ matrices, as done in figures IV.1-IV.3. The least squares coefficients can be used as a base line to compare to Ridge and Lasso. The two diagonals offset by ± 1 from the center diagonal have the largest coefficients, all being about -0.5. This is due to the model not favoring J_{kl} over J_{lk}, opting to sharing the variability between them so that their sum is $J_{kl} + J_{lk} = -1.0$, which is the physically expected coupling coefficient. However, the least squares regression does not penalize model complexity, and sets the off-diagonals to small random values about the expected 0.0.

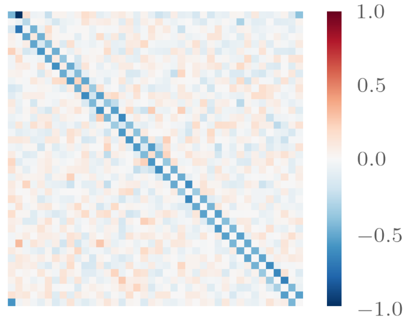

Figure IV.1: The coefficient matrix **X** found from least squares linear regression.

Applying Ridge regression with different regularization parameters yields improved results. As seen from fig. IV.2, the penalty sets the off-diagonal elements closer to zero as we would like. Since Ridge uses L_2 penalty, they do not get set to exactly zero.

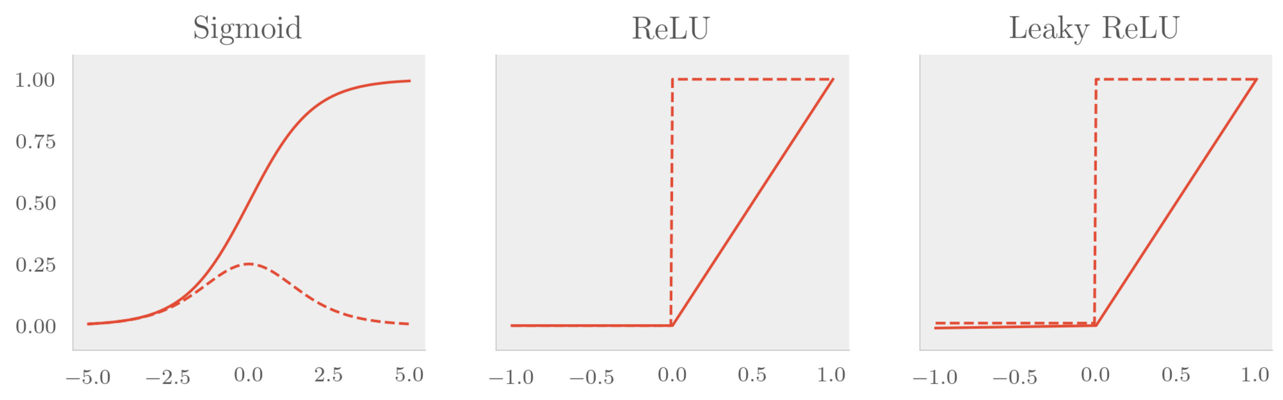

Figure II.2: The three activation functions studied. The whole lines show the functions themselves while the dashed their derivatives. The sigmoid has vanishing derivatives at both ends, ReLU at the negative side, while the leaky ReLU does not have any vanishing derivatives. The two latter lack continuous derivatives, seen by the jump at 0.

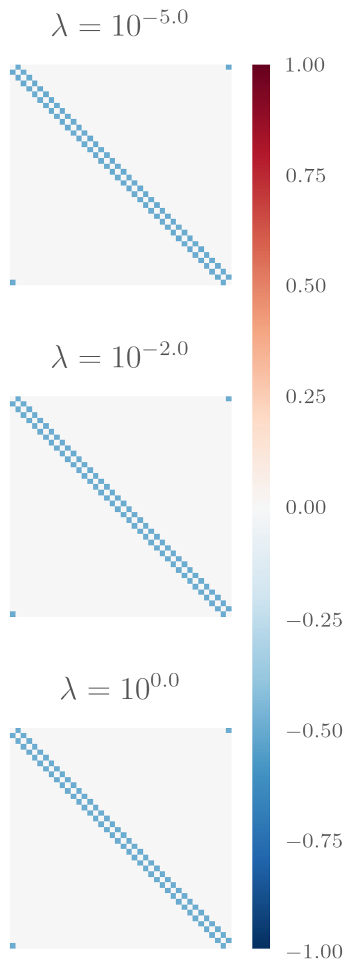

Figure IV.2: The coefficient matrix $\mathbf{X}$ found from Ridge regression for different values of regularization parameter. The change is so small that it is not noticeable.

Lasso behaves differently from the others as it explains all variability using the upper left triangular part of the coefficient matrix. This is owned to Lasso's L_1 penalization, forcing some coefficients to exactly zero. As all variability in an $\{J_{kl}, J_{lk}\}$ pair can be explained by one coefficient, the other is set to zero, giving only one diagonal. A too large regularization forces all elements to zero, as seen in the last panel of fig. IV.3, giving no predictive power.

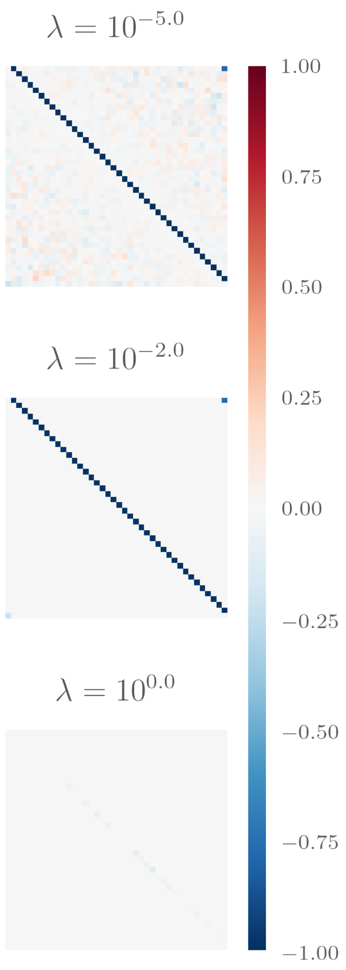

Figure IV.3: The coefficient matrix **X** found from Lasso regression for different values of regularization parameter. A small penalty removes the off-diagonal coefficients, while a too large coefficient removes them all.

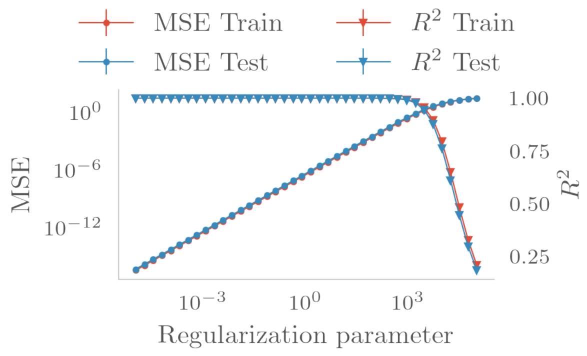

Figure IV.4: The mean square error and r^2 coefficient for Ridge regression with different regularization parameters λ from five fold cross validation. Any increase in the regularization worsens the performance, as both MSE and r^2 are monotonically decreasing for the training and test data.

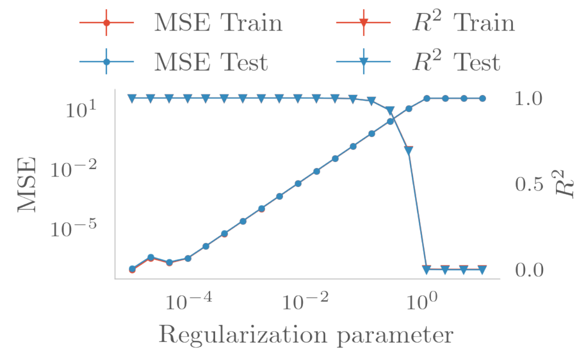

Figure IV.5: The mean square error and R^2 coefficient for Lasso regression with different regularization parameters λ from five fold cross validation. Any increase in the regularization worsens the performance, as both MSE and R^2 are monotonically decreasing for the training and test data.

By running a search over log-spaced regularization parameters, the optimal regularization can be found. This is done for Ridge in fig. IV.4 and for Lasso in fig. IV.5. The impact on both MSE and R^2 is shown. Somewhat unexpectedly, the models worsen as the regularization increases, both for training and test data. The Ridge is very stable over large ranges, decreasing only for very large regularizations. Lasso exhibits the same behavior, but dropping quickly to zero as all coefficients are forced to zero.

The explanation for this is the large number of redundant coefficients in the model. Only the two off-diagonal lines give non-zero contributions, explaining all the variance. Once a small regularization is applied, the redundancy is quickly decreased, decreasing the model variance, and won't decrease again until it is strong enough to affect the off-diagonal terms, at which point the bias quickly increases.

1. *Regression with Neural Nets*

A neural net was trained using a single linear layer on the same data as for the regression methods. The net was highly sensitive for different learning rates, so a low rate of $\eta = 0.0001$ had to be used. Nevertheless, the net quickly converged, suggesting the weights shown in fig. IV.6. Comparing to the previous results, we see that it is qualitatively identical to least squares and Ridge, but having off-diagonal less noise than least squares and more than Ridge. This is expected, as the neural net does not use any regularization.

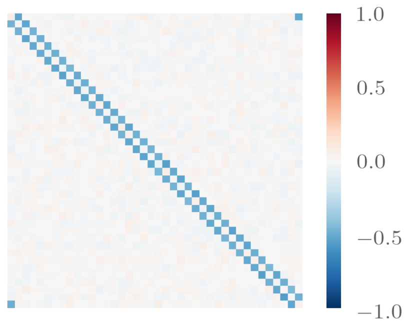

Figure IV.6: The coefficient matrix X found by neural net. The prediction using X has $R^2 = 1.0$ test score.

B. Classification

Classification was performed on ordered and disordered data with $N = 10000$ samples. It was discovered that the accuracy score varied wildly depending on which data was used while cross entropy loss remained nearly constant, but using the whole data set proved to be too time consuming.

The implementations of normal gradient descent and Nesterov gradient descent were used together with stochastic descent from `sklearn`. The behavior of the former are shown in fig. IV.7. Both methods overfit the data quickly, with Nesterov being much faster to overfit. To remedy this, early stopping was used, causing both to stop after only $3 - 10$ iterations. The results are shown in table IV.1.

These results are in agreement with Mehta [3] with test classification accuracy being around 50%. Since the problem is a binary classification problem, this is disappointing as it is no better than a random guess.

Method	Accuracy score	
	Train	Test
GD	0.61	0.49
NGD	0.51	0.70
SGD	0.75	0.52

Table IV.1: The accuracy score for both training and test data for custom implementations of gradient descent and Nesterov descent, as well as `sklearn`'s stochastic gradient descent. The samples were $N = 10000$. The results varied wildly depending on the data set used.

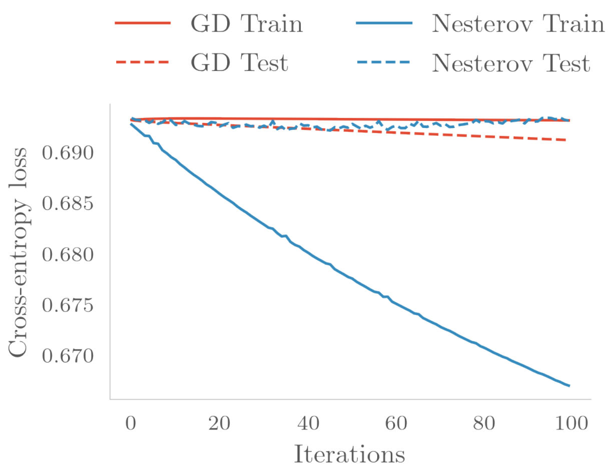

Figure IV.7: The log cross entropy loss for each iteration of normal gradient descent and Nesterov gradient descent with learning rate $\eta = 0.001$ for gradient descent and $\eta = 0.0001$ for Nesterov. The Nesterov descent used a batch size 250.

C. Classification With Neural Nets

While classifying with a logistic classifier has only a handful of hyperparameters to tune, neural nets have so many that the parameter space can not be searched within practical time limits. In addition, tuning many parameters begs for visualizing high dimensional data, which quickly becomes noisy and uninformative. As such, only learning rates, activation function and number of hidden neurons were tuned.

Figures IV.8, IV.9 and IV.10 show the test accuracy as a function of learning rate η and number of neurons in the hidden layer for sigmoid, ReLU and leaky ReLU activation functions. At a glance there are some noticeable differences.

For both ReLU and leaky ReLU, a too high learning rate ends in divergence. The reason is that the network is taking too large steps and quickly diverging. However, once the learning rate is low enough, the fast learning of ReLU's allows for the networks to get high accuracy even with few neurons. A near perfect test accuracy is achieved with only 5 neurons in both cases.

The sigmoidal networks behave differently. The best results are achieved at high learning rates, becoming progressively poorer at lower rates. In addition, few neurons are preferred. Both of these facts suggest that the network learns too slowly to fully exploit more neurons, something which is only worsened by lower learning rates.

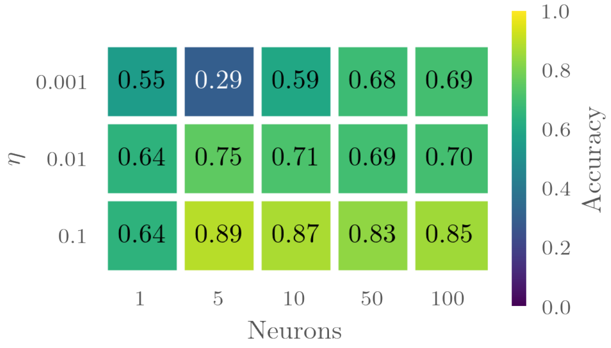

Figure IV.8: The test accuracy of a NN with sigmoid activation functions in the hidden layer. The network performs best at high learning rates and few neurons.

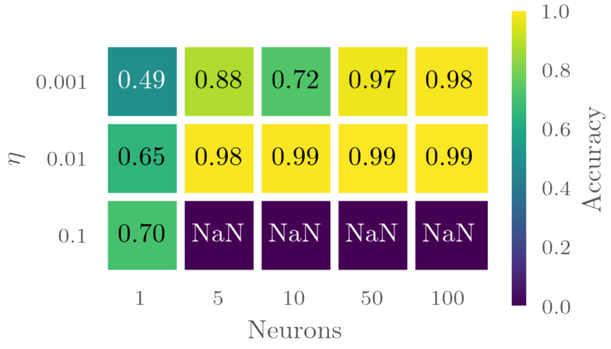

Figure IV.9: The test accuracy of a NN with ReLU activation functions in the hidden layer. For high learning rates the network diverges. Best results are achieved with more than 1 neuron and low learning rates.

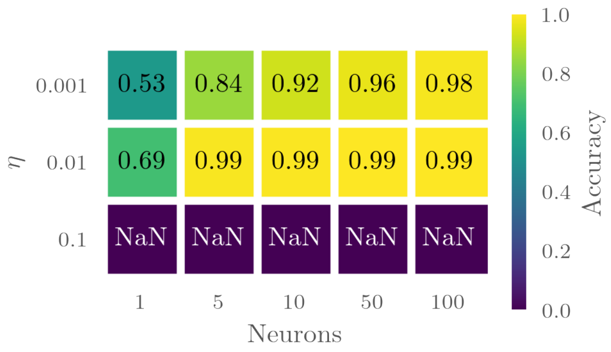

Figure IV.10: The test accuracy of a NN with leaky ReLU activation functions in the hidden layer. Diverges for too high learning rates, while achieved nearly perfect accuracy for all other cases.

The near perfect learning rates are at first sight dubious, but this is consistent with independent analyses, most notably Mehta et al.[3]. It is neither an unreasonable result, as the classification problem is binary with predictors of equal variance. There is also the additional benefit that random data can be classified as unordered systems, as this would indeed be correct. In other words, the network does not need to learn what unordered systems look like, only what ordered systems look like, as everything that is not ordered, be it "real" unordered systems or random data, are indistinguishable from unordered.
In total, using ReLU or leaking ReLU with low learning rates and a few neurons gives optimal results with the least computational demand. The classification was not tested in the critical regime, something which should be performed in future work.

V. CONCLUSION

Neural network is a very versatile method capable of repeating and outperforming the results of more traditional machine learning methods in linear and logistic regression. Ridge and Lasso regressions were found to outperform neural nets on regression, but this is not comparable as the neural nets used were the simplest possible, with no regularization. For classification problems the neural net greatly outperformed logistic regression by gradient descent. The results obtained are all in agreement with Mehta et al. The full extent of neural nets were not adequately explored due to design flaws and lack of time, but remains an area for future investigation.

[1] L. Onsager, Physical Review **65**, 117 (1944).
[2] J. F. Trevor Hastie, Robert Tibshirani, *The Elements of Statistical Learning* (Springer, 2017).
[3] P. Mehta, M. Bukov, C.-H. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, ArXiv e-prints (2018), arXiv:1803.08823 [physics.comp-ph].