Project 2

Edvard B. Rørnes[*] and Isak O. Rukan[†]

Institute of Physics, University of Oslo,
0371 Oslo, Norway

(Dated: November 2, 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. 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 performed poorly compared to linear regression, with peak R^2-scores of x and y respectively. For the cancer data, the highest accuracies achieved with ReLU, LeakyReLU, and Sigmoid activation functions were $97.9\%, 98.6\%$, and 97.9%, respectively, compared to $z\%$ for logistic regression. Notably, the Sigmoid activation function exhibited minimal sensitivity to variations in the hyperparameter λ at optimal learning rates, whereas ReLU and LeakyReLU 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. 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 indispensable 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. The results 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. THEORY

In this section we derive and discuss the relevant theory behind the methods used.

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:

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

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

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. The two common regularization methods that we inspected previously are Ridge and Lasso regularization. In this project we will only be considering Ridge, 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 the hyperparameter λ controls the magnitude of the penalty to large coefficients. For more details see [5].

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:

$$P(y = 1 | \boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{\beta}) = \frac{1}{1 + e^{-\boldsymbol{X}\boldsymbol{\beta}}} \tag{6}$$

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

$$C(\boldsymbol{\beta}) = -\frac{1}{n} \sum_{i=1}^{n} (y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i)) \tag{7}$$

where y_i is the class label and $\hat{y}_i$ is the predicted probability for sample i.

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:

$$\theta_i^{(j+1)} = \theta_i^{(j)} - \eta \frac{\partial C}{\partial \theta_i} \tag{8}$$

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

$$\theta^{(j+1)} = \theta^{(j)} - \eta \nabla_\theta C \tag{9}$$

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:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{10}$$

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

$$R(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ z & \text{if } z > 0 \end{cases}. \tag{11}$$

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:

$$LR(z) = \begin{cases} az & \text{if } z \leq 0 \\ z & \text{if } z > 0 \end{cases} \tag{12}$$

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

In this project, we implement a Feedforward Neural Network (FFNN) to analyze the Franke function. The FFNN is structured with an input layer, one or more hidden layers, and an output layer, with each layer utilizing an activation function. The architecture is defined by specifying the input size, the number of neurons in hidden layers, and the output size.

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 several activation functions, including ReLU, Sigmoid, and Leaky ReLU. The activation functions are chosen based on their performance in the forward pass, where the input data is transformed through the layers of the network.

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. 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 metrics such as accuracy and mean squared error.

4. RESULTS & DISCUSSION

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 and keras NN with different activation functions are given in Fig. 1. Here we can see that the MSE (R^2) decrease (increase) as the learning rate becomes larger where the maximal learning rate is given by 0.3. The sigmoid activation can be seen to perform quite poorly here, only
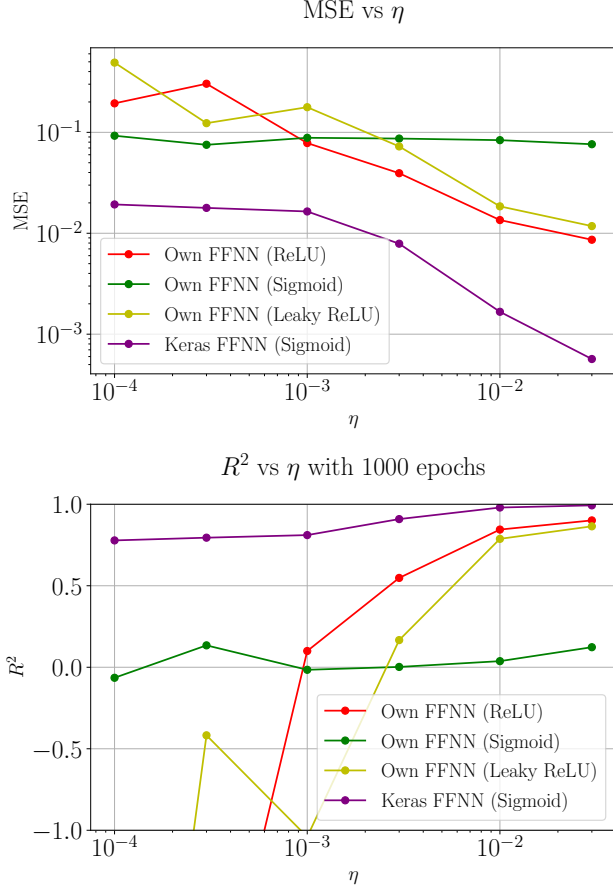
of these issues are likely a problem with our implementation.

Further we used the result for the best learning rate to plot the MSE after each epoch, given in Fig. 2. The sigmoid activation function seems to flatten quite early on, and already get diminising returns after epoch number 10. ReLU and LReLU both get increasingly better as we increase the number of epochs, flattening towards the
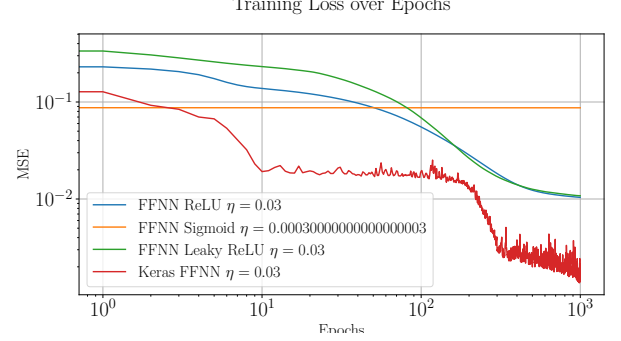

Fig. 2: The MSE regression results for the FFNN as a function of number of epochs for ReLU, LReLU, Sigmoid and keras.

end of the plot. Finally the keras version with sigmoid as the activation function has negligible improvements between epochs 10-200, but improves massively after that. It clearly outperforms our implementations throughout.

This can be seen even clearer in Fig. 3. Here we plot the predictions of each activation function given 250 samples, represented as blue dots, from the Franke function. The sigmoid activation function seemingly only maps the average of all the points. ReLU and LReLU perform appreciably better, but are dwarfed when compared to keras' sigmoid. The latter almost perfectly fits the Franke function given the number of points, in agreement with its much lower MSE.

4.2. Cancer Data

The accuracy of the FFNN for the three different activation functions for different values of η and λ are given in Fig. 4.

5. CONCLUSION

Fig. 1: MSE and R^2 regression results for the FFNN as a function of η with 1000 epochs. The results quickly diverge past 0.3 thus we only plot that far.

achieving a measly R^2 score of ~ 0.1, being slightly better than taking the average.

Increasing the learning rate any higher than $\eta = 0.3$ immediately begins to cause large divergences for multiple of the activation functions. This is on the contrary to what one would expect. If, say $\eta = 0.3$ is the optimal learning rate for a given activation function, one would expect $\eta = 0.4$ to give a slightly larger MSE, but not completely diverge. It is not clear what could be causing this. The fact that all activation functions also perform best at the same learning rate is also unexpected. Both

[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.
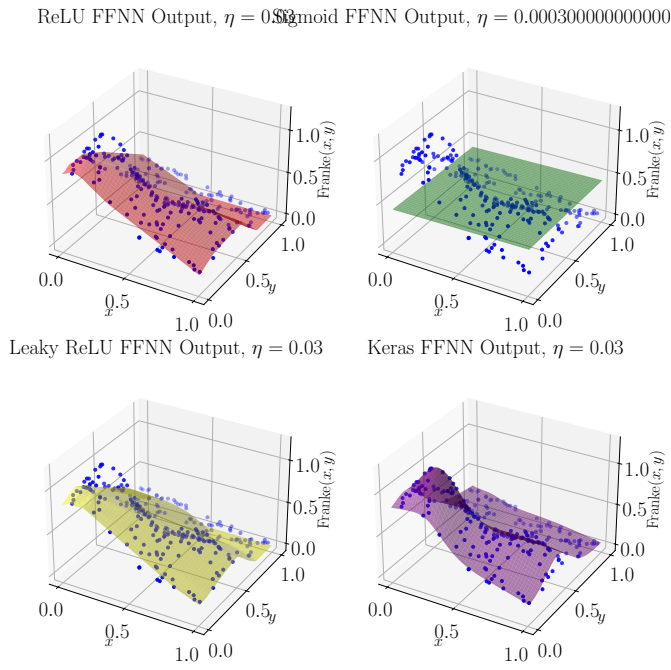
ReLU FFNN Output, $\eta = 0.03$ Sigmoid FFNN Output, $\eta = 0.000300000000000$

[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.

Leaky ReLU FFNN Output, $\eta = 0.03$ Keras FFNN Output, $\eta = 0.03$

Fig. 3: 3D plots of the regression results for the FFNN with activation functions ReLU, LReLU and Sigmoid, along with keras' neural network using the Sigmoid activation. The blue dots correspond to the sampled points from the Franke function with 250 total samples. 1000 epochs were used.
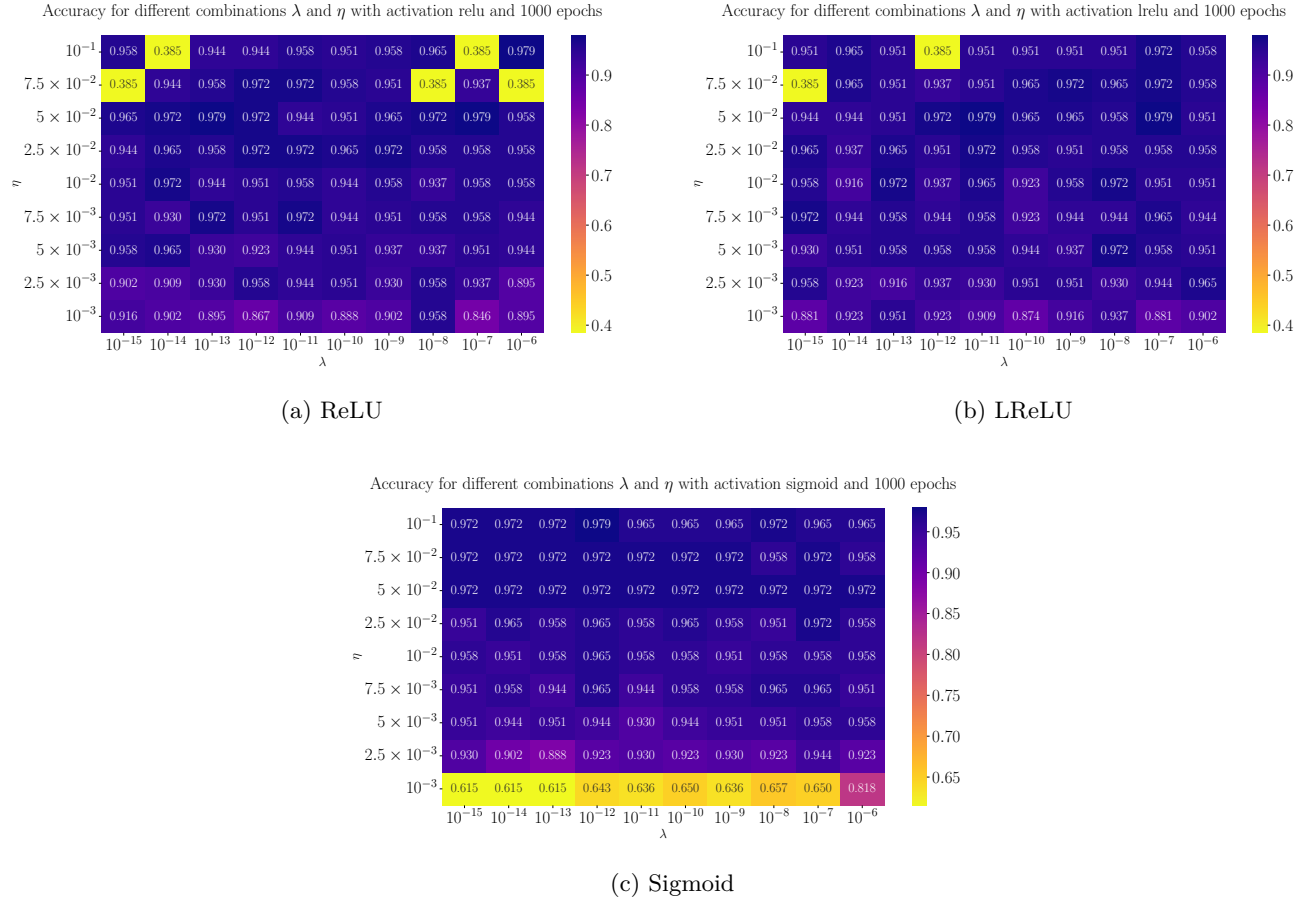
(a) ReLU

(b) LReLU

(c) Sigmoid

Fig. 4: Accuracy for various combinations of η and λ for ReLU, LReLU and Sigmoid activation functions with 1000 epochs.