1.Loss Functions and Cost Functions for Regression

Overview

Loss functions and cost functions are crucial components in machine learning that measure the difference between predicted and actual values. This guide covers four fundamental functions used in regression problems: **Mean Squared Error (MSE)**, **Mean Absolute Error (MAE)**, **Root Mean Squared Error (RMSE)**, and **Huber Loss**.

Terminology Clarification

- **Loss Function**: Measures error for a **single training example** $L(y_i, \hat{y}_i)$
- **Cost Function**: Measures **average error across the entire dataset** $J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i)$
- **What we optimize**: Cost functions (the average) during training
- **What we compute per sample**: Loss functions (individual errors)

1. Mean Squared Error (MSE)

Loss Function (Single Sample)

$$L_{MSE}(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

Cost Function (Entire Dataset)

$$J_{MSE}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n} L_{MSE}(y_i, \hat{y}_i)$$

Where:

- y_i = true value for sample i
- $\hat{y}_i$ = predicted value for sample i
- n = number of samples

- θ = model parameters

Key Characteristics

Advantages:

- **Differentiable everywhere** - smooth gradient for optimization
- **Strongly penalizes large errors** due to squaring
- **Mathematically convenient** for analytical solutions
- **Widely supported** in ML frameworks

Disadvantages:

- **Sensitive to outliers** - large errors dominate the cost
- **Units are squared** - less interpretable than original scale
- **Can lead to overfitting** when outliers are present

Gradients for Backpropagation

Loss Function Gradient (per sample):

$$\frac{\partial L_{MSE}}{\partial \hat{y}_i} = 2(\hat{y}_i - y_i)$$

Cost Function Gradient (for optimization):

$$\frac{\partial J_{MSE}}{\partial \hat{y}_i} = \frac{2}{n}(\hat{y}_i - y_i)$$

When to Use MSE

- When large errors are significantly more costly than small ones
- Your data has roughly normal distribution of errors
- You need fast, stable optimization
- No significant outliers are present

2. Mean Absolute Error (MAE)

Loss Function (Single Sample)

$$L_{MAE}(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$$

Cost Function (Entire Dataset)

$$J_{MAE}(\theta) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n}L_{MAE}(y_i, \hat{y}_i)$$

Key Characteristics

Advantages:

- **Robust to outliers** - all errors weighted equally
- **Same units as target variable** - highly interpretable
- **Simple to understand** - average absolute deviation
- **Less sensitive to noise**

Disadvantages:

- **Not differentiable at zero** - gradient is undefined when error = 0
- **Constant gradient** - can slow convergence near optimum
- **May underfit** in presence of heteroscedastic noise

Gradients for Backpropagation

Loss Function Gradient (per sample):

$$\frac{\partial L_{MAE}}{\partial \hat{y}_i} = \text{sign}(\hat{y}_i - y_i) = \begin{cases} +1 & \text{if } \hat{y}_i > y_i \\ -1 & \text{if } \hat{y}_i < y_i \\ \text{undefined} & \text{if } \hat{y}_i = y_i \end{cases}$$

Cost Function Gradient (for optimization):

$$\frac{\partial J_{MAE}}{\partial \hat{y}_i} = \frac{1}{n} \times \text{sign}(\hat{y}_i - y_i)$$

When to Use MAE

- When outliers are present and should not dominate the cost
- When you need interpretable error metrics
- For problems where all errors should be treated equally
- When the underlying distribution has heavy tails

3. Root Mean Squared Error (RMSE)

Loss Function (Single Sample)

$$L_{RMSE}(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$$

(Note: Individual RMSE loss is just absolute error)

Cost Function (Entire Dataset)

$$J_{RMSE}(\theta) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} = \sqrt{J_{MSE}(\theta)}$$

Key Characteristics

Advantages:

- **Same units as target variable** - interpretable like MAE
- **Sensitive to large errors** like MSE
- **Widely used benchmark** - standard reporting metric
- **Balances interpretability and sensitivity**

Disadvantages:

- **Still sensitive to outliers** (inherited from MSE)
- **More complex gradient** than MSE
- **Computationally more expensive** due to square root

Gradients for Backpropagation

Loss Function Gradient (per sample):

$$\frac{\partial L_{RMSE}}{\partial \hat{y}_i} = \text{sign}(\hat{y}_i - y_i)$$

Cost Function Gradient (for optimization):

$$\frac{\partial J_{RMSE}}{\partial \hat{y}_i} = \frac{\hat{y}_i - y_i}{n \times J_{RMSE}}$$

When to Use RMSE

- For reporting and model comparison (interpretable units)
- When you want MSE behavior but interpretable scale
- In competitions and benchmarks (common standard)
- Note: Usually optimize MSE cost function, report with RMSE

4. Huber Loss

Loss Function (Single Sample)

$$L_{Huber}(y_i, \hat{y}_i, \delta) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta) & \text{if } |y_i - \hat{y}_i| > \delta \end{cases}$$

Cost Function (Entire Dataset)

$$J_{Huber}(\theta, \delta) = \frac{1}{n}\sum_{i=1}^{n} L_{Huber}(y_i, \hat{y}_i, \delta)$$

Where δ is the threshold parameter (typically $\delta = 1$).

Key Characteristics

Advantages:

- **Best of both worlds** - quadratic for small errors, linear for large ones
- **Robust to outliers** while maintaining smooth gradients

- **Tunable sensitivity** via δ parameter
- **Differentiable everywhere**

Disadvantages:

- **Additional hyperparameter** δ to tune
- **More complex to implement**
- **Less intuitive** than MSE/MAE

Gradients for Backpropagation

Loss Function Gradient (per sample):

$$\frac{\partial L_{Huber}}{\partial \hat{y}_i} = \begin{cases} (\hat{y}_i - y_i) & \text{if } |y_i - \hat{y}_i| \le \delta \\ \delta \times \text{sign}(\hat{y}_i - y_i) & \text{if } |y_i - \hat{y}_i| > \delta \end{cases}$$

Cost Function Gradient (for optimization):

$$\frac{\partial J_{Huber}}{\partial \hat{y}_i} = \frac{1}{n} \times \frac{\partial L_{Huber}}{\partial \hat{y}_i}$$

When to Use Huber Loss

- When you have both small and large errors to handle
- In robust regression with some outliers
- When you need smooth gradients but outlier resistance
- For time series with occasional anomalies

Numerical Comparison Example

Let's compare all four functions with a concrete example:

Setup:

- True values: $y = [3, 5, 2]$
- Predicted values: $\hat{y} = [2.5, 7, 1]$
- Errors: $e_i = \hat{y}_i - y_i = [-0.5, 2, -1]$
- Absolute errors: $|e_i| = [0.5, 2, 1]$

Individual Loss Function Values

For each sample i:

Sample	y_i	$\hat{y}_i$	e_i	L_{MSE}	L_{MAE}	L_{RMSE}	$L_{Huber}(\delta = 1)$
1	3	2.5	-0.5	$(-0.5)^2 = 0.25$	0.5	0.5	$\frac{1}{2}(0.5)^2 = 0.125$
2	5	7	2	$(2)^2 = 4$	2	2	$1(2 - 0.5) = 1.5$
3	2	1	-1	$(-1)^2 = 1$	1	1	$\frac{1}{2}(1)^2 = 0.5$

Cost Function Calculations

MSE Cost Function

$$J_{MSE} = \frac{1}{3}(0.25 + 4 + 1) = \frac{5.25}{3} = 1.75$$

MAE Cost Function

$$J_{MAE} = \frac{1}{3}(0.5 + 2 + 1) = \frac{3.5}{3} = 1.167$$

RMSE Cost Function

$$J_{RMSE} = \sqrt{1.75} = 1.323$$

Huber Cost Function ($\delta = 1$)

$$J_{Huber} = \frac{1}{3}(0.125 + 1.5 + 0.5) = \frac{2.125}{3} = 0.708$$

Cost Function Comparison

Cost Function	Final Value	Interpretation
MSE	**1.75**	Average squared error of 1.75 units²
MAE	**1.167**	Average absolute error of 1.17 units
RMSE	**1.323**	Typical error of 1.32 units
Huber ($\delta = 1$)	**0.708**	Robust cost balancing quadratic/linear penalties

Gradient Calculations

Individual Loss Function Gradients

For each sample i:

Sample	Error e_i	$\frac{\partial L_{MSE}}{\partial \hat{y}_i}$	$\frac{\partial L_{MAE}}{\partial \hat{y}_i}$	$\frac{\partial L_{RMSE}}{\partial \hat{y}_i}$	$\frac{\partial L_{Huber}}{\partial \hat{y}_i}$
1	-0.5	$2(-0.5) = -1$	$\text{sign}(-0.5) = -1$	$\text{sign}(-0.5) = -1$	-0.5 (quadratic region)
2	2	$2(2) = 4$	$\text{sign}(2) = +1$	$\text{sign}(2) = +1$	$1 \times \text{sign}(2) = +1$ (linear region)
3	-1	$2(-1) = -2$	$\text{sign}(-1) = -1$	$\text{sign}(-1) = -1$	-1 (boundary case)

Cost Function Gradients (Used in Optimization)

MSE Cost Gradients:

$$\frac{\partial J_{MSE}}{\partial \hat{y}_i} = \frac{2}{n} \times e_i$$

- $\frac{\partial J_{MSE}}{\partial \hat{y}_1} = \frac{2}{3} \times (-0.5) = -0.333$
- $\frac{\partial J_{MSE}}{\partial \hat{y}_2} = \frac{2}{3} \times (2) = +1.333$
- $\frac{\partial J_{MSE}}{\partial \hat{y}_3} = \frac{2}{3} \times (-1) = -0.667$

MAE Cost Gradients:

$$\frac{\partial J_{MAE}}{\partial \hat{y}_i} = \frac{1}{n} \times \text{sign}(e_i)$$

- $\frac{\partial J_{MAE}}{\partial \hat{y}_1} = \frac{1}{3} \times (-1) = -0.333$
- $\frac{\partial J_{MAE}}{\partial \hat{y}_2} = \frac{1}{3} \times (+1) = +0.333$
- $\frac{\partial J_{MAE}}{\partial \hat{y}_3} = \frac{1}{3} \times (-1) = -0.333$

RMSE Cost Gradients:

$$\frac{\partial J_{RMSE}}{\partial \hat{y}_i} = \frac{e_i}{n \times J_{RMSE}}$$

- $\frac{\partial J_{RMSE}}{\partial \hat{y}_1} = \frac{-0.5}{3 \times 1.323} = -0.126$
- $\frac{\partial J_{RMSE}}{\partial \hat{y}_2} = \frac{2}{3 \times 1.323} = +0.504$
- $\frac{\partial J_{RMSE}}{\partial \hat{y}_3} = \frac{-1}{3 \times 1.323} = -0.252$

Huber Cost Gradients:

$$\frac{\partial J_{Huber}}{\partial \hat{y}_i} = \frac{1}{n} \times \frac{\partial L_{Huber}}{\partial \hat{y}_i}$$

- $\frac{\partial J_{Huber}}{\partial \hat{y}_1} = \frac{1}{3} \times (-0.5) = -0.167$
- $\frac{\partial J_{Huber}}{\partial \hat{y}_2} = \frac{1}{3} \times (+1) = +0.333$
- $\frac{\partial J_{Huber}}{\partial \hat{y}_3} = \frac{1}{3} \times (-1) = -0.333$

Cost Function Gradient Comparison Summary

Cost Function	Gradient Vector	Key Characteristic
MSE	$[-0.333, +1.333, -0.667]$	**Proportional to error size** - large errors get big gradients
MAE	$[-0.333, +0.333, -0.333]$	**Constant magnitude**, only direction matters
RMSE	$[-0.126, +0.504, -0.252]$	**Scaled MSE gradients** - smaller magnitude due to normalization
Huber	$[-0.167, +0.333, -0.333]$	**Clipped for large errors** - combines quadratic and linear behavior

Decision Framework: Which Function to Choose?

Use MSE when:

- Large errors are significantly more costly than small ones
- Your data has roughly normal distribution of errors
- You need fast, stable optimization
- No significant outliers are present

Use MAE when:

- All errors should be penalized equally
- Your data contains outliers you want to ignore
- You need highly interpretable metrics
- The error distribution has heavy tails

Use RMSE when:

- You want MSE behavior but interpretable reporting
- Comparing models (industry standard)
- You need to communicate results to non-technical stakeholders
- Benchmark comparisons are important

Use Huber Loss when:

- Your data has mixed characteristics (some outliers, some normal errors)
- You need robust regression with smooth gradients
- You're willing to tune the δ parameter
- Working with time series or noisy real-world data

Implementation Notes

Training vs. Evaluation

- **Common practice**: Optimize using MSE cost function (simpler gradients), evaluate/report with RMSE (interpretable)
- **For outlier-prone data**: Optimize with Huber or MAE cost function, evaluate with multiple metrics

Loss vs Cost in Practice

- **Deep Learning**: Individual loss functions computed per sample, cost function is the batch average
- **Optimization**: Gradient descent minimizes the cost function (dataset average)
- **Reporting**: Often use interpretable metrics like RMSE or MAE

Hyperparameter Tuning

- **Huber Loss**: Start with $\delta = 1$, tune based on validation performance
- **Consider data scale**: Normalize features and targets for consistent δ values

Computational Considerations

- **Speed**: MSE > MAE > RMSE > Huber
- **Memory**: All have similar memory requirements
- **Numerical stability**: Be careful with RMSE when MSE approaches zero

Conclusion

Each loss/cost function serves specific purposes in regression problems. Understanding their mathematical properties, gradient behaviors, and appropriate use cases enables better model selection and improved performance. The key distinction between loss functions (per-sample) and cost functions (dataset average) is crucial for proper implementation and optimization.

Remember:

- **Loss functions** measure individual sample errors
- **Cost functions** are what we actually optimize (averages)
- **Gradients** of cost functions drive the learning process
- **Choice depends** on your data characteristics and business requirements

Loss Functions for Classification Problems

Introduction

Loss functions are essential components in machine learning that help optimizers minimize prediction errors during model training. For classification problems, we use specialized loss functions that differ from those used in regression tasks.

Types of Classification Problems

Classification problems can be categorized into two main types:

1. **Binary Classification**: Problems with only two possible output classes
2. **Multi-class Classification**: Problems with more than two possible output classes

Cross-Entropy Loss Functions

For classification problems, we primarily use **Cross-Entropy** as the loss function family. Based on the type of classification problem, we have three different variants:

1. Binary Cross-Entropy

Binary cross-entropy is specifically designed for **binary classification** problems.

Mathematical Formula:

$$\mathcal{L} = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

Where:

- y = actual value (ground truth)
- $\hat{y}$ = predicted value (model output)

Conditional Breakdown:

- **If $y = 0$:** $\mathcal{L} = -\log(1 - \hat{y})$
- **If $y = 1$:** $\mathcal{L} = -\log(\hat{y})$

How $\hat{y}$ is computed:

In binary classification, we apply the **sigmoid activation function** in the output layer:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Where $z = w_3 \cdot o_1 + b_2$ (weighted sum from previous layer)

2. Categorical Cross-Entropy

Categorical cross-entropy is used for **multi-class classification** problems where the output is one-hot encoded.

Mathematical Formula:

$$\mathcal{L}(x_i, y_i) = -\sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

Where:

- C = number of categories
- i = data point index (from 1 to n total data points)
- j = category index (from 1 to C categories)
- y_{ij} = 1 if element belongs to class j, 0 otherwise
- $\hat{y}_{ij}$ = predicted probability for class j

One-Hot Encoding Process:

The categorical cross-entropy first converts the output variable using one-hot encoding (OHE):

- If output is "good" → [1, 0, 0]
- If output is "bad" → [0, 1, 0]
- If output is "neutral" → [0, 0, 1]

How $\hat{y}_{ij}$ is computed:

For multi-class classification, we apply the **softmax activation function** in the output layer:

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}}$$

Output Characteristics:

- Provides probability distribution across all categories
- Sum of all probabilities equals 1
- Shows probability of belonging to each category

3. Sparse Categorical Cross-Entropy

Sparse categorical cross-entropy is also used for **multi-class classification** but with a key difference in output format.

Key Differences from Categorical Cross-Entropy:

Aspect	Categorical Cross-Entropy	Sparse Categorical Cross-Entropy
Output Format	Probability distribution	Single index (highest probability)
Probability Info	Shows probabilities for all categories	Only shows the winning category
Information Loss	No information loss	Loses probability information of other categories

Example Output Comparison:

For probabilities [0.2, 0.3, 0.5]:

- **Categorical**: Returns [0.2, 0.3, 0.5]
- **Sparse**: Returns index 2 (highest probability)

Advantage: Simpler output interpretation
Disadvantage: Loss of information about probability distribution of other categories

Activation Function and Loss Function Combinations

Combination 1: Binary Classification

```
Hidden Layers: ReLU activation
Output Layer: Sigmoid activation
Problem Type: Binary Classification
Loss Function: Binary Cross-Entropy
```

Combination 2: Multi-class Classification

```
Hidden Layers: ReLU activation
Output Layer: Softmax activation
Problem Type: Multi-class Classification
Loss Function: Categorical OR Sparse Categorical Cross-Entropy
```

Combination 3: Regression

```
Hidden Layers: ReLU (or its variants: Leaky ReLU, PReLU, ELU)
Output Layer: Linear activation
Problem Type: Regression
Loss Functions: MSE, MAE, Huber Loss, RMSE
```

When to Use Each Loss Function

Binary Cross-Entropy

- **Use when:** Solving binary classification problems
- **Output:** Single probability value between 0 and 1
- **Activation:** Sigmoid in output layer

Categorical Cross-Entropy

- **Use when:** You need probability information for all categories
- **Output:** Probability distribution across all classes
- **Activation:** Softmax in output layer
- **Best for:** When understanding confidence across all classes is important

Sparse Categorical Cross-Entropy

- **Use when:** You only need the final prediction (winning class)
- **Output:** Index of the class with highest probability
- **Activation:** Softmax in output layer
- **Best for:** When you don't need probability information for other classes

Important Interview Points

1. **Binary Cross-Entropy** uses the same log-loss formula as logistic regression
2. **Categorical Cross-Entropy** requires one-hot encoding of target variables
3. **Sparse Categorical Cross-Entropy** trades probability information for simplicity
4. The choice between categorical and sparse depends on whether you need probability distributions
5. All cross-entropy variants work with their respective activation functions (sigmoid for binary, softmax for multi-class)

Summary

Understanding loss functions is crucial for effective neural network training. The combination of appropriate activation functions in the output layer with corresponding loss functions ensures efficient optimization and model convergence. Choose your loss function based on:

- **Problem type** (binary vs. multi-class)
- **Information requirements** (probabilities vs. single prediction)
- **Computational efficiency needs**

The optimizer will use these loss functions to minimize prediction errors through forward and backward propagation until convergence is achieved.