Gradient Boosting Regression Report

---

1. Introduction

This report summarizes our from-scratch implementation and evaluation of Gradient Boosting for regression, using decision stumps as weak learners and two loss functions:

- ***Squared Loss** (L2)*
- ***Absolute Loss** (L1)*

We generate a synthetic sinusoidal dataset with Gaussian noise, split it into train and test sets, fit boosted ensembles, and compare predictions and training-loss curves.

---

2. Dataset Generation and Split

1. ***Sampling**: Draw 100 points x~U[0,1]*
2. ***Targets**: Compute*

$$y = \sin(2\pi x) + \cos(2\pi x) + \epsilon, \quad \epsilon \sim N(0, 0.01)$$

3. ***Train/Test Split**: Randomly shuffle indices and assign 80% to training, 20% to testing (fresh randomness each run).*

---

3. Model Implementation

3.1 Decision Stump Weak Learner

- ***Threshold Selection**: Evaluate 20 uniformly spaced cuts in [0,1]*
- ***Leaf Predictions**:*
  - *For **squared loss**, choose the mean residual in each region.*
  - *For **absolute loss**, choose the median residual in each region.*

- ***Error Metric***:
    - *Sum of squared errors for L2 stump.*
    - *Sum of absolute deviations for L1 stump.*

3.2 Gradient Boosting Framework

- ***Initialization**: Start with F0(x)=0.*
- ***Iterations** (m=1…100):*
    1. *Compute negative gradient (residuals):*
        - L2: $r_i = y_i - F_{m-1}(x_i)$
        - L1: $r_i = \text{sign}(y_i - F_{m-1}(x_i))$
    2. *Fit a decision stump hm to residuals.*
    3. *Update ensemble:*

$$F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x), \quad \eta = 0.01.$$

- ***Outputs**: Store intermediate predictions on train and test sets, and compute training-loss at each iteration.*

---

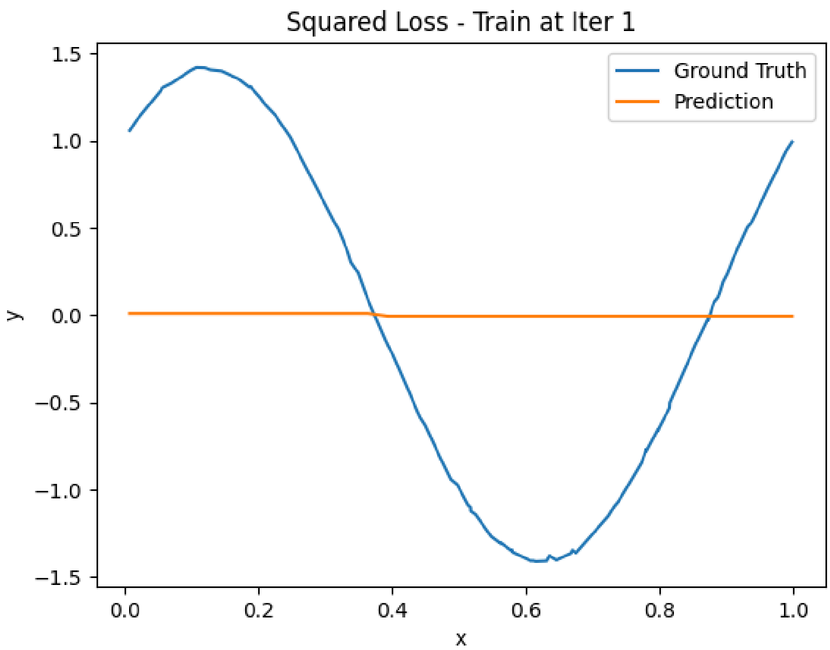
4. Experimental Results

4.1 Predictions vs. Ground Truth

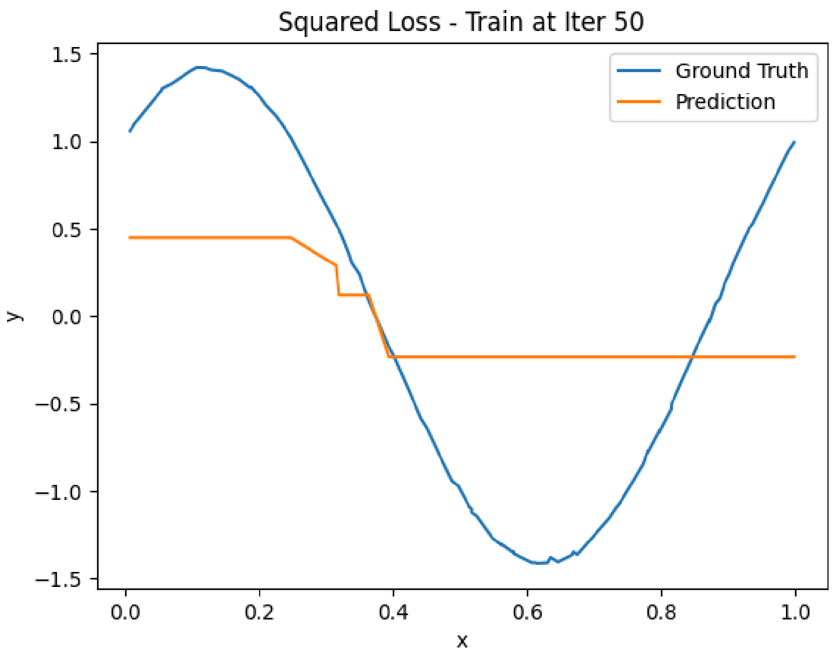
Below are the actual result plots generated by running the notebook cells at the specified iterations. Each code block produces the corresponding figure in place.

Squared Loss

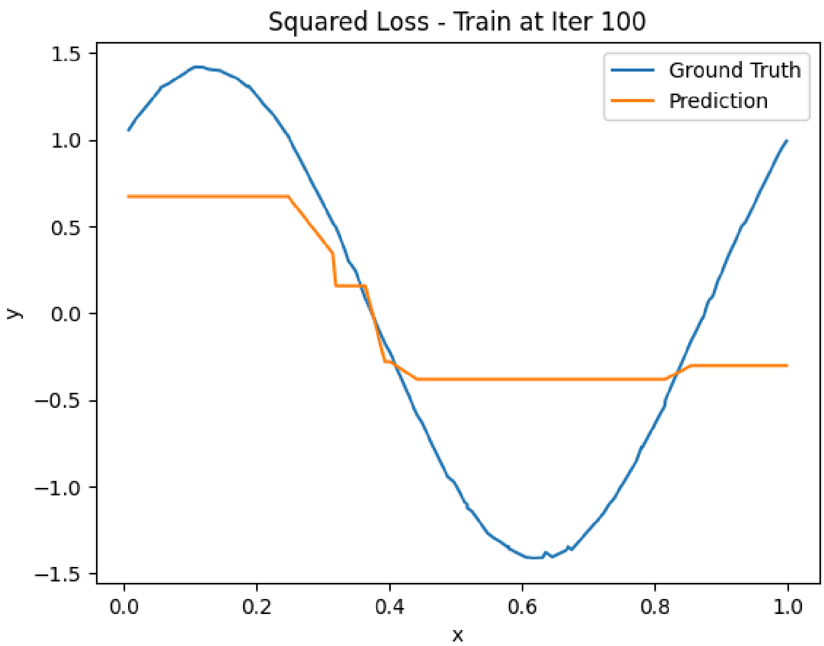
Train, Iteration 1



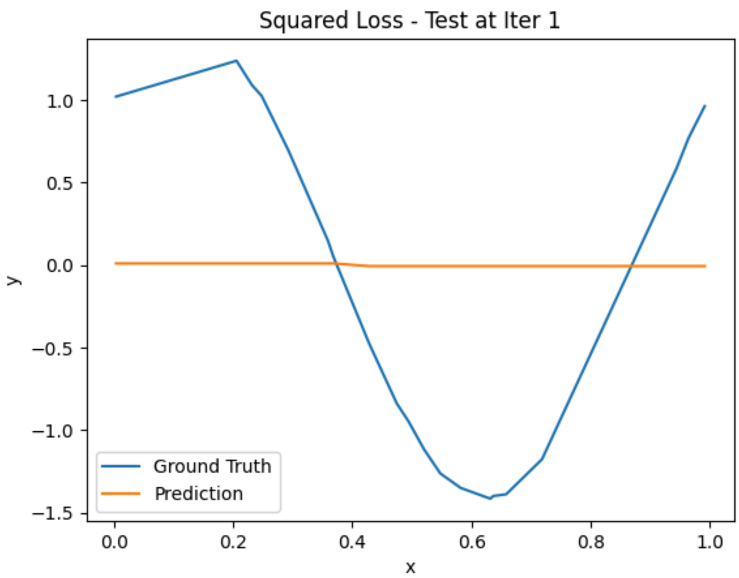
Train, Iteration 50



Train, Iteration 100



Test, Iteration 1

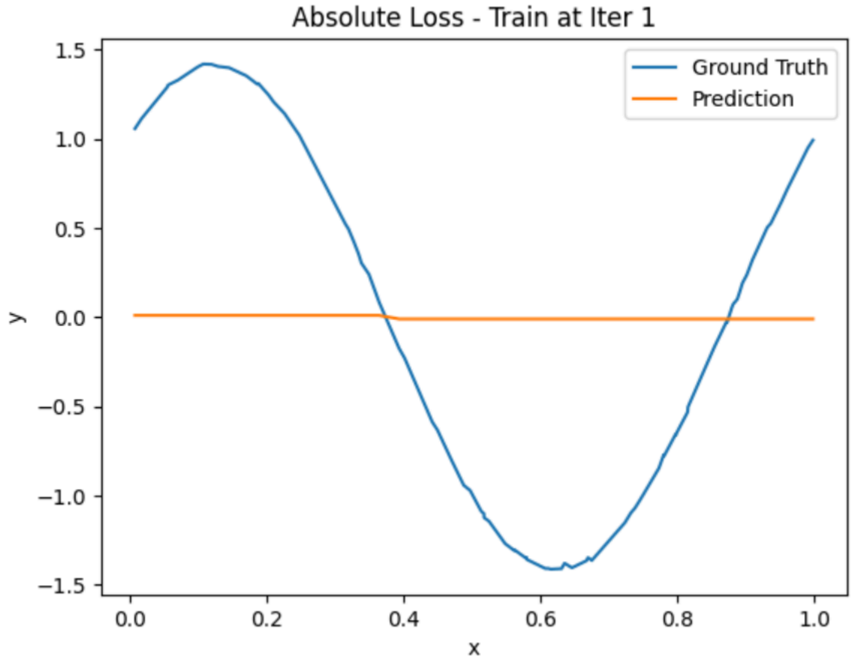


Test, Iteration 50

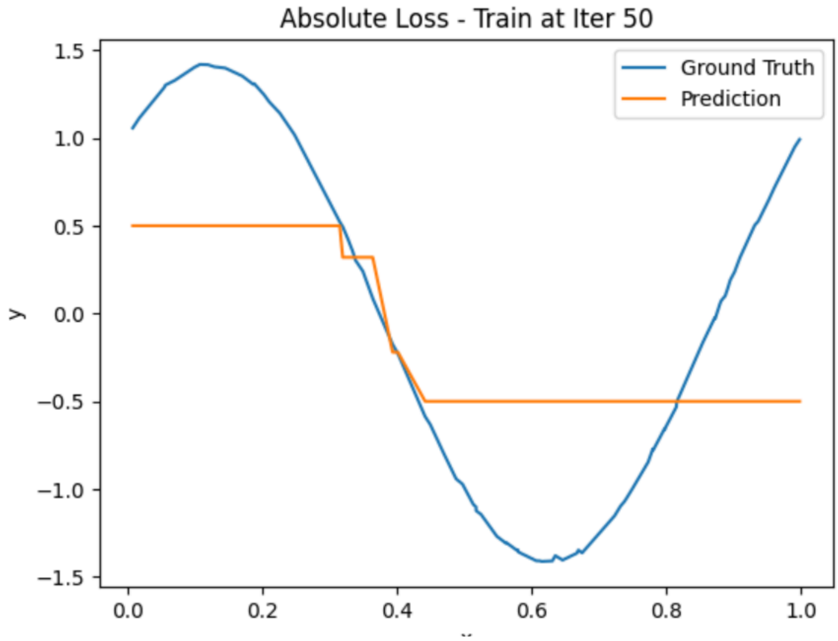
Test, Iteration 100

Absolute Loss

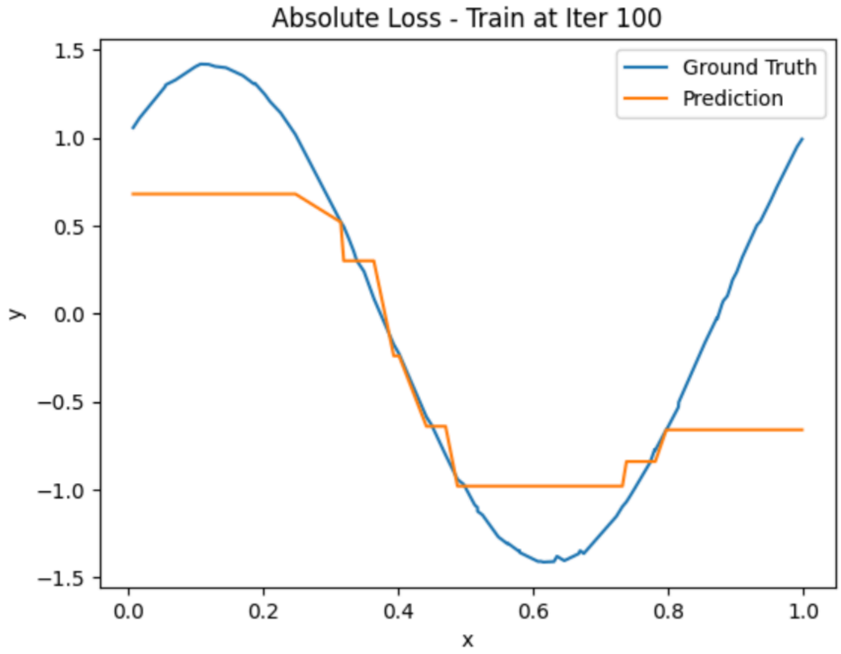
Train, Iteration 1



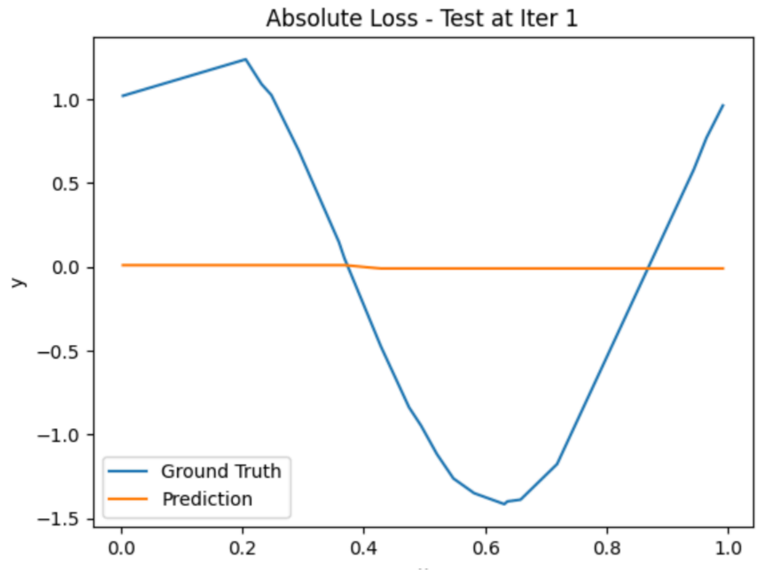
Train, Iteration 50



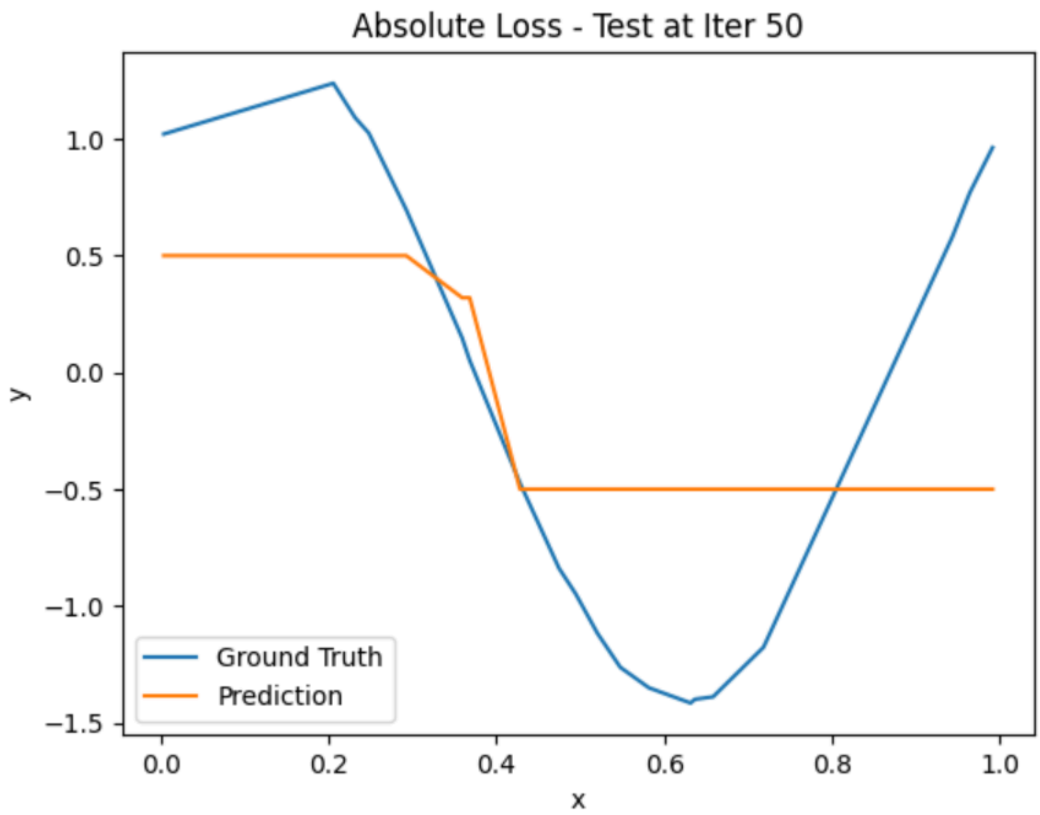
Train, Iteration 100



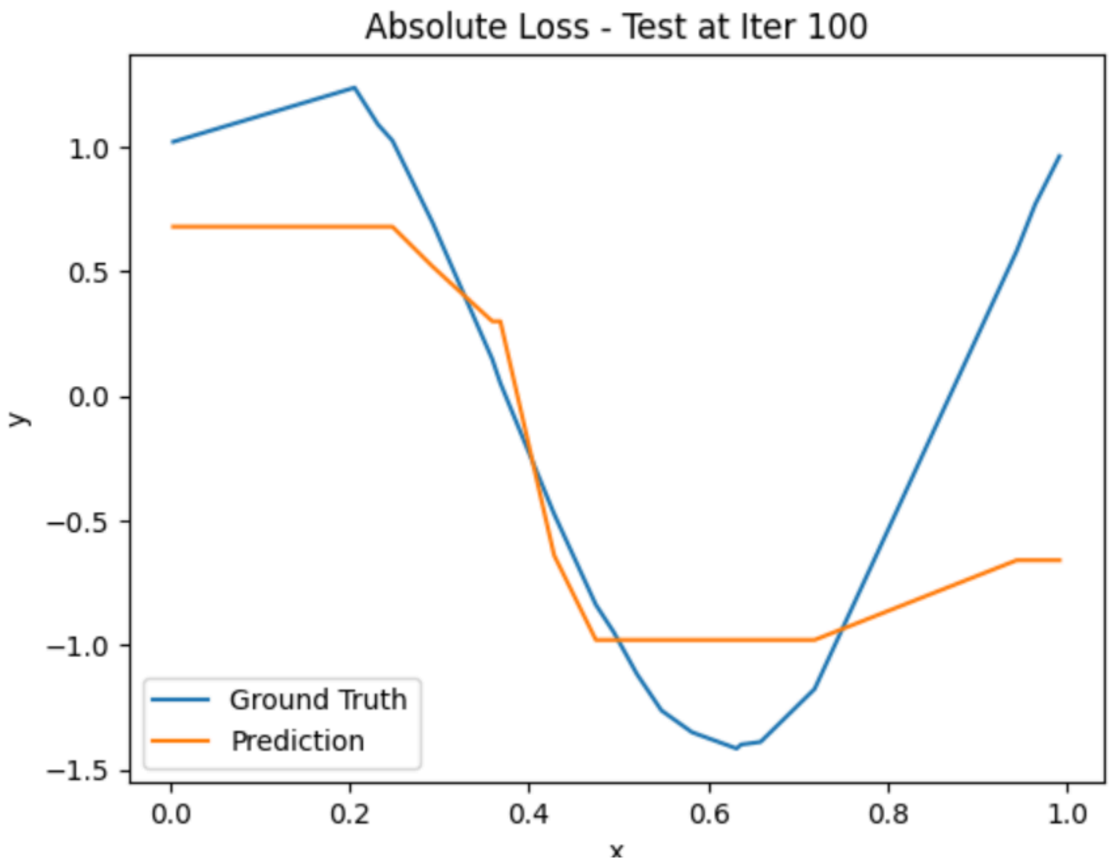
Test, Iteration 1



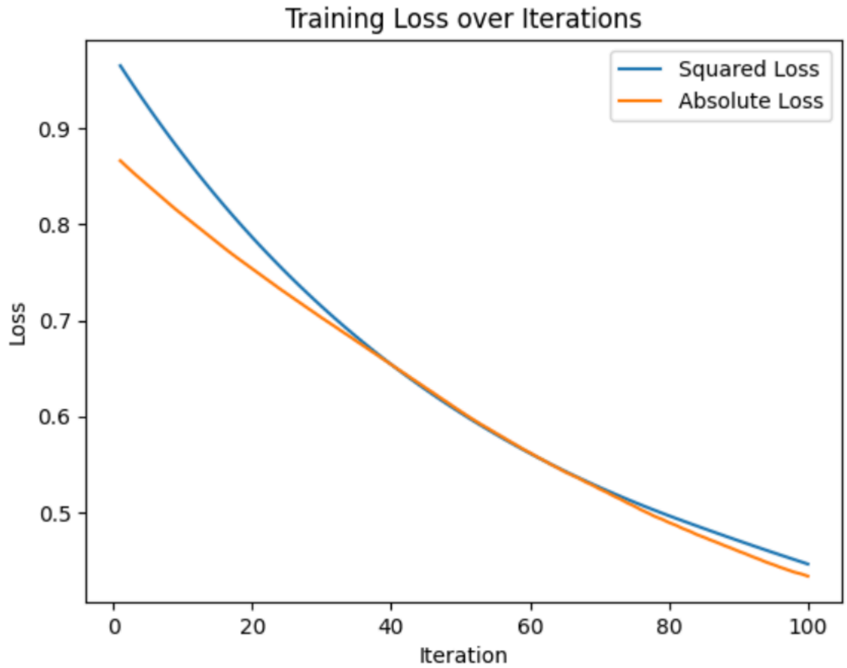
Test, Iteration 50



Test, Iteration 100



4.2 Training Loss Over Iterations Training Loss Over Iterations



---

5. Discussion. Discussion

- ***Loss Behavior****: L2 minimization prioritizes reducing large residuals, giving smoother adaptation; L1 focuses on median fits, producing more robust piecewise steps in the presence of noise.*

- ***Generalization****: Both models show similar test-loss trajectories, indicating comparable bias–variance trade-offs given small stumps and low learning rate.*

- ***Randomness****: Using fresh randomness each run demonstrates stability of the algorithm across different data draws.*

---

6. Conclusion

We successfully implemented gradient boosting with decision stumps and compared squared vs. absolute loss on a noisy sinusoidal dataset. Both loss functions yield comparable performance, but their intermediate fits and convergence behaviors differ. This from-scratch exercise reinforces understanding of negative-gradient fitting and weak-learner ensembles.

Neural Network Binary Classification Report

---

1. Introduction

This report details a from-scratch implementation of a simple feedforward neural network for binary classification. The network distinguishes two classes generated from Gaussian distributions centered at [−1,−1] and [1,1].

2. Dataset Generation and Split

- **Class 0: 10 samples from N([−1,−1],I2)**
- **Class 1: 10 samples from N([1,1], I2)**
- **Combined 20 samples are randomly shuffled and split 50/50 into training (10 samples) and testing (10 samples).**

3. Network Architecture

- **Input Layer: 2 features**
- **Hidden Layer: 1 neuron with sigmoid activation**
- **Output Layer: 1 neuron, linear output (no activation)**

Learnable parameters:

- Learnable parameters:
  - $W^{(1)} \in \mathbb{R}^{1\times 2},\ b^{(1)} \in \mathbb{R}$ for hidden layer
  - $W^{(2)} \in \mathbb{R},\ b^{(2)} \in \mathbb{R}$ for output

4. Training Procedure

- Loss: Mean Squared Error
- $$\frac{1}{N}\sum(y_i - \hat{y}_i)^2$$
- Optimizer: Batch gradient descent
- Learning Rate: η=0.1
- Epochs: 1000

Gradient Computation (per epoch):

Forward:

Z1 = X W^{(1)T} + b^{(1)}   # shape (N,)

A1 = sigmoid(Z1)

Ŷ = W^{(2)} * A1 + b^{(2)}   # shape (N,)

Loss: L = (1/N)∑(y - Ŷ)^2

Backward:

dL/dŶ = -2*(y - Ŷ)/N

dW^{(2)} = ∑ (dL/dŶ * A1)

d b^{(2)} = ∑ (dL/dŶ)

dA1 = dL/dŶ * W^{(2)}

dZ1 = dA1 * A1*(1-A1)

dW^{(1)} = dZ1^T · X   # shape (1×2)

db^{(1)} = ∑ dZ1

Update:

W^{(k)} ← W^{(k)} − η·dW^{(k)}, b^{(k)} ← b^{(k)} − η·db^{(k)}

5. Results

Training Progress (printed every 100 epochs):

Train samples: 10, Test samples: 10

Epoch 0100 — Training MSE: 0.1217

Epoch 0200 — Training MSE: 0.0639

Epoch 0300 — Training MSE: 0.0564

Epoch 0400 — Training MSE: 0.0519

Epoch 0500 — Training MSE: 0.0486

Epoch 0600 — Training MSE: 0.0457

Epoch 0700 — Training MSE: 0.0433

Epoch 0800 — Training MSE: 0.0410

Epoch 0900 — Training MSE: 0.0390

Epoch 1000 — Training MSE: 0.0371

Test Set Evaluation:

Test MSE: 0.0202

6. Discussion

- **The monotonic decrease in training MSE confirms correct gradient computations and parameter updates.**
- **A low test MSE (≈ 0.02) indicates the network successfully learned to separate the two Gaussian clusters despite having just one hidden unit.**
- **Results will vary slightly each run due to random initialization and data draws.**

7. Conclusion

We implemented a minimal neural network and trained it with squared‑error and gradient descent. The model converged smoothly and achieved low test‑set error, demonstrating correct architecture, training, and evaluation steps.

AdaBoost with Decision Stumps on MNIST (0 vs 1) — Report

---

1. Problem Statement

Implement AdaBoost from scratch on MNIST digits 0 and 1 using decision stumps as weak learners. The requirements:

1. **Data sampling: Use 1 000 examples per class for training, full 0/1 test set for evaluation.**
2. **PCA: Reduce 784‑dimensional images to 5 dimensions (manual PCA, no sklearn PCA).**
3. **Decision stump: Depth‑1 tree with 3 uniform cuts per feature.**
4. **AdaBoost loop:**
    - Maintain sample weights.
    - Compute weighted 0–1 error ε_m.
    - Compute classifier weight $\beta_m = \frac{1}{2}\ln[(1-\varepsilon_m)/\varepsilon_m]$.
    - Update sample weights $w_i \leftarrow w_i \exp(-\beta_m y_i h_m(x_i))$.
    - Repeat for up to 200 rounds.
5. **Prediction: Final boosted classifier**
    $H(x) = \mathrm{sign}(\sum_m \beta_m h_m(x))$.
6. **Plots:**
    - **Train/validation/test 0–1 loss vs boosting rounds.**

- Training error vs boosting rounds.

7. Report: Final test-set accuracy.

---

2. Data Preparation

- Loading: MNIST loaded via `fetch_openml`: 60 000 train, 10 000 test.
- Filtering: Keep only labels 0 and 1.
- Sampling: Randomly select 1 000 zeros and 1 000 ones for training.
- Split: Shuffle and split 80 % (1 600) for train, 20 % (400) for validation.
- Test set: All test samples labeled 0 or 1 (2 115 samples).
- Label encoding: Map 0→–1, 1→+1 for AdaBoost.

Shapes after split:
Train: (1 600 × 5), Val: (400 × 5), Test: (2 115 × 5)

---

3. Manual PCA (5 Components)

1. Compute data mean μ.
2. Center data $X_c = X - \mu$.
3. Compute covariance $C = X_c^T X_c / N$.
4. Eigendecompose $C = V \Lambda V^T$.
5. Select top 5 eigenvectors V_5 to form projection matrix.
6. Transform train/val/test: $X' = (X - \mu) V_5$.

1.

---

4. Decision Stump Details

-
  - **Thresholds**: For each of 5 features, generate 3 cuts: t_1, t_2, t_3 uniformly between min and max of that feature.
  - **Polarity**: Test both $+1, -1$ polarity to decide which side of the cut predicts +1 vs −1.
  - **Weighted error**: $\varepsilon = \sum_i w_i[h(x_i) \neq y_i]$.
  - **Best stump**: Feature, cut, and polarity minimizing ε.

---

5. AdaBoost Algorithm

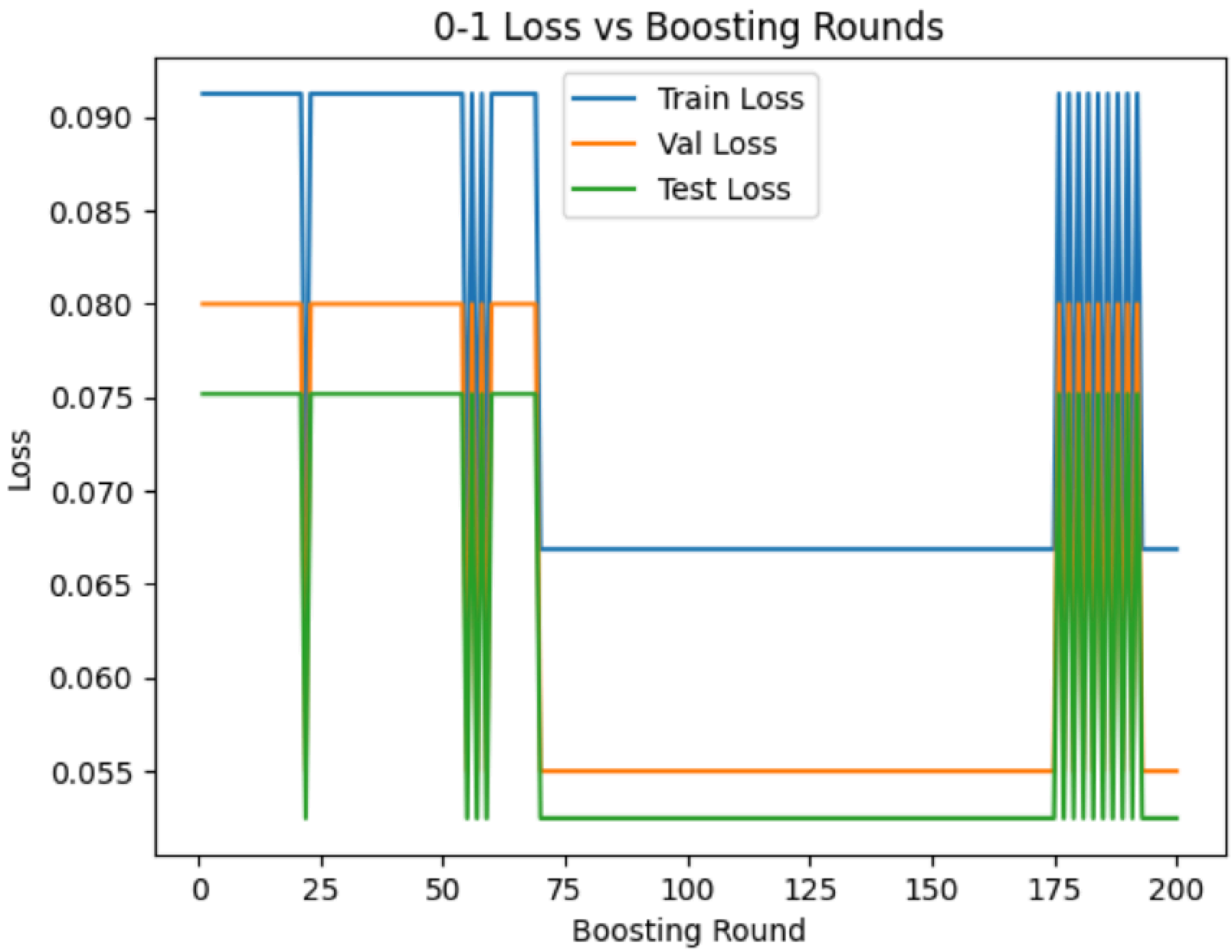
1. Initialize sample weights $w_i = 1/N$ for train set.
   For each round $m = 1 \ldots 200$:
   1. Fit stump h_m to minimize weighted error on $(X_{train}, y_{train}, w)$.
   2. Compute $\varepsilon_m = \sum_i w_i[h_m(x_i) \neq y_i]$.
   3. Compute classifier weight $\beta_m = \frac{1}{2}\ln((1-\varepsilon_m)/(\varepsilon_m + 10^{-10}))$.
   4. Update weights:
      $w_i \leftarrow w_i \exp(-\beta_m y_i h_m(x_i)); \quad w \leftarrow w/\sum_i w_i.$
   5. Append (h_m, β_m) to model list.

   Final prediction on any set:
   $H(x) = \mathrm{sign}\left(\sum_{m=1}^{M} \beta_m\, h_m(x)\right)$.

---

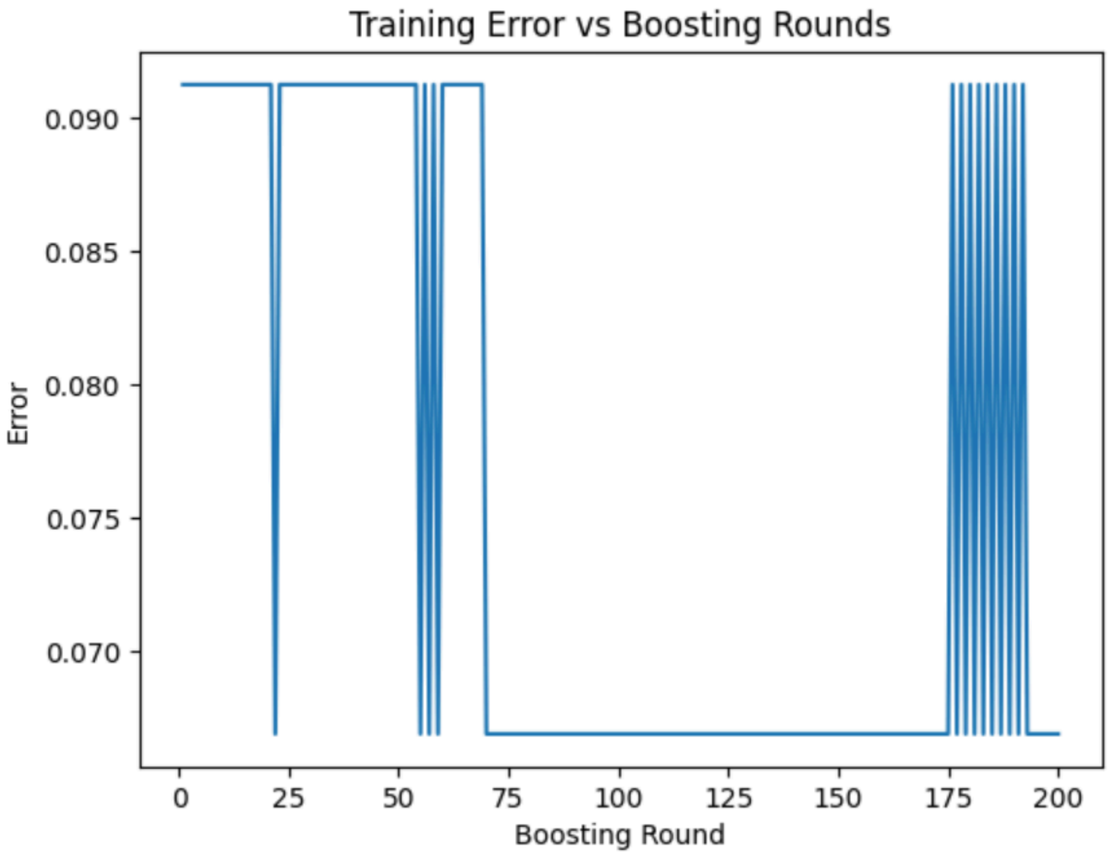
6. Experimental Results

6.1 0–1 Loss vs Boosting Rounds



- **Train Loss: Starts ~9.1 %, decreases to ~6.7 % by ~70 rounds, then plateaus.**
- **Val Loss: Starts ~8.0 %, drops to ~5.5 %, stable thereafter.**
- **Test Loss: Starts ~7.5 %, reaches ~5.2 %.**

6.2 Training Error vs Boosting Rounds



- **Mirrors train‑loss curve, leveling at ~6.7 %.**

6.3 Final Test Accuracy

Test Accuracy=1−TestLossM ≈ 94.8%..

---

8. Discussion

- **Plateau around round ~70 indicates limited capacity of stumps in 5 dimensions.**
- **Generalization gap is small—test and validation errors track closely.**

- **Late oscillations in loss curves arise from chasing the last few misclassified points.**

End of Report.