Title

Anton Nicolay Torgersen
University of Oslo

(Dated: November 2, 2025)

This study details the implementation of a flexible feed-forward neural network (FFNN) from scratch, featuring configurable layers, nodes, and activation functions (Sigmoid, ReLU, Leaky ReLU). We implement backpropagation with optimization algorithms like RMSprop and ADAM. The FFNN is first benchmarked on a regression task (Runge's function, $f(x) = 1/(1 + 25x^2)$), comparing it to previous OLS, Ridge, and LASSO models. The network is then adapted for classification using a Softmax output and cross-entropy cost function, and its accuracy is evaluated on the MNIST dataset. We analyze the impact of network architecture and regularization, benchmarking our results against standard libraries.

I. EXERCISE 44

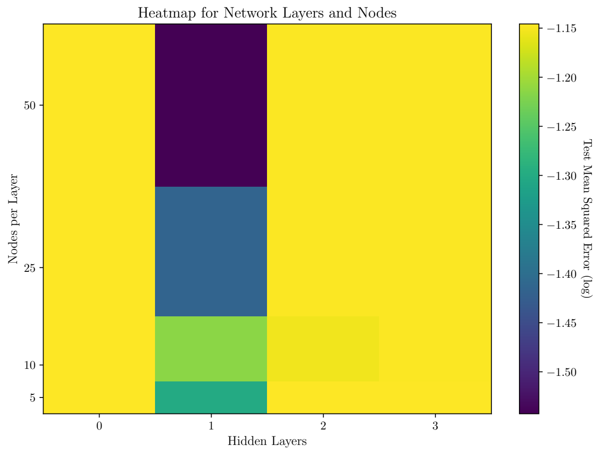

Figure 1: Heatmap for Runge's function with varying number of layers and nodes per layer. Iterations: 10000, Learning Rate: 0.05, N = 100

An okay plot:

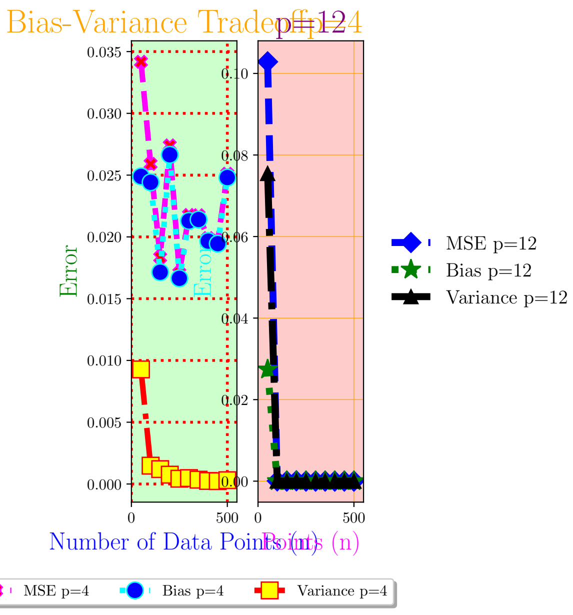

Figure 2: Bias-Variance tradeoff for polynomial degrees 4 and 12

Figure 2 is lacking as it has text upon text, different colors and styles for the lines and markers that make it hard to read. The points that are interesting is also hard to read as the markers are too small and the lines clutter the plot. The height of the graphs is also not chosen to show what is happening.

II. INTRODUCTION

The challenge of function approximation lies at the heart of machine learning, where the goal is to construct models that can accurately capture underlying patterns in data. In a previous study, we explored polynomial regression techniques, including Ordinary Least Squares (OLS), Ridge, and LASSO regression, to fit Runge's function, $f(x) = 1/(1 + 25x^2)$. While polynomial regression provides a foundational approach to function approximation, it often struggles with issues such as overfitting, especially with high-degree polynomials, leading to poor generalisation on unseen data. It also faces the extra work of finding the right model to fit the data on, something which a FFNN can adapt to by itself.

One technique that addresses these challenges is the use of FFNNs, which are capable of modelling complex, non-linear relationships through their layered architecture and non-linear activation functions. This method leverages multiple layers of interconnected nodes (neurons) to learn hierarchical representations of data, making them particularly well-suited for tasks that require capturing intricate patterns. It has been seen to have surprisingly good performance and adaptability to different problems. From large language models, image recognition and solving differential equations, to mention a few.

Because of these advantages, having a solid understanding of the underlying workings of an FFNN is crucial for effectively applying them to various machine learning tasks. But also to understand one of the dominant methods of data analysis today, which is finding different uses in many different areas.

This report documents the design and implementation of a flexible FFNN from scratch. The primary goal is to build and validate this model on two distinct tasks. First, we revisit the regression problem of fitting Runge's function, $f(x) = 1/(1 + 25x^2)$. This provides a direct benchmark against the performance of the OLS, Ridge, and LASSO models, allowing for a comparative analysis of their respective strengths in managing the bias-variance tradeoff on a familiar problem. We will investigate how network hyperparameters—such as the number of layers, nodes per layer, choice of activation function (Sigmoid vs. ReLU), and L_1/L_2 regularization—impact the network's ability to approximate the target function.

Second, the FFNN framework is extended to a multiclass classification task. By substituting the mean-squared error cost function with cross-entropy and the linear output layer with Softmax, the same underlying architecture is applied to the MNIST handwritten digit dataset. This part of the study will focus on optimizing the network for classification accuracy and comparing its performance against both a simpler logistic regression model and standard library implementations.

This report is structured as follows?

III. METHODS

This section details the theory and methodology used in this project to implement a flexible FFNN from scratch. We will first describe the data preparation process, followed by the fundamental building blocks of the FFNN, including the architecture, activation functions, cost functions, and optimization algorithms. Finally, we will discuss the implementation details and code structure. The primary dataset used for regression tasks is Runge's function,

$$f(x) = 1/(1 + 25x^2),$$

which is notoriously difficult for high-degree polynomial interpolation. Which we will compare to our earlier results using OLS, Ridge and LASSO regression. For more information on these methods and the results from them, see [1].

For more information on the theory behind FFNNs see the lecture notes [2], or the books [3] and [4].

A. Data Preparation

The fundamental data for this study was generated from Runge's function, $f(x) = 1/(1 + 25x^2)$, with x values uniformly distributed in the interval $[-1, 1]$. The primary study utilized a sparse dataset of $N = 18$ points to acutely demonstrate the high variance associated with overfitting.

Two versions of the dataset were created: a noiseless set, used for the main analysis, and a noisy set generated by adding Gaussian stochastic noise, $\epsilon \sim N(0, 0.1)$. The datasets were consistently split into training and test sets at an 80/20 ratio to evaluate generalization performance.

To ensure numerical stability and prevent coefficients corresponding to higher-degree polynomial terms from dominating the cost function, all features were scaled and centered by subtracting the mean value. This preprocessing step is critical, especially when dealing with regularization techniques, as it prevents the scale differences between polynomial features (e.g., x vs. x^{15}) from skewing the optimization process.

B. Fundamental Building Blocks

C. Implementation and Code

The code for this project was written in Python, utilizing libraries such as NumPy for numerical computations, Matplotlib for plotting, and Scikit-learn for some machine learning functionalities. There are two main files, utils.py and ResultGeneration.ipynb. The utils.py file contains functions for generating data, creating polynomial features, and implementing the analytical regression methods (OLS and Ridge) as well as the calculation of the gradient. Though much of the code for the different optimization algorithms are written directly in the ResultGeneration.ipynb file, under the section it belongs to. The ResultGeneration.ipynb file is a Jupyter notebook that is structured into different sections generating the results and plots for each part of this research paper. It is structured to allow modification of parameters such as the number of data points, polynomial degree, regularization strength, learning rate, and number of iterations. All results and plots are generated directly from this notebook with a fixed random seed ensuring reproducibility.

1. *Testing and Comparison*

To ensure correctness in the implementation of the metrics and the analytical solutions for OLS and Ridge regression, the results were cross-validated against Scikit-learn's built-in functions. Here we saw that the results matched quite well for both the metrics MSE and R2 score, as well as the actual parameter values θ in the ridge and OLS methods. For the MSE and R2 score the results were identical, while the parameters θ matched up to a precision of 10^{-14}. See the GitHub repository [5] Code/ResultGeneration.ipynb for the comparison results.

When it comes to the gradient implementations, they were not directly compared to Scikit-learn, but the convergence behavior and final MSE values were monitored to ensure they aligned with theoretical expectations and the results from the analytical methods.

D. Use of AI tools

Ai usage

IV. RESULTS AND DISCUSSION

Results and discussion goes here.

V. FURTHER WORK

further work goes here.

VI. CONCLUSION

A conclusion

[1] A. N. Torgersen, *A study of regularization and resampling for high-degree polynomial regression on a sparse data set* (2025), URL `https://github.com/FellowFrank/Fys-STK4155/tree/main/Project%20Work/Project%202`.

[2] M. Hjorth-Jensen, *Computational Physics Lecture Notes 2015* (Department of Physics, University of Oslo, Norway, 2015), URL `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf`.

[3] T. Hastie, R.Tibshirani, and J.Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics* (Springer, New York, 2009), URL `https://link.springer.com/book/10.1007%2F978-0-387-84858-7`.

[4] K. B. Hein, *Data Analysis and Machine Learning: Using Neural networks to solve ODEs and PDEs* (Department of Informatics, University of Oslo, Norway, 2018), URL `https://compphysics.github.io/MachineLearning/doc/pub/odenn/html/._odenn-bs000.html`.

[5] A. N. Torgersen, *Project 2 - fys-stk4155* (2025), URL `https://github.com/FellowFrank/Fys-STK4155/tree/main/Project%20Work/Project%202`.