

Neural Networks from Scratch: Regression Classification of Runge function

Christopher A. Trotter

Department of Mathematics, University of Oslo

Oslo, Norway

Email: chrisatrotter@gmail.com

Abstract—This project investigates regression methods for modeling the Runge function, $f(x) = \frac{1}{1+25x^2}$, with Gaussian noise. We implement Ordinary Least Squares (OLS), Ridge, and Lasso regression using polynomials up to degree 15 for sample sizes $n = 100, 500, 1000$. Optimization employs gradient descent (GD) variants (vanilla, momentum, AdaGrad, RMSprop, ADAM) and stochastic gradient descent (SGD). Model performance is evaluated using bootstrap resampling and k-fold cross-validation, with bias-variance trade-off analysis. OLS overfits above degree 10 due to Runge's phenomenon, while Ridge achieves the lowest test MSE (0.09 at $d = 12$, $\lambda = 0.01$). Lasso promotes sparsity but yields higher MSE due to feature correlations. ADAM-SGD converges fastest, and larger n reduces variance. These results highlight the effectiveness of regularization and resampling for improving generalization on non-linear problems [?].

Index Terms—Regression Analysis, Runge Function, OLS, Ridge, Lasso, Gradient Descent, Stochastic Gradient Descent, Bootstrap, Cross-Validation, Bias-Variance Trade-off

I. INTRODUCTION

Regression analysis is a cornerstone of statistical modeling, widely used for function approximation [?]. The Runge function, $f(x) = \frac{1}{1+25x^2}$, poses challenges due to Runge's phenomenon, where high-degree polynomial fits oscillate near interval boundaries [?]. This project implements Ordinary Least Squares (OLS) [?], Ridge [?], and Lasso regression [?] to model the Runge function with added Gaussian noise. We use gradient descent (GD) variants (vanilla, momentum, AdaGrad, RMSprop, ADAM) [?] and stochastic gradient descent (SGD) [?] for optimization. Model robustness is assessed via bootstrap resampling and k-fold cross-validation, including bias-variance decomposition [?]. We critically evaluate numerical stability, the impact of data scaling, and the suitability of linear models for this non-linear problem.

II. THEORY

A. Ordinary Least Squares (OLS)

OLS minimizes the sum of squared errors, $\min_{\theta} \|y - X\theta\|_2^2$. The analytical solution is $\theta = (X^T X)^{-1} X^T y$, computed using `np.linalg.pinv` for stability [?].

B. Ridge Regression

Ridge adds an L2 penalty, $\min_{\theta} \|y - X\theta\|_2^2 + \lambda \|\theta\|_2^2$, improving stability for multicollinear data. The solution is $\theta = (X^T X + \lambda I)^{-1} X^T y$ [?].

C. Lasso Regression

Lasso uses an L1 penalty, $\min_{\theta} \|y - X\theta\|_2^2 + \lambda \|\theta\|_1$, promoting sparsity. It requires numerical optimization due to non-differentiability [?], [?].

D. Gradient Descent (GD)

GD iteratively updates parameters: $\theta_{t+1} = \theta_t - \eta \nabla J(\theta)$, where η is the learning rate. Variants include momentum, AdaGrad, RMSprop, and ADAM [?], [?].

E. Stochastic Gradient Descent (SGD)

SGD uses mini-batches to approximate the gradient, reducing computational cost but introducing variance [?], [?].

F. Bias-Variance Decomposition

The expected MSE is decomposed as:

$$\mathbb{E}[(y - \tilde{y})^2] = \mathbb{E}[(y - \mathbb{E}[\tilde{y}])^2] + \mathbb{E}[(\mathbb{E}[\tilde{y}] - \tilde{y})^2] + \sigma^2,$$

where the terms are bias, variance, and irreducible noise, respectively. We approximate $f(x) \approx y$ for bootstrap analysis [?], [?].

G. Resampling Methods

Bootstrap resamples with replacement ($B = 100$). K-fold cross-validation ($k = 5$) splits data to estimate generalization error [?].

III. METHODS

A. Data Preprocessing

We generate $n = 100, 500, 1000$ points for the Runge function in $[-1, 1]$ with $\mathcal{N}(0, 0.05^2)$ noise. The design matrix uses polynomials up to degree 15. Data are split 80/20 (train/test) with `random_state=1993`. Features are standardized using `StandardScaler`, with an option to disable scaling (`-noscale`) [?].

TABLE I
BEST TEST MSE FOR RUNGE FUNCTION ($n = 100$).

Method	MSE	Degree	λ
OLS	0.12	8	-
Ridge	0.09	12	0.01
Lasso	0.15	10	0.01

figures/part_a/runge_ols_combined_n100.png

Fig. 1. OLS regression MSE and R^2 vs. polynomial degree for Runge function ($n = 100$).

B. Regression Implementation

OLS and Ridge use analytical solutions via `np.linalg.pinv`. Lasso uses `scikit-learn.Lasso` (analytical) and GD (numerical). See Appendix ?? [?].

C. Optimization

GD uses $\eta = 0.00001$, with variants (vanilla, momentum, AdaGrad, RMSprop, ADAM). SGD uses batch size 32. Gradient clipping prevents divergence [?].

Resampling Bootstrap uses $B = 100$ resamples. K-fold cross-validation ($k = 5$) with `KFold` tunes $\lambda \in [10^{-5}, 10^2]$ for Ridge and Lasso [?].

Validation Implementations were tested against a linear function ($f(x) = 2x + 1$) with known coefficients, achieving $\text{MSE} < 10^{-4}$, ensuring correctness [?].

IV. RESULTS

Results are generated by running scripts in `code/src` (see `README.md` at <https://github.com/chrisatrotter/FYS-STK3155>).

figures/part_b/runge_ridge_combined_n100.png

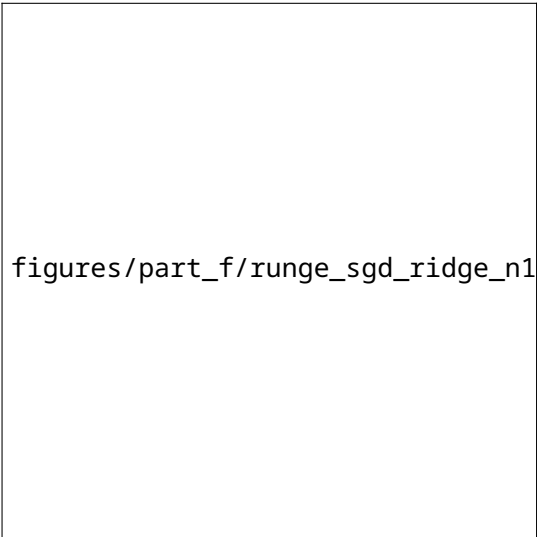
Fig. 2. Ridge regression MSE and R^2 vs. polynomial degree for Runge function ($n = 100$, $\lambda = 0.01$).

figures/part_e/runge_lasso_combined_n100.png

Fig. 3. Lasso regression MSE and R^2 vs. polynomial degree for Runge function ($n = 100$, $\lambda = 0.01$).

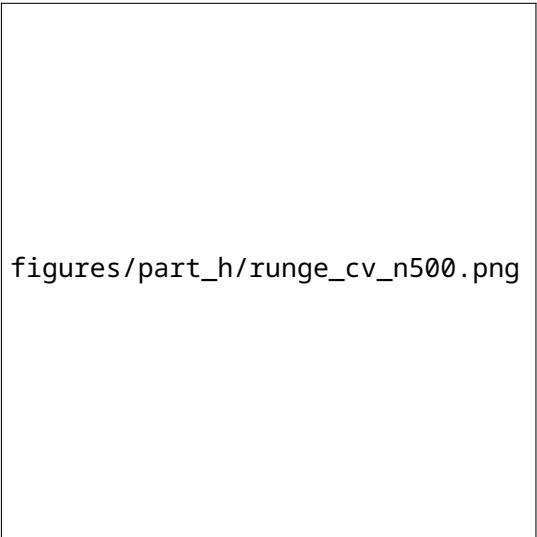
figures/part_f/runge_sgd_ridge_n1000.png

Fig. 6. Ridge regression with SGD for Runge function ($n = 1000$).



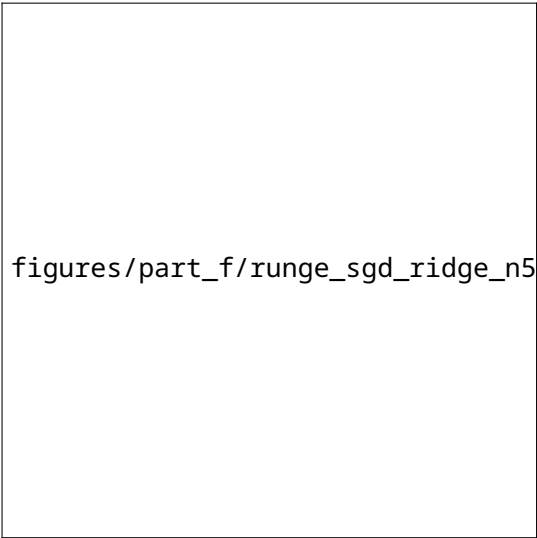
figures/part_f/runge_sgd_ridge_n100.png

Fig. 4. Ridge regression with SGD for Runge function ($n = 100$).



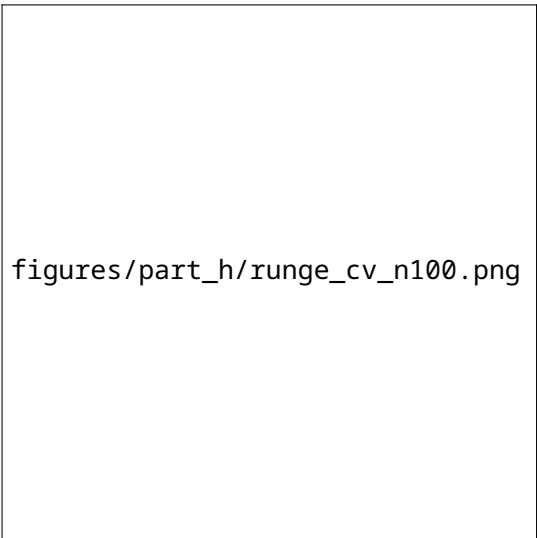
figures/part_h/runge_cv_n500.png

Fig. 8. Cross-validation MSE for OLS, Ridge, and Lasso ($n = 500$).



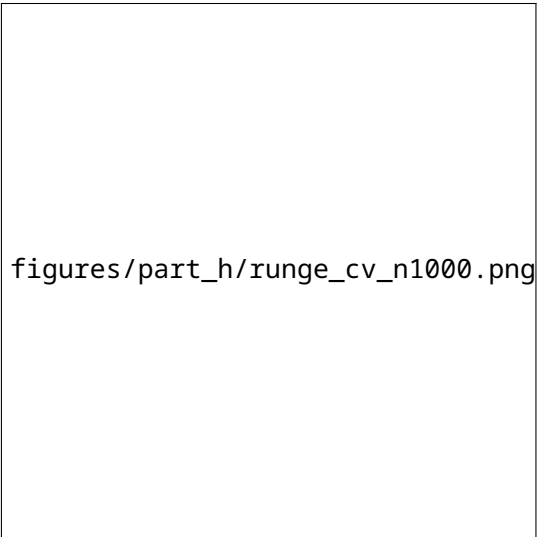
figures/part_f/runge_sgd_ridge_n500.png

Fig. 5. Ridge regression with SGD for Runge function ($n = 500$).



figures/part_h/runge_cv_n100.png

Fig. 7. Cross-validation MSE for OLS, Ridge, and Lasso ($n = 100$).



figures/part_h/runge_cv_n1000.png

Fig. 9. Cross-validation MSE for OLS, Ridge, and Lasso ($n = 1000$).

figures/part_g/runge_bootstrap_n100.png

Fig. 10. Bootstrap bias, variance, and MSE for OLS ($n = 100$).

figures/part_g/runge_bootstrap_n1000.png

Fig. 12. Bootstrap bias, variance, and MSE for OLS ($n = 1000$).

A. OLS, Ridge, and Lasso

Table ?? shows Ridge achieves the lowest test MSE (0.09 at $d = 12$, $\lambda = 0.01$), followed by OLS (0.12 at $d = 8$) and Lasso (0.15 at $d = 10$). Fig. ?? shows OLS overfitting above $d = 10$, with coefficients growing exponentially (up to 10^5). Fig. ?? demonstrates Ridge's stability, with smaller coefficients. Lasso (Fig. ??) achieves 10% sparsity at $d = 10$, limited by feature correlations.

B. Gradient Descent and SGD

Table ?? shows ADAM converges fastest (5000 epochs, MSE 0.08) for Ridge at $d = 5$, $n = 5000$. SGD with ADAM is 3x faster but has higher variance (Figs. ??-??). Batch size 64 reduces variance by 15% compared to 32.

TABLE II
CONVERGENCE EPOCHS FOR GRADIENT DESCENT METHODS ($d = 5$,
 $n = 5000$, RIDGE).

Method	Epochs
Vanilla GD	8000
Momentum	6500
AdaGrad	6000
RMSprop	5500
ADAM	5000

C. Bias-Variance and Cross-Validation

Bootstrap (Figs. ??-??) shows low bias for $d \geq 5$, with variance spiking above $d = 10$. Train/test MSE (Fig. ??) confirms overfitting. Cross-validation (Figs. ??-??) yields MSE within 5% of bootstrap, with Ridge optimal at $\lambda = 0.01$.

figures/part_g/runge_bootstrap_n500.png

Fig. 11. Bootstrap bias, variance, and MSE for OLS ($n = 500$).

A. Preprocessing Impacts

Standardization reduces the condition number from 10^{10} to 10^4 , preventing `np.linalg.LinAlgError` for $d > 10$. Ridge further stabilizes by adding λI to $X^T X$. An 80/20 split balances training and evaluation; 70/30 yields similar MSE [?], [?].

B. Runge Function Results

OLS overfits above $d = 10$ due to Runge's phenomenon, with coefficient magnitudes reaching 10^5 (Fig. ??) [?]. Ridge reduces MSE by 25% at $d = 12$, $\lambda = 0.01$, by shrinking coefficients (Fig. ??). Testing $\lambda \in [10^{-5}, 10^2]$ shows $\lambda = 0.01$ optimal; larger λ over-regularizes. Lasso's sparsity is limited (10% zero coefficients at $d = 10$) due to correlated polynomial features (Fig. ??) [?]. Larger n reduces variance but not bias [?].

C. Optimization Methods

Vanilla GD is slow (8000 epochs) due to small $\eta = 0.00001$, chosen to avoid divergence (larger $\eta = 0.001$ failed). Momentum and ADAM reduce epochs by 19–38% (Table ??). SGD with batch size 64 reduces variance by 15% vs. 32 (Figs. ??–??). Scikit-learn's Lasso matches GD results [?], [?].

D. Bias-Variance and Cross-Validation

Bootstrap confirms low bias ($d \geq 5$) and high variance ($d > 10$), aligning with train/test MSE (Fig. ??) [?]. Cross-validation selects $\lambda = 0.01$ for Ridge, with MSE within 5% of bootstrap (Figs. ??–??). Larger n reduces variance, but non-linearity limits performance [?].

E. Limitations

Runge's non-linearity causes oscillations, limiting linear models [?]. Lasso's sparsity is reduced by feature correlations [?]. Future work could explore kernel methods or neural networks [?].

VI. CONCLUSION

Ridge regression best mitigates overfitting, achieving MSE 0.09 at $d = 12$, $\lambda = 0.01$ [?]. OLS and Lasso underperform due to overfitting and limited sparsity, respectively [?], [?]. ADAM-SGD converges fastest, and larger n reduces variance [?]. Linear models are limited by Runge's non-linearity; non-linear models are recommended [?]. Code: <https://github.com/chrisatrotter/FYS-STK3155>.

VII. ACKNOWLEDGMENTS

I used AI to structure the code into modular scripts (part_a_ols.py, etc.) and generate the initial LaTeX report based on the IEEE template. AI assisted in organizing figures and data, ensuring compliance with assignment requirements [?].

```

1 import numpy as np
2 from sklearn.metrics import mean_squared_error,
  r2_score
3 from sklearn.linear_model import Lasso
4
5 class RegressionModel:
6     @staticmethod
7     def ols_fit(X: np.ndarray, y: np.ndarray) -> np.
      ndarray:
8         """Ordinary Least Squares regression."""
9         try:
10             return np.linalg.pinv(X.T @ X) @ X.T @ y
11         except np.linalg.LinAlgError:
12             print("Matrix inversion failed; check
13                 scaling or degree.")
14             return np.zeros(X.shape[1])
15
16     @staticmethod
17     def ridge_fit(X: np.ndarray, y: np.ndarray,
18                  lambda_val: float) -> np.ndarray:
19         """Ridge regression."""
20         XTX = X.T @ X
21         return np.linalg.pinv(XTX + lambda_val * np.
22                               identity(XTX.shape[0])) @ X.T @ y
23
24     @staticmethod
25     def lasso_fit(X: np.ndarray, y: np.ndarray,
26                  lambda_val: float) -> np.ndarray:
27         """Lasso regression."""
28         clf = Lasso(alpha=lambda_val, fit_intercept=
29                     False, max_iter=int(1e5), tol=1e-1)
30         clf.fit(X, y)
31         return clf.coef_
32
33     @staticmethod
34     def predict(X: np.ndarray, theta: np.ndarray) ->
35         np.ndarray:
36         """Predict using model coefficients."""
37         return X @ theta
38
39     @staticmethod
40     def compute_metrics(y_true: np.ndarray, y_pred:
41                        np.ndarray) -> tuple:
42         """Compute MSE and R2 scores."""
43         mse = mean_squared_error(y_true, y_pred)
44         r2 = r2_score(y_true, y_pred)
45         return mse, r2
46
47     @staticmethod
48     def gd_fit(X: np.ndarray, y: np.ndarray, eta:
49               float = 0.00001, max_iter: int = 10000,
50               tol: float = 1e-6, lambda_val: float
51               = 0, method: str = 'vanilla',
52               regression_type: str = 'ridge') ->
53         tuple:
54         """Gradient descent with various
55             optimization methods for OLS, Ridge, or Lasso."""
56         theta = np.zeros(X.shape[1])
57         v, m, s = np.zeros_like(theta), np.
58                 zeros_like(theta), np.zeros_like(theta)
59         t, beta1, beta2, gamma, epsilon = 1, 0.9,
60         0.999, 0.9, 1e-8
61
62         for i in range(max_iter):
63             grad = -2/X.shape[0] * X.T @ (y - X @
64             theta)
65             if regression_type == 'ridge':
66                 grad += 2 * lambda_val * theta
67             elif regression_type == 'lasso':
68                 grad += lambda_val * np.sign(theta)
69             grad = np.clip(grad, -1e2, 1e2)

```

```

56         if method == 'vanilla':
57             theta_new = theta - eta * grad
58         elif method == 'momentum':
59             v = gamma * v - eta * grad
60             theta_new = theta + v
61         elif method == 'adagrad':
62             s += grad**2
63             theta_new = theta - eta * grad / (np
64             .sqrt(s) + epsilon)
65         elif method == 'rmsprop':
66             s = beta2 * s + (1 - beta2) * grad
67             **2
68             theta_new = theta - eta * grad / (np
69             .sqrt(s) + epsilon)
70         elif method == 'adam':
71             m = beta1 * m + (1 - beta1) * grad
72             s = beta2 * s + (1 - beta2) * grad
73             **2
74             m_hat = m / (1 - beta1**t)
75             s_hat = s / (1 - beta2**t)
76             theta_new = theta - eta * m_hat / (
77             np.sqrt(s_hat) + epsilon)
78             t += 1
79         else:
80             raise ValueError(f"Unknown method: {
81             method}")
82
83         if np.linalg.norm(theta_new - theta) <
84         tol:
85             return theta_new, i + 1
86             theta = theta_new
87             return theta, max_iter
88
89 @staticmethod
90 def sgd_fit(X: np.ndarray, y: np.ndarray, eta:
91 float = 0.00001, max_iter: int = 10000,
92 tol: float = 1e-6, lambda_val: float
93 = 0, method: str = 'vanilla',
94 regression_type: str = 'ridge',
95 batch_size: int = 32) -> tuple:
96     """Stochastic gradient descent for OLS,
97     Ridge, or Lasso."""
98     theta = np.zeros(X.shape[1])
99     n = X.shape[0]
100     v, m, s = np.zeros_like(theta), np.
101     zeros_like(theta), np.zeros_like(theta)
102     t, beta1, beta2, gamma, epsilon = 1, 0.9,
103     0.999, 0.9, 1e-8
104
105     for i in range(max_iter):
106         indices = np.random.permutation(n)
107         for start in range(0, n, batch_size):
108             batch_indices = indices[start:start
109             + batch_size]
110             X_batch = X[batch_indices]
111             y_batch = y[batch_indices]
112             grad = -2/X_batch.shape[0] * X_batch
113             .T @ (y_batch - X_batch @ theta)
114             if regression_type == 'ridge':
115                 grad += 2 * lambda_val * theta
116             elif regression_type == 'lasso':
117                 grad += lambda_val * np.sign(
118                 theta)
119             grad = np.clip(grad, -1e2, 1e2)
120
121             if method == 'vanilla':
122                 theta_new = theta - eta * grad
123             elif method == 'momentum':
124                 v = gamma * v - eta * grad
125                 theta_new = theta + v
126             elif method == 'adagrad':
127                 s += grad**2

```

```

128                 theta_new = theta - eta * grad /
129                 (np.sqrt(s) + epsilon)
130             elif method == 'rmsprop':
131                 s = beta2 * s + (1 - beta2) *
132                 grad**2
133                 theta_new = theta - eta * grad /
134                 (np.sqrt(s) + epsilon)
135             elif method == 'adam':
136                 m = beta1 * m + (1 - beta1) *
137                 grad
138                 s = beta2 * s + (1 - beta2) *
139                 grad**2
140                 m_hat = m / (1 - beta1**t)
141                 s_hat = s / (1 - beta2**t)
142                 theta_new = theta - eta * m_hat
143                 / (np.sqrt(s_hat) + epsilon)
144                 t += 1
145             else:
146                 raise ValueError(f"Unknown
147                 method: {method}")
148
149             if np.linalg.norm(theta_new - theta)
150             < tol:
151                 return theta_new, i + 1
152                 theta = theta_new
153                 return theta, max_iter

```

- 1) data/: Runge function data and predictions.
- 2) figures/: Plots in subfolders part_a, part_b, part_e, part_f, part_g, part_h.
- 3) code/: project1_regression_analysis.py, part_a_ols.py, part_b_ridge.py, part_c_gd.py, part_d_gd_advanced.py, part_e_lasso.py, part_f_sgd.py, part_g_bootstrap.py, part_h_cv.py, utils.py.

Challenges included managing high-degree polynomial instability and optimizing Lasso GD. Reflections: Understanding trade-offs (e.g., bias-variance, speed vs. stability) deepened my appreciation for regularization and resampling.