Project 2

Edvard B. Rørnes* and Isak O. Rukan†
Institute of Physics, University of Oslo,
0371 Oslo, Norway
(Dated: November 3, 2024)

In recent years, neural networks have become increasingly important for advancements in various scientific fields. In this report, we develop a program[a] that consists of both linear and logistic regression methods and a feed-forward neural network. In particular, the program implements gradient descent, stochastic gradient descent, and a simple feed-forward neural network with backpropagation. The neural network is evaluated against linear regression on the Franke function and against logistic regression using sklearn's breast cancer data, where it identifies malignant tumors. We explore three activation functions: Sigmoid, ReLU, and LeakyReLU. The analysis involves tuning the number of epochs, hidden nodes, and the hyperparameters λ (regularization) and η (learning rate) to optimal values. On the Franke function, our own implementation of the neural network achieved a peak R^2-score of x and y respectively. For the cancer data, the highest accuracies achieved with ReLU, , and Sigmoid activation functions were $97.9\%, 98.6\%$, and 97.9%, respectively, compared to $z\%$ for logistic regression. Notably, both the Sigmoid and LeakyReLU activation functions exhibited minimal sensitivity to variations in the hyperparameter λ at optimal learning rates, whereas ReLU showed significant responsiveness. Overall, while LeakyReLU yielded the best performance across all tests, its improvement over the other activation functions was marginal when applied to the cancer dataset, but considerate on the Franke function. Additionally, the Sigmoid activation function was found to be appreciably slower than the ReLU variants.

1. INTRODUCTION

Over the last few years, machine learning and neural networks have become an increasingly important part of data analysis with an enormous range of applications. From image recognition to predictive analytics and scientific simulations, these techniques are reshaping the way the scientific community tackles complicated problems. The flexibility of neural networks in approximating complex, non-linear relationships has made them invaluable across diverse fields, such as biology, engineering, finance, physics etc. For just a few examples, see [1–3].

Neural networks are a powerful tool for handling large datasets where traditional modeling may fall short. This allows us to extract patterns and make accurate predictions. As the applications of these techniques continue to expand, so does the demand for a deeper understanding of their underlying principles and implementation details. Thus, we aim to investigate the foundational aspects of neural networks, explore various activation functions and learning algorithms, and study their effects on model performance. By doing so, we hope to develop a robust framework for applying neural networks effectively to complex real-world data.

The main goal of this project is understand the various techniques behind machines learning, and investigate they can be applied to the healthcare system, specifically in diagnosing malignant breast cancer tumors based on the tumor's several features based on the data from `sklearn`'s datasets [4]. We do this by testing the performance of a neural network by comparing it with logistic regression as we are working with discrete data.

We begin by introducing the relevant background necessary to understand the implementations. Then we outline said implementations before discussing the results. We go over both linear and non-linear regression on the Franke function where performance issues in the implementation are easier to visualize. Finally the neural network and logistic regression are applied to the cancer data.

2. METHODS

In this section we present the various methods used in this report.

2.1. Linear Regression

As discussed in a previous project [5], linear regression is the simplest method for fitting a continuous given a data set. The data set is approximated by $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}$ and the β coefficients are found by minimizing the cost function. For this project we consider the two regression methods:

$$C_{\text{OLS}}(\boldsymbol{\beta}) = \frac{2}{n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2, \tag{1}$$

$$C_{\text{Ridge}}(\boldsymbol{\beta}) = C_{\text{OLS}}(\boldsymbol{\beta}) + \lambda||\boldsymbol{\beta}||_2^2. \tag{2}$$

We then insist that the derivative of these w.r.t. $\boldsymbol{\beta}$ is 0, and choose the resulting β coefficients as out model.

* e.b.rornes@fys.uio.no
† icrukan@uio.no
[a] Github

Doing this we arrive at:

$$\boldsymbol{\beta}_{\text{OLS}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}, \tag{3}$$

$$\boldsymbol{\beta}_{\text{Ridge}} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{4}$$

2.2. Regularization Terms

Regularization is a technique to prevent overfitting by adding a penalty to the cost function that discourages complex models. Overfitting occurs when a model learns the noise in the training data rather than the underlying patterns, leading to poor generalization on unseen data. In the context of neural networks, regularization plays a critical role, especially when working with architectures that have many parameters.

The two common regularization methods that we inspected previously are Ridge and Lasso regularization. In this project, we will only be considering Ridge regularization, where the cost function is given by

$$C_{\text{Ridge}}(\boldsymbol{\beta}) = C_{\text{OLS}}(\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}_2^2, \tag{5}$$

where $C_{\text{OLS}}(\boldsymbol{\beta})$ is the ordinary least squares cost function, $\boldsymbol{\beta}$ represents the model parameters, and the hyperparameter λ controls the magnitude of the penalty to large coefficients. For more details on regularization in linear regression, see [5].

In the context of neural networks, regularization techniques are crucial for controlling model complexity and improving generalization. Ridge regularization can be implemented in neural networks by applying L2 regularization to the weights of the network. This approach adds the penalty term $\lambda\|\mathbf{W}\|_2^2$ to the loss function, where $\mathbf{W}$ is the weight matrix of the neural network. The modified cost function for a neural network with L2 regularization becomes

$$C_{\text{NN}} = C_{\text{loss}} + \lambda\|\mathbf{W}\|_2^2, \tag{6}$$

where C_{loss} will be the MSE for regression tasks and cross-entropy loss for classification tasks.

Incorporating regularization in neural networks offers several important benefits. Firstly, L2 regularization effectively applies weight decay, which shrinks the weights during training and helps prevent the model from becoming overly complex. This leads to smoother decision boundaries and reduces the risk of overfitting, allowing the model to generalize better to unseen data. Secondly, regularized models tend to exhibit increased robustness to variations in the data, as the penalty encourages the learning of simpler models that do not fit the noise in the training set. Lastly, by tuning the hyperparameter λ, practitioners can effectively control the model's complexity, striking a balance between bias and variance and tailoring the model to the specific problem at hand. Overall, these advantages highlight the critical role that regularization plays in developing neural networks that perform well in real-world applications.

Overall, incorporating regularization terms such as Ridge regularization in the training of neural networks is essential for developing models that generalize well to new, unseen data.

2.3. Logistic Regression

Whilst linear regression is quite successful in fitting continuous data such as terrain data, when the output is supposed to be discrete it fails. Linear regression predicts values across a continuous spectrum, resulting in predictions outside the range of valid class labels, such as giving negative probabilities. Logistic regression on the other hand is specifically designed for binary classification problems, and is thus ideal when dealing with discrete outcomes.

Logistic regression models the probability that a given input belongs to a particular class, typically using the sigmoid function to map any real-valued number to a value between 0 and 1. Given an input vector $\boldsymbol{X}$ and weights $\boldsymbol{\beta}$, logistic regression predicts the probability of the class as:

$$P(y = 1|\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{\beta}) = \frac{1}{1 + e^{-\boldsymbol{X}\boldsymbol{\beta}}} \tag{7}$$

where σ represents the sigmoid function. To find optimal weights, logistic regression minimizes the cross-entropy cost function:

$$C(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \ln(\hat{y}_i) + (1 - y_i)\ln(1 - \hat{y}_i)\right) \tag{8}$$

where y_i is the class label and $\hat{y}_i$ is the predicted probability for sample i. To penalize overfitting, and regularization term may be added to $C(\boldsymbol{\beta})$. This terms adds a penalty for large weights, trying to keep the weights (relatively) small. Adding the ℓ^2 regularization term results in

$$C(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \ln(\hat{y}_i) + (1 - y_i)\ln(1 - \hat{y}_i)\right) + \lambda\sum_{i=1}^{n} w_j^2. \tag{9}$$

2.4. Resampling Methods

Resampling methods are used to estimate the accuracy of predictive models by splitting the data into training and testing sets or by generating multiple datasets. Common techniques include cross-validation and bootstrapping. In cross-validation, the data is split into k folds, and the model is trained on $k - 1$ folds and tested on the remaining fold. This process is repeated k times, and the average accuracy is computed. Bootstrapping involves sampling with replacement from the dataset to create multiple training sets. These methods help assess model stability and generalizability on unseen data.

2.5. Gradient Descent

Gradient descent (GD) is an essential optimization algorithm in machine learning, commonly used to minimize cost functions by adjusting model parameters iteratively. Given model parameters θ and a cost function $C(\theta)$, the GD update rule adjusts parameters in the opposite direction of the gradient:

$$\theta_i^{(j+1)} = \theta_i^{(j)} - \eta \frac{\partial C}{\partial \theta_i} \tag{10}$$

where η is the learning rate. Batch gradient descent (BGD) calculates the gradient over the entire dataset:

$$\theta^{(j+1)} = \theta^{(j)} - \eta \nabla_\theta C \tag{11}$$

BGD is computationally expensive for large datasets but provides smooth convergence toward the minimum.

2.6. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a variation of gradient descent where each parameter update is performed on a single data point or a small batch. The update rule for SGD is:

$$\theta_i^{(j+1)} = \theta_i^{(j)} - \eta \frac{\partial C^{(i)}}{\partial \theta_i}$$

where $C^{(i)}$ is the cost function evaluated at a single data point i. While SGD introduces noise in the updates, it often converges faster for large datasets and helps escape local minima, making it ideal for training neural networks.

2.7. Neural Networks

Neural networks are computational models inspired by the human brain, designed to recognize patterns and relationships within data. They consist of layers of interconnected neurons or nodes, where each neuron applies a transformation to the input data. In each layer, neurons take a weighted sum of inputs, apply an activation function to introduce non-linearity, and pass the result to the next layer. The final layer produces the output, serving as the network's prediction.

2.7.1. Feed Forward Neural Networks

Feed-forward neural networks (FFNNs) are the simplest type of neural network, where data flows forward from input to output without forming cycles. These networks contain one or more hidden layers that apply an activation function to capture complex, nonlinear patterns in the data. The training process adjusts the weights of each connection to minimize a cost function, typically using gradient descent.

2.7.2. Activation Functions

Activation functions play a critical role in neural networks by introducing non-linearity, which enables the network to approximate more complex functions beyond simple linear mappings. In this project, we use three different activation functions: sigmoid, tanh, ReLU, and Leaky ReLU. Each function has different properties that can impact training performance and convergence.

- The sigmoid function, suitable for binary classification, takes an input $z \in \mathbb{R}$ and outputs a value in the range $(0, 1)$. This makes it useful for probabilistic interpretations. As mentioned prior, it is given by:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{12}$$

- The ReLU (Rectified Linear Unit) function activates only positive values:

$$R(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ z & \text{if } z > 0 \end{cases}. \tag{13}$$

  This reduces the number of calculations that the network has to perform and can speed up the training.

- The Leaky ReLU (LReLU) function is a variation of ReLU that allows a small gradient when $z \leq 0$. This can help mitigate a known issue known as "dying ReLU" where neurons become inactive due to consistently recieving negative inputs., helping to mitigate issues with inactive neurons. It is given by:

$$LR(z) = \begin{cases} az & \text{if } z \leq 0 \\ z & \text{if } z > 0 \end{cases} \tag{14}$$

  where a is some small number. In this project we consider only $a = 0.01$.

By selecting appropriate activation functions for each layer, FFNNs can effectively capture complex data patterns, enhancing model performance.

2.7.3. Backpropagation

Backpropagation is the key algorithm for training neural networks by optimizing weights to minimize the cost function. It works by propagating the error backward from the output layer to the input layers, computing gradients for each weight based on the error. These gradients are then used to update the weights, enabling the network to learn from its errors and make more accurate predictions over time.

3. IMPLEMENTATION

(Write about your implementation Isak)

Further, the Feedforward Neural Network (FFNN) was implemented to be able to analyze both the Franke function and the breast cancer data. The FFNN is structured with an input layer, one or more hidden layers, and an output layer, with each layer utilizing an activation function, and currently only supports the Adam optimizer. The architecture is defined by specifying the input size, the number of neurons in hidden layers, and the output size. In the case of the Franke function, the input layer is size 2, one for x and one for y, whilst for the breast cancer data the input layer corresponds to the number of features, i.e. 30, which corresponds to several properties of the tumor, such as circumference, density etc.

We tested out various different ways of layering the hidden layers. In the end, for the Franke function, we settled with using a pyramid-like architecture for the hidden layers $[4, 8, 16, 32, 16, 8, 4, 2]$ whilst for the breast cancer data we used $[15, 30, 15, 8, 4, 2]$. These were decided on based on the size of both the input and output, where the latter is 1 in both cases. The weights and biases are initialized using random values scaled by the number of input neurons, which aids in faster convergence during training. The network supports ReLU, Sigmoid, and Leaky ReLU as activation functions. The forward propagation computes the output of the network by applying the activation function to the weighted sum of inputs at each layer. The backward propagation algorithm updates the weights and biases using gradient descent, where the mean squared error serves as the loss function. We implement regularization techniques to mitigate overfitting.

The training process includes splitting the data into training and test sets, followed by iterating through a specified number of epochs, during which the network adjusts its weights to minimize the error. The test size used throughout is 25%. After training, the network can predict outputs for new data, allowing for evaluation against known values from the Franke function. The overall performance of the model is assessed using MSE and accuracy for Franke and cancer data respectively.

4. RESULTS & DISCUSSION

We represent the results and compare the various methods. The larger plots have been placed in Appendix A to preserve space for the sake of readability.

4.1. Franke

The results for the MSE and R^2 as a function of the learning rate η with 1000 epochs with our own FFNN with different activation functions and keras NN with the sigmoid activation function are given in Fig. 1. In this particular plot we are not using any regularization terms. The reason for this is simply due to `keras` being very slow, but we still wanted to include it as a reference. The optimal learning rates for $\lambda = 0$ can easily be read off this plot.

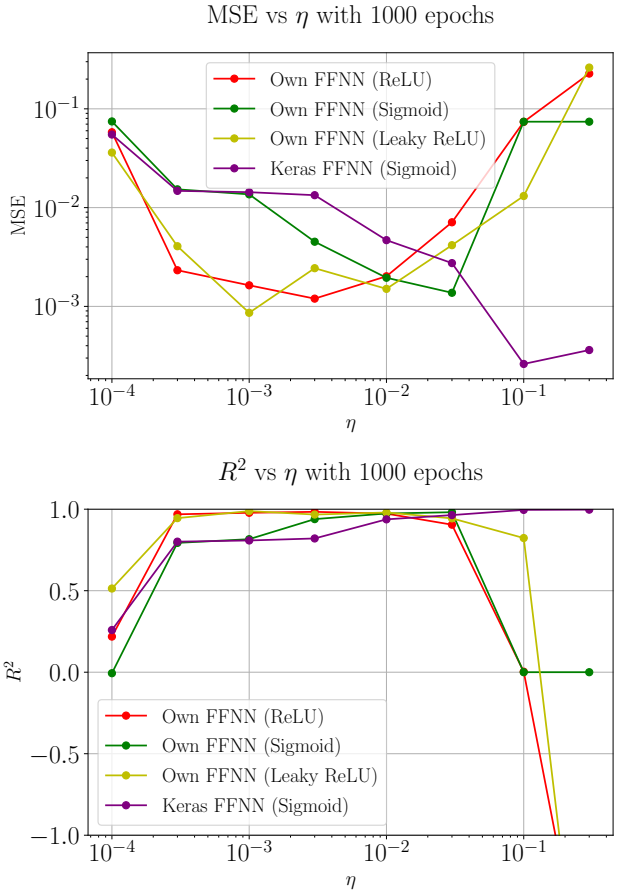


Fig. 1: MSE and R^2 regression results for the FFNN as a function of η with 1000 epochs.

Further we used the result for the best learning rate to plot the MSE after each epoch, given in Fig. 2. As expected, all activation functions perform poorly on low epochs, and gradually improve as you go along. At the end, all seem to achieve a relatively good MSE as the number of epochs increase.

Overall these metrics seem to imply that ReLU and LReLU are outperforming `keras`, but when explicitly plotting the predictions this does not seem to be the case, as can be seen in Fig. 5a. Note that this plotted prediction is a rerun with the best parameters for each activation type. Thus their performance being worse here compared to `keras` may simply be due to large variation from one run to another in our implementation.

We then did the above again, but now with a regularization parameter λ and excluded `keras` due to time

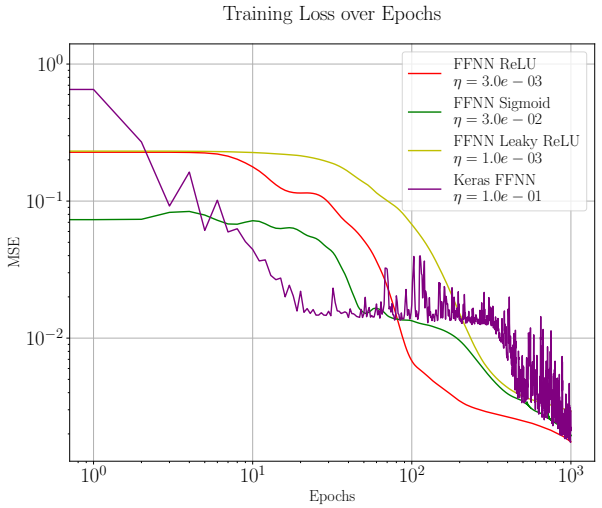


Fig. 2: The MSE regression results for the FFNN as a function of number of epochs for ReLU, LReLU, Sigmoid and keras.

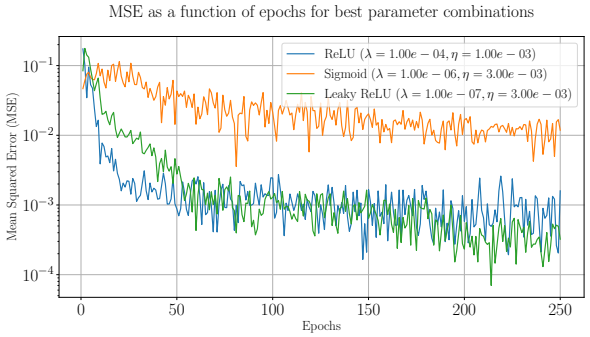


Fig. 3: The MSE regression results for the FFNN as a function of number of epochs for ReLU, LReLU, Sigmoid with the best performing combination of λ and η.

constraints. The MSE for various combinations of η and λ are given in Fig. 6. The best results from here are then picked to once again plot the MSE over epochs given in Fig. 3 and another 3D plot to visualize the results in Fig. 5b. Clearly the sigmoid activation function performs worse overall, whilst ReLU and LReLU have a close competition between them. The convergence is much faster here, and we same performance with ReLU and LReLU after only 50 epochs. The sigmoid function converges much slower, but this is due to it being suited for binary classification and not regression. The much faster convergence does suggest that the regularization parameter is improving our performance significantly.

4.2. Cancer Data

The accuracy for logistic regression and our own FFNN for the three different activation functions and for different values of η and λ are given in Figs. 4 and 7 respectively.

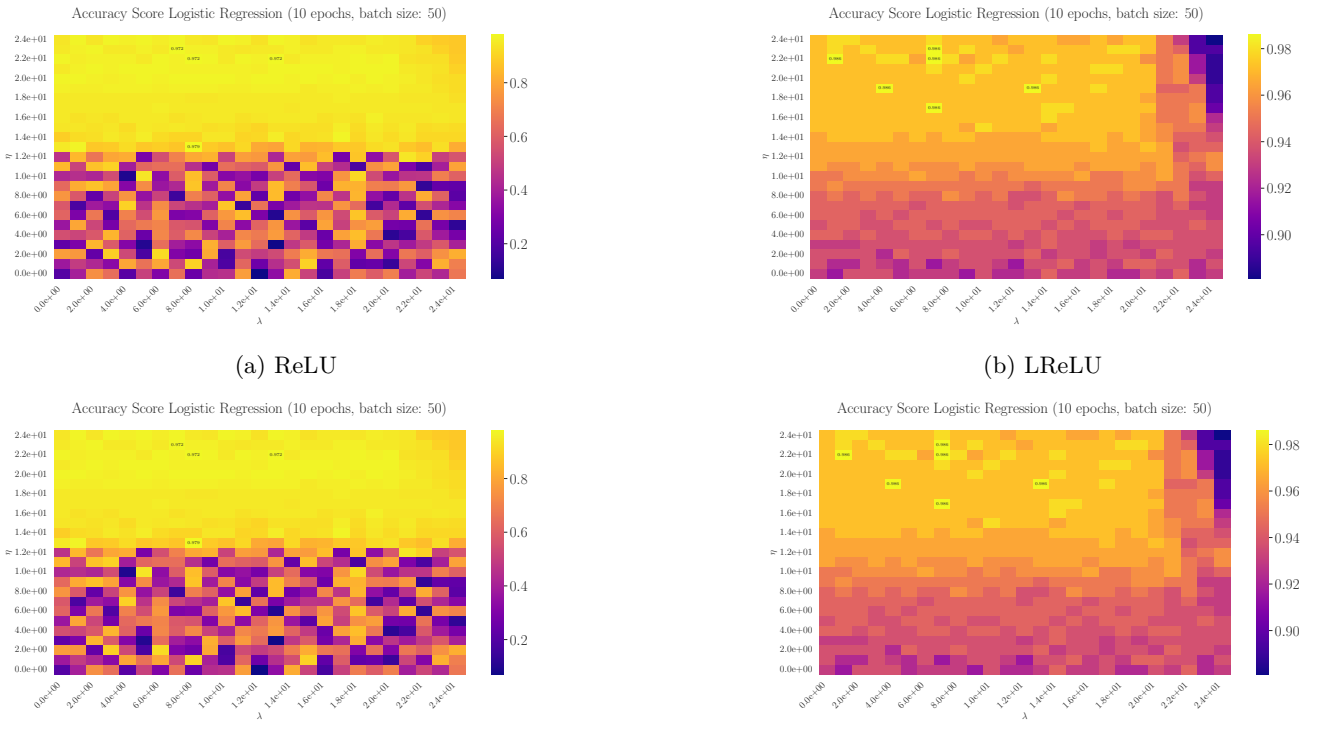
5. CONCLUSION

On the Franke function, we found that Sigmoid is not overly sensitive to changes in the learning rate and hyperparameter λ, but also struggles to get good results. On the other hand ReLU has the ability to get great results, but only for very specific combinations of η and λ. LReLU seems to be the best of both worlds, being relatively insensitive to getting the exact right combination of η and λ, whilst still managing to obtain very good MSE scores like ReLU. For logistic regression our FFNN performs quite well, approaching 98% accuracy with certain parameter combinations.

A step forward would for example be testing out different values of a in LReLU and seeing whether it may achieve a higher level of performance. Thoroughly testing out other optimizers such as Adagrad and RMSprop for the FFNN may also yield benefits, but due to time constraints we did not implement this properly.

[1] A. Dawid, J. Arnold, *et. al.*, *Modern applications of machine learning in quantum sciences*, 2023.
[2] V. Thapar, *Applications of machine learning to modelling and analysing dynamical systems*, 2023.
[3] F. G. Mohammadi, F. Shenavarmasouleh, and H. R. Arabnia, *Applications of machine learning in healthcare and internet of things (iot): A comprehensive review*, 2022.
[4] F. Pedregosa, G. Varoquaux, *et. al.*, *Scikit-learn: Machine learning in python*, *Journal of Machine Learning Research* **12** (2011) 2825–2830.
[5] I. Rukan and E. R. rnes.

Appendix A: Large Figures



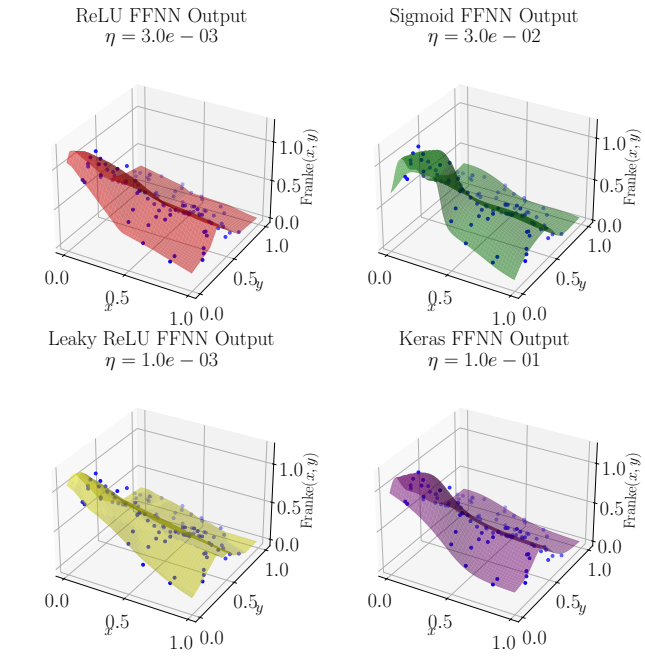
(a) ReLU

(b) LReLU

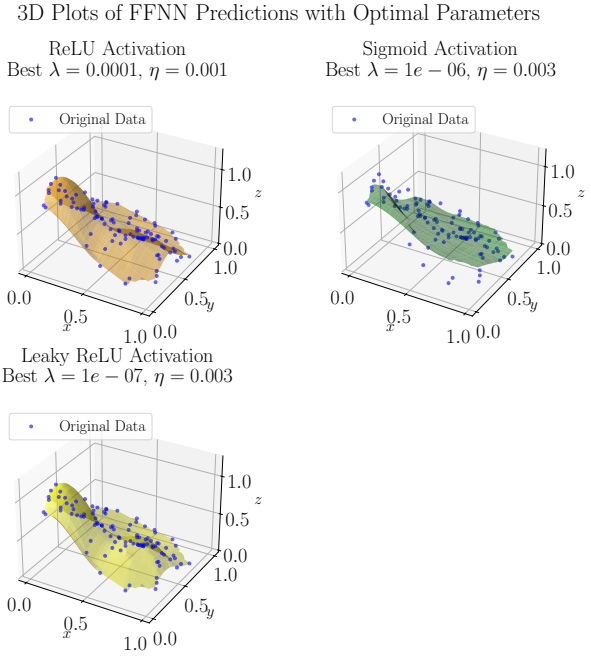
(c) Sigmoid

(d) LReLU

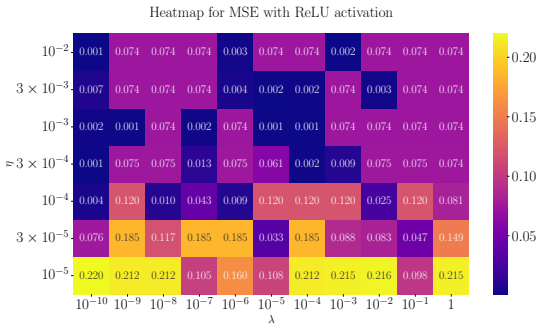
Fig. 4: Accuracy score for logistic regression, for number of epochs $N = 10$ (left), $N = 100$ (right). The figures to the right are zoomed in versions of the ones to the left.



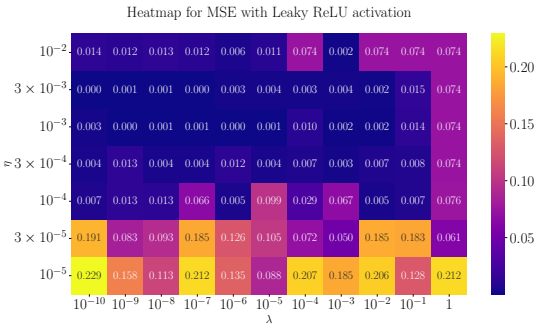
(a) 3D plots for best η with 1000 epochs. The blue dots correspond to the sampled points from the Franke function with 100 total samples.



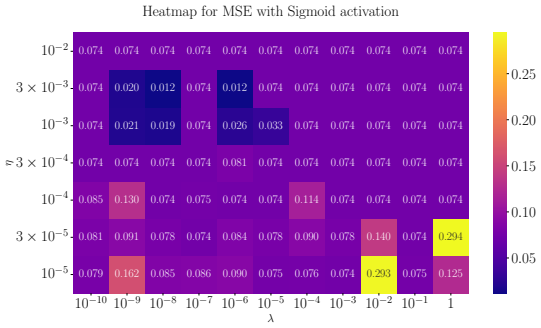
(b) 3D plots for best combination of λ and η for our own neural network with 250 epochs and 100 samples from the Franke function.



(a) ReLU

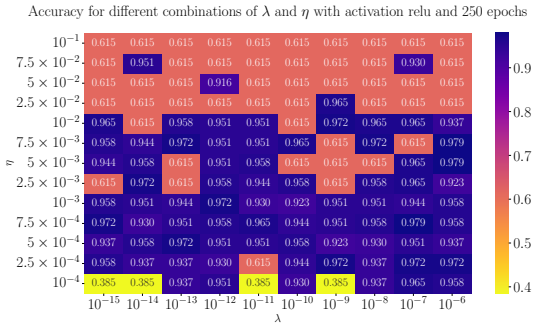


(b) LReLU

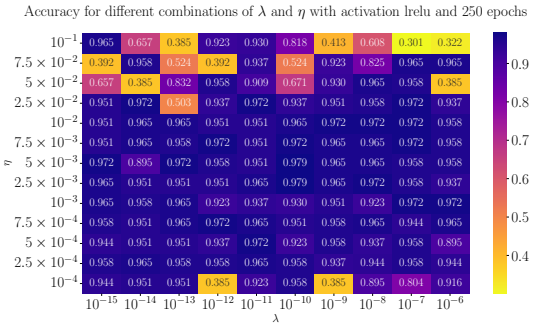


(c) Sigmoid

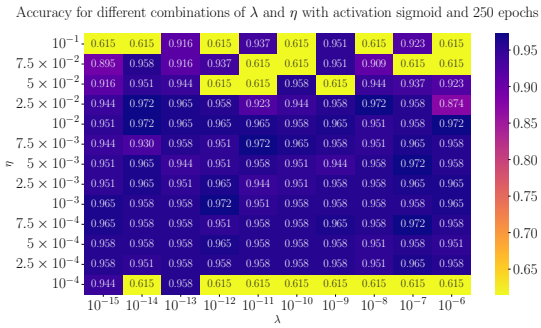
Fig. 6: MSE for various combinations of η and λ for ReLU, LReLU and Sigmoid activation functions with 250 epochs on the Franke function.



(a) ReLU



(b) LReLU



(c) Sigmoid

Fig. 7: Accuracy for various combinations of η and λ for ReLU, LReLU and Sigmoid activation functions with 250 epochs on the breast cancer dataset.