# Title

Anton Nicolay Torgersen
*University of Oslo*
(Dated: November 1, 2025)

Developing a neural network from scratch to perform regression and classification tasks to solve Runge's function and the MNIST dataset respectively. Comparing the results with analytical methods, scikit-learn implementations and earlier projects involving linear regression and logistic regression using gradient descent methods.

## I. EXERCISE 44

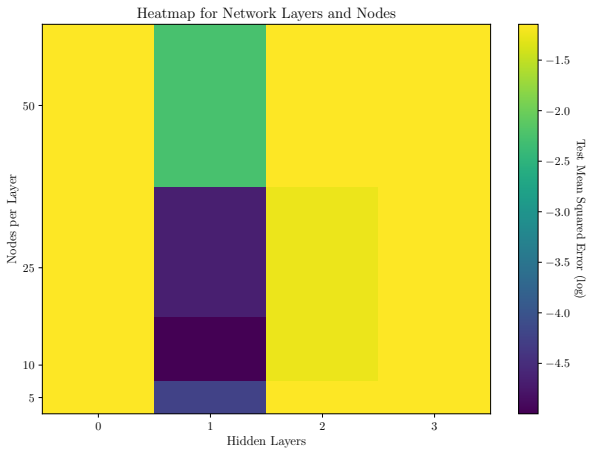


Figure 1: Heatmap for Runge's function with varying number of layers and nodes per layer. Iterations: 10000, Learning Rate: 0.05, N = 100

An okay plot:

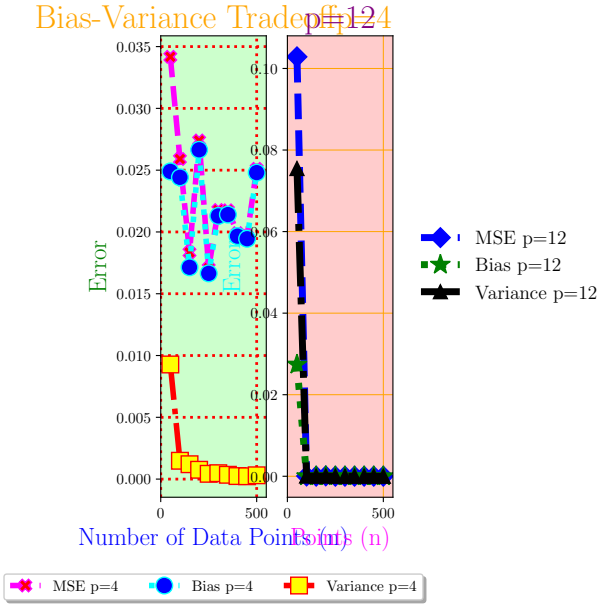


Figure 2: Bias-Variance tradeoff for polynomial degrees 4 and 12

## II. INTRODUCTION

The study of labled data through regression and classification techniques contains some presumptions about the underlying data structure. Which may not always hold true, especially when dealing with complex functions or high-dimensional data. Where one might want to use a more flexible model, such as a neural network, to capture the underlying patterns in the data. These methods are particularly useful when dealing with non-linear relationships or complex decision boundaries that traditional methods may struggle to model effectively. They are a flexible method where we say we dont know but hope that a sufficiently complex model can learn the underlying patterns in the data. This loose some of the interpretability we get from traditional methods, but can often lead to better performance on complex tasks.

## III. METHODS

This section details the regression, optimization, and resampling techniques employed to analyze Runge's function,

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

which is notoriously difficult for high-degree polynomial interpolation. We begin by describing data preparation, followed by the three regression methods and the gradient descent algorithms used for optimization.

### 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. Methods

### 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 $\theta$ in the ridge and OLS methods. For the MSE and R2 score the results were identical, while the parameters $\theta$ matched up to a precision of $10^{-14}$. See the GitHub repository [**?** ] 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