Mini Neural Network Implementation Report

By: Mohammed Saad Alshutwi

1. Objective

The goal of this project was to implement a small neural network from scratch without using any machine learning libraries such as TensorFlow, Keras, or scikit-learn. The task involves predicting a continuous target y from two input features x and z.

2. Tools Used

- NumPy: for matrix operations and numerical computations
- pandas: for data handling and manipulation
- matplotlib: for visualizing training results
- Python built-in modules: random and math

3. Implementation Process

The project was divided into the following stages:

1. Data Generation:

- Synthetic dataset with 200 samples.
- y = 2x + 3 + noise, where x and z are random values.

2. Feature Engineering:

- Polynomial features: x^2, x^3, z^2, z^3
- Interactions and ratios: x*z, x/z
- Log, root, trigonometric, and noise-based features
- One-hot encoding for quartile bins of z

3. Data Splitting:

- 60% training, 20% validation, 20% test

4. Model Architecture:

- Input layer based on features
- One hidden layer with 5 neurons and ReLU activation
- One output neuron (regression)

5. Training Pipeline:

- Mini-batch gradient descent
- Momentum
- Learning rate decay when validation loss plateaus
- Early stopping
- L1 and L2 regularization

4. Challenges and How I Solved Them

During implementation, the following challenges were encountered:

- NaN in predictions:

This happened due to numerical instability in some features, especially division (x/z).

I fixed this using `np.clip` to restrict extreme values and adding checks in the training loop to catch NaNs.

- Exploding gradients:

Occurred when using high learning rates or small batch sizes. I resolved this by using smaller learning rates and batch sizes like 5.

- Empty validation loss list:

When the model stopped early due to NaNs, the loss lists were empty. I returned default values to prevent crashes.

5. Final Results

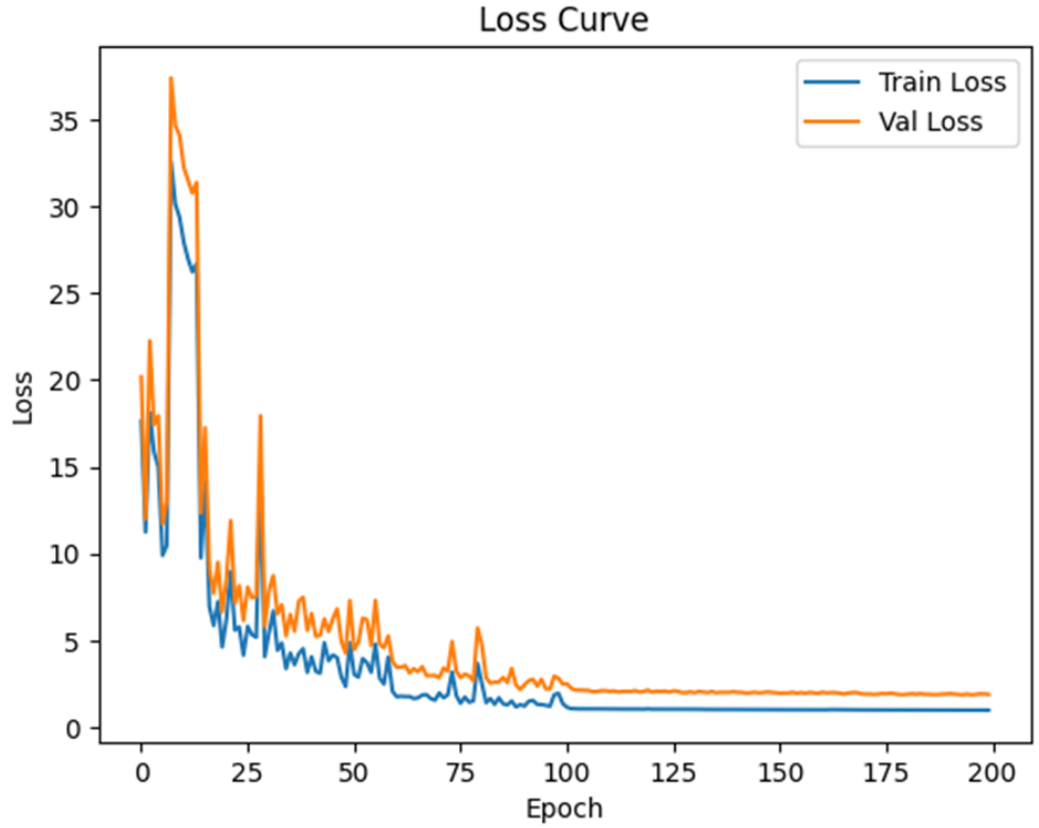
Best hyperparameter configuration:

- Learning Rate: 0.001
- Momentum: 0.5
- L2 Regularization: 0.01
- L1 Regularization: 0.01
- Batch Size: 5

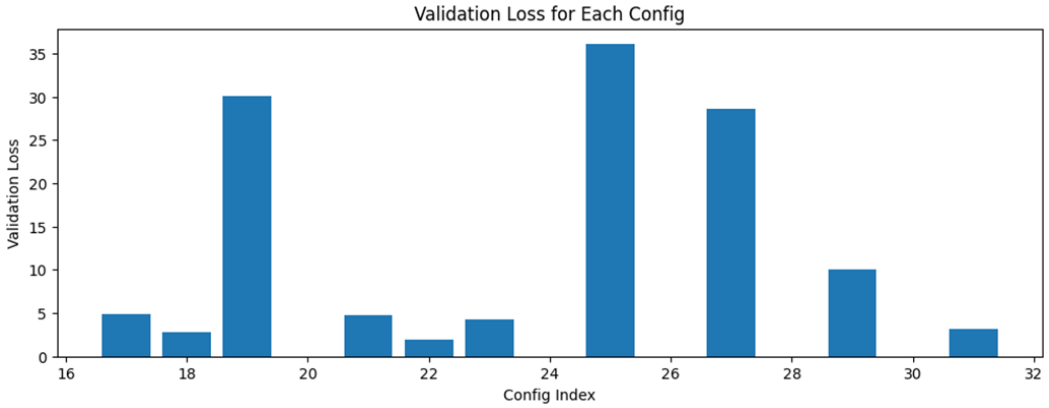
Evaluation metrics:

- Test MSE: 1.641
- Test MAE: 1.053
- Test RMSE: 1.281

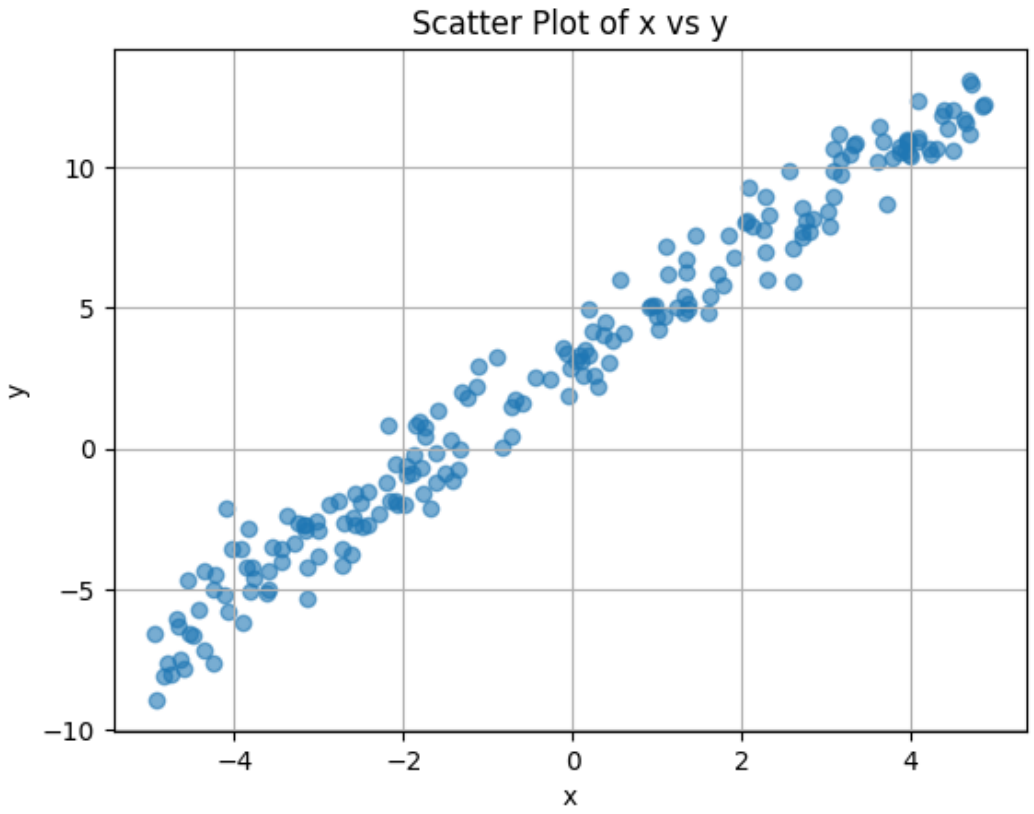
6. Visualizations



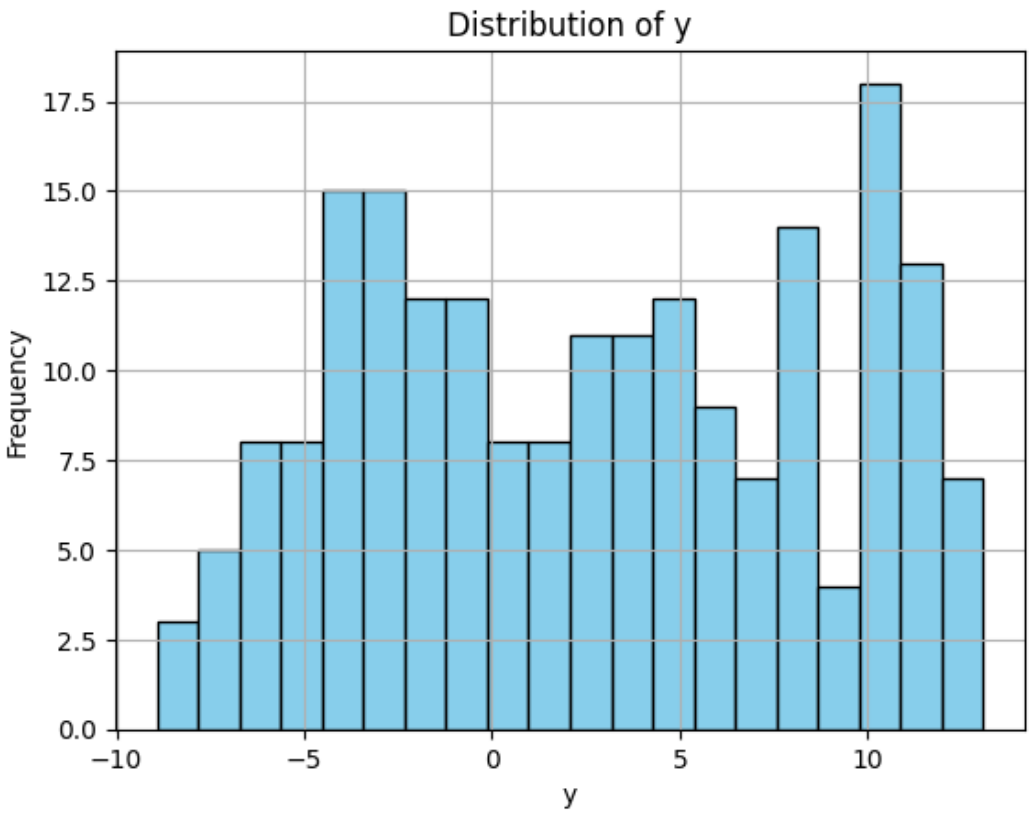
Loss Curve: This plot shows the training and validation loss over epochs. It helps to visualize if the model is learning and if overfitting is occurring.



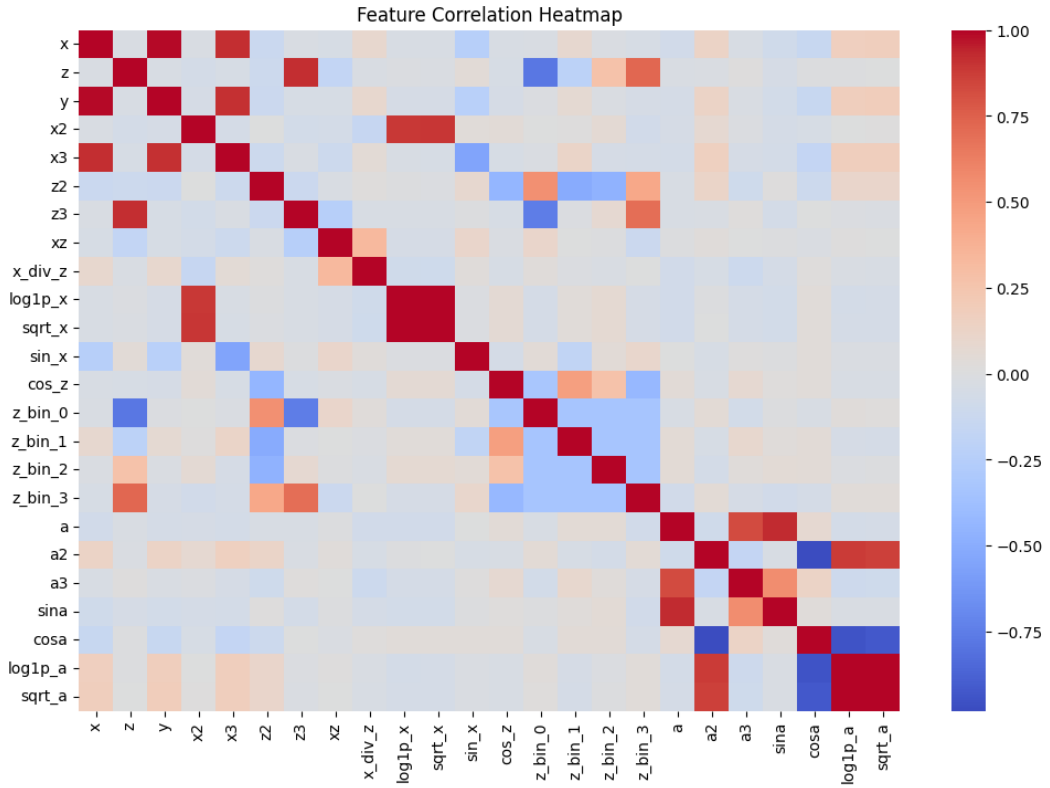
Validation Loss per Configuration: Each bar represents the validation loss for a different hyperparameter configuration. It helps identify the best configuration.



Scatter Plot of x vs y: This plot shows the relationship between the synthetic feature x and the target y. It indicates a strong linear trend.



Distribution of y: A histogram showing how the target variable y is distributed, helping to identify skewness or outliers.



Feature Correlation Heatmap: This heatmap shows the correlation between all features in the dataset, useful for feature selection and understanding multicollinearity.

7.Code Execution Screenshots

```
[35]            for l1 in [0, 0.01]:
                    for batch in  [5, len(X_train)]:
                        config = {'lr': lr, 'momentum': mom, 'l2': l2, 'l1': l1, 'batch_size': batch}
                        model, train_loss, val_loss = train_model(X_train, y_train, X_val, y_val, config)
                        final_val = val_loss[-1] if len(val_loss) > 0 else np.inf
                        configs.append(config)
                        results.append((model, train_loss, val_loss, final_val))
```

```
/tmp/ipython-input-33-3836382564.py:32: RuntimeWarning: overflow encountered in matmul
  dW2 = self.A1.T @ dZ2 + self.l2 * 2 * self.W2 + self.l1 * np.sign(self.W2)
/tmp/ipython-input-33-3836382564.py:34: RuntimeWarning: overflow encountered in matmul
  dA1 = dZ2 @ self.W2.T
/tmp/ipython-input-33-3836382564.py:36: RuntimeWarning: invalid value encountered in matmul
  dW1 = X.T @ dZ1 + self.l2 * 2 * self.W1 + self.l1 * np.sign(self.W1)
/tmp/ipython-input-33-3836382564.py:19: RuntimeWarning: invalid value encountered in matmul
  self.Z1 = X @ self.W1 + self.b1
/tmp/ipython-input-33-3836382564.py:25: RuntimeWarning: overflow encountered in square
  mse = np.mean((y_pred - y_true) ** 2)
/tmp/ipython-input-33-3836382564.py:21: RuntimeWarning: overflow encountered in matmul
  self.Z2 = self.A1 @ self.W2 + self.b2
/tmp/ipython-input-33-3836382564.py:32: RuntimeWarning: invalid value encountered in matmul
  dW2 = self.A1.T @ dZ2 + self.l2 * 2 * self.W2 + self.l1 * np.sign(self.W2)
/tmp/ipython-input-33-3836382564.py:35: RuntimeWarning: invalid value encountered in multiply
  dZ1 = dA1 * self.relu_deriv(self.Z1)
/tmp/ipython-input-33-3836382564.py:26: RuntimeWarning: overflow encountered in square
  reg = self.l2 * (np.sum(self.W1 ** 2) + np.sum(self.W2 ** 2)) +              self.l1 * (np.sum(np.abs(self.W1)) + np.sum(np.abs(self.W2)))
/tmp/ipython-input-33-3836382564.py:26: RuntimeWarning: invalid value encountered in scalar multiply
  reg = self.l2 * (np.sum(self.W1 ** 2) + np.sum(self.W2 ** 2)) +              self.l1 * (np.sum(np.abs(self.W1)) + np.sum(np.abs(self.W2)))
```

```
np.random.seed(42)
x = np.random.uniform(-5, 5, 200)
z = np.random.uniform(-5, 5, 200)
noise = np.random.normal(0, 1, 200)
y = 2 * x + 3 + noise
df = pd.DataFrame({'x': x, 'z': z, 'y': y})
df.head()
```

	x	z	y
0	-1.254599	1.420316	1.796281
1	4.507143	-4.158600	12.035290
2	2.319939	-3.383713	8.321832
3	0.986585	3.985542	4.662903
4	-3.439814	1.064291	-3.555461