Deep Learning for Perception

Lecture 02: Loss Functions & Gradient Descent

Optimization & Learning

Topics Covered in This Lecture:

- Loss Function vs Cost Function
- Mean Squared Error (MSE)
- Binary Cross-Entropy Loss
- Categorical Cross-Entropy Loss
- Gradient Descent Algorithm
- Batch vs Stochastic Gradient Descent
- Numerical Examples (Solved)

Instructor: Aqsa Younas

Department of Computer Science
FAST-NU, CFD Campus

Contents

Advance Organizer — What You'll Learn

Learning Objectives: By the end of this lecture, you will be able to:

1. **Distinguish** between loss functions and cost functions
2. **Calculate** MSE for regression problems step-by-step
3. **Apply** Binary Cross-Entropy for classification tasks
4. **Compute** Categorical Cross-Entropy for multi-class problems
5. **Implement** Gradient Descent to update weights
6. **Compare** Batch GD vs Stochastic GD with numerical examples

Prior Knowledge Required:

- Basic calculus (derivatives, partial derivatives)
- Perceptron model and weight update concept
- Logarithms and exponential functions

1 Introduction to Loss Functions

Why It Matters

Machines learn by minimizing a loss function. Without a way to measure "how wrong" the model is, there's no way to improve it. The loss function is the **compass** that guides the learning process.

1.1 What is a Loss Function?

Definition

A **Loss Function** (also called error function or objective function) measures how well a prediction model performs by quantifying the difference between **predicted values** and **actual values**.

- **Good predictions** $\rightarrow$ **Low loss**
- **Bad predictions** $\rightarrow$ **High loss**

Analogy — Think of It Like This

Think of a loss function as a **grading system** for your model. If a student (model) answers a question incorrectly, they lose points. The goal is to minimize the total points lost—this drives the student to learn better!

1.2 Loss Function vs Cost Function

Definition

Loss Function: Measures error for a **single training example**.

$$\text{Loss} = L(y, \hat{y}) \quad \text{(one sample)}$$

Cost Function: Measures **average error** over the **entire dataset**.

$$\text{Cost} = J = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i) \quad \text{(all samples)}$$

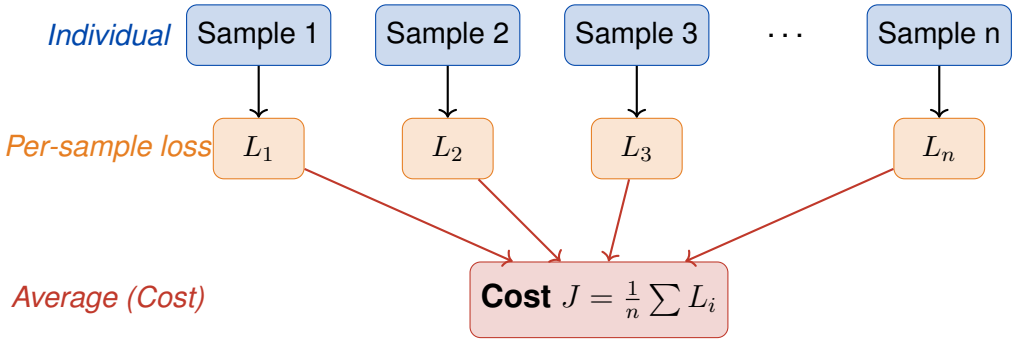


Figure 1: Loss (single sample) vs Cost (average over all samples)

Memory Hook — Remember This!

Quick Memory Aid:

- **Loss** = Error for **ONE** student's exam
- **Cost** = **Average** error for the **whole class**

1.3 Categories of Loss Functions

Category	Task Type	Common Loss Functions
Regression	Predict **continuous** values (price, temperature)	MSE, MAE, MSLE, Huber Loss
Classification	Predict **discrete** categories (spam/not spam)	Binary CE, Categorical CE, Hinge Loss

2 Mean Squared Error (MSE)

Why It Matters

MSE is the **most commonly used** loss function for regression problems. It penalizes larger errors more heavily due to the squaring operation, making it sensitive to outliers.

2.1 MSE Formula

Key Formula

Mean Squared Error (MSE):

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

Where:

- n = Number of samples
- y_i = Actual (true) value for sample i
- $\hat{y}_i$ = Predicted value for sample i
- $(y_i - \hat{y}_i)$ = Error (residual) for sample i

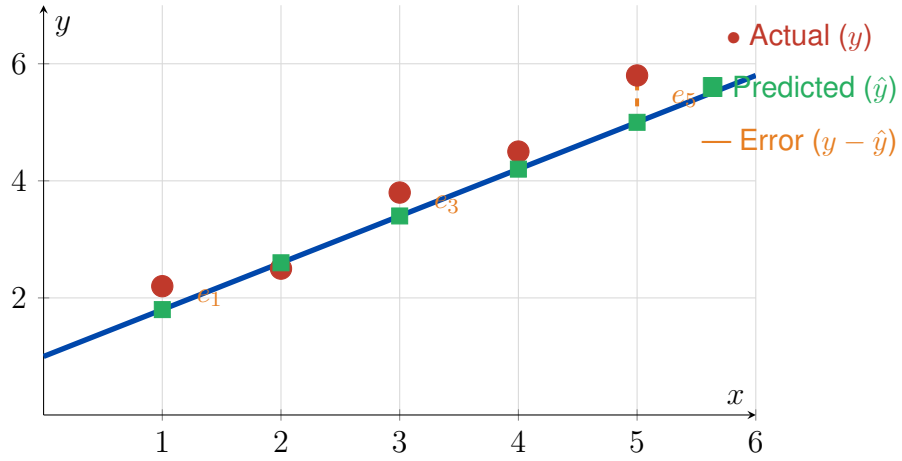


Figure 2: MSE measures the squared vertical distances between actual and predicted values

2.2 Step-by-Step MSE Calculation

Solved Example 1: MSE Calculation

Given Data:
A model predicts house prices (in $1000s). Calculate the MSE.

Sample	Actual Price (y)	Predicted Price ($\hat{y}$)
1	100	110
2	150	140
3	200	210
4	250	240
5	300	290

Solution:
Step 1: Calculate Error for each sample $(y_i - \hat{y}_i)$

$$\begin{aligned} e_1 &= 100 - 110 = -10 \\ e_2 &= 150 - 140 = +10 \\ e_3 &= 200 - 210 = -10 \\ e_4 &= 250 - 240 = +10 \\ e_5 &= 300 - 290 = +10 \end{aligned}$$

Step 2: Square each error $(y_i - \hat{y}_i)^2$

$$\begin{aligned} e_1^2 &= (-10)^2 = 100 \\ e_2^2 &= (10)^2 = 100 \\ e_3^2 &= (-10)^2 = 100 \\ e_4^2 &= (10)^2 = 100 \\ e_5^2 &= (10)^2 = 100 \end{aligned}$$

Step 3: Sum the squared errors

$$\sum_{i=1}^{5} (y_i - \hat{y}_i)^2 = 100 + 100 + 100 + 100 + 100 = 500$$

Step 4: Calculate the mean

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \frac{500}{5} = 100$$

Answer: MSE $= 100$ (in squared units of $1000s)
Interpretation: On average, the squared prediction error is 100. The Root MSE (RMSE) = $\sqrt{100} = 10$, meaning predictions are off by about $10,000 on average.

Memory Hook — Remember This!

Why Square the Errors?

1. Makes all errors **positive** (negative errors become positive)
2. **Penalizes large errors more** than small ones
3. Makes the function **differentiable** (smooth curve for optimization)

3 Binary Cross-Entropy Loss (BCE)

Why It Matters

For **binary classification** problems (two classes: 0 or 1), we can't use MSE effectively because outputs are probabilities. Binary Cross-Entropy measures how well predicted probabilities match actual binary labels.

3.1 BCE Formula

Key Formula

Binary Cross-Entropy (BCE):

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(p_i) + (1-y_i) \cdot \log(1-p_i)]$$

Where:

- N = Number of samples
- y_i = Actual label (0 or 1) for sample i
- p_i = Predicted probability of class 1 for sample i
- $\log$ = Natural logarithm (base e)

3.2 Understanding the BCE Formula

The formula has two parts:

- When $y = 1$: Loss $= -\log(p)$ (penalize if p is low)
- When $y = 0$: Loss $= -\log(1-p)$ (penalize if p is high)

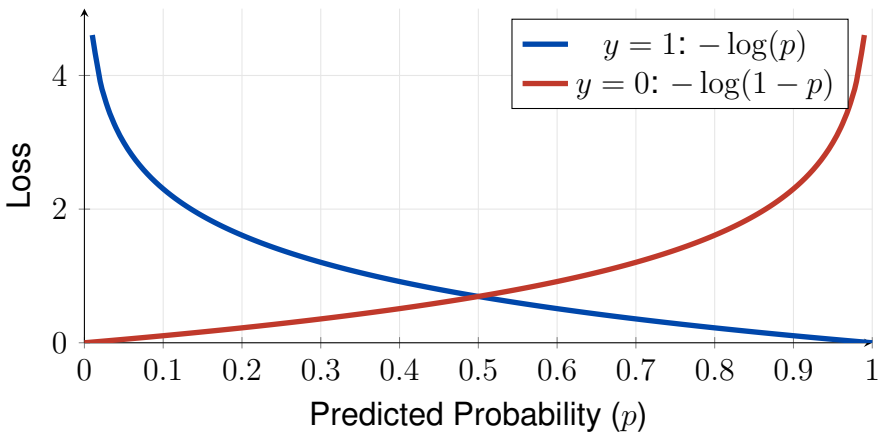


Figure 3: BCE Loss: When $y = 1$, loss decreases as $p \to 1$; When $y = 0$, loss decreases as $p \to 0$

Analogy — Think of It Like This

Think of BCE as a Confidence Penalty:

- If actual label is 1 and you predict 0.9 $\rightarrow$ Small penalty (confident and correct!)
- If actual label is 1 and you predict 0.1 $\rightarrow$ **Large penalty** (confident but WRONG!)

BCE punishes **confident wrong predictions** severely.

3.3 Step-by-Step BCE Calculation

Solved Example 2: Binary Cross-Entropy

Given Data:
A spam detection model outputs probabilities. Calculate the BCE loss.

Email	Actual (y)	Predicted Prob (p)	Meaning
1	1 (Spam)	0.9	90% sure it's spam
2	0 (Not spam)	0.2	20% sure it's spam
3	1 (Spam)	0.8	80% sure it's spam
4	1 (Spam)	0.7	70% sure it's spam
5	0 (Not spam)	0.1	10% sure it's spam

Solution:
Use formula: $L_i = -[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$
Sample 1: $y = 1, p = 0.9$

$$\begin{aligned} L_1 &= -[1 \cdot \log(0.9) + 0 \cdot \log(0.1)] \\ &= -[\log(0.9)] = -(-0.1054) = \mathbf{0.1054} \end{aligned}$$

Sample 2: $y = 0, p = 0.2$

$$\begin{aligned} L_2 &= -[0 \cdot \log(0.2) + 1 \cdot \log(0.8)] \\ &= -[\log(0.8)] = -(-0.2231) = \mathbf{0.2231} \end{aligned}$$

Sample 3: $y = 1, p = 0.8$

$$L_3 = -[\log(0.8)] = \mathbf{0.2231}$$

Sample 4: $y = 1, p = 0.7$

$$L_4 = -[\log(0.7)] = \mathbf{0.3567}$$

Sample 5: $y = 0, p = 0.1$

$$L_5 = -[\log(0.9)] = \mathbf{0.1054}$$

Average BCE:

$$\text{BCE} = \frac{0.1054 + 0.2231 + 0.2231 + 0.3567 + 0.1054}{5} = \frac{1.0137}{5} = \mathbf{0.2027}$$

Answer: BCE $= 0.2027$
Interpretation: A perfect model has BCE = 0. Our model has a relatively low loss, indicating good predictions. The highest individual loss was for Sample 4 (0.3567) where the model was only 70% confident about a spam email.

4 Categorical Cross-Entropy Loss (CCE)

Why It Matters

When you have **more than 2 classes** (multi-class classification), BCE won't work. Categorical Cross-Entropy extends the concept to handle multiple categories using **one-hot encoding**.

4.1 One-Hot Encoding

Definition

One-Hot Encoding: Represents categorical labels as binary vectors where only one element is 1 ("hot") and all others are 0.
Example: For 3 classes (Cat, Dog, Bird):

- Cat $\rightarrow$ [1, 0, 0]
- Dog $\rightarrow$ [0, 1, 0]
- Bird $\rightarrow$ [0, 0, 1]

4.2 CCE Formula

Key Formula

Categorical Cross-Entropy (CCE):

$$\boxed{L(y, \hat{y}) = -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i)}$$

Cost Function (Average over all samples):

$$\boxed{J = -\frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{C} y_{ji} \cdot \log(\hat{y}_{ji})}$$

Where:

- C = Number of classes
- y_i = True label (1 for correct class, 0 otherwise)
- $\hat{y}_i$ = Predicted probability for class i
- N = Number of samples

Memory Hook — Remember This!

Key Insight: Since $y_i = 0$ for all classes except the true class, the sum simplifies to:

$$\text{CCE} = -\log(\hat{y}_{\text{true class}})$$

Only the predicted probability for the **correct class** matters!

4.3 Step-by-Step CCE Calculation

Solved Example 3: Categorical Cross-Entropy

Given Data:
An image classifier predicts probabilities for 3 classes. Calculate CCE for one sample.

	Class 1 (Cat)	Class 2 (Dog)	Class 3 (Bird)
True Label (One-hot)	0	0	1
Predicted Probability	0.1	0.2	0.7

The image is actually a **Bird** (Class 3).

Solution:
Apply formula: CCE $= -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i)$
Calculate each term:

$$\begin{aligned}
\text{Class 1:} \quad & -y_1 \cdot \log(\hat{y}_1) = -0 \times \log(0.1) = 0 \\
\text{Class 2:} \quad & -y_2 \cdot \log(\hat{y}_2) = -0 \times \log(0.2) = 0 \\
\text{Class 3:} \quad & -y_3 \cdot \log(\hat{y}_3) = -1 \times \log(0.7) = -(-0.3567) = 0.3567
\end{aligned}$$

Sum all terms:

$$\text{CCE} = 0 + 0 + 0.3567 = \mathbf{0.3567}$$

Answer: CCE $= 0.3567$
Interpretation: The model predicted 70% probability for Bird (correct class). If it had predicted 100%, the loss would be $-\log(1) = 0$ (perfect). The loss of 0.3567 indicates room for improvement.

Solved Example 4: CCE with Multiple Samples

Given Data:
Calculate the **average CCE** (Cost) for 3 samples.

Sample	True Label (One-hot)	Predicted Probabilities
1	[1, 0, 0] (Cat)	[0.8, 0.1, 0.1]
2	[0, 1, 0] (Dog)	[0.2, 0.7, 0.1]
3	[0, 0, 1] (Bird)	[0.1, 0.2, 0.7]

Solution:
For each sample, only the probability of the true class matters.
Sample 1: True class = Cat, $\hat{y}_{\text{cat}} = 0.8$

$$L_1 = -\log(0.8) = 0.2231$$

Sample 2: True class = Dog, $\hat{y}_{\text{dog}} = 0.7$

$$L_2 = -\log(0.7) = 0.3567$$

Sample 3: True class = Bird, $\hat{y}_{\text{bird}} = 0.7$

$$L_3 = -\log(0.7) = 0.3567$$

Average Cost:

$$J = \frac{L_1 + L_2 + L_3}{3} = \frac{0.2231 + 0.3567 + 0.3567}{3} = \frac{0.9365}{3} = \mathbf{0.3122}$$

Answer: Average CCE (Cost) $= 0.3122$

5 Gradient Descent

Why It Matters

How does the model actually **learn**? By adjusting weights to minimize the loss function. **Gradient Descent** is the algorithm that finds which direction to adjust weights and by how much.

5.1 The Concept

Definition

Gradient Descent is an iterative optimization algorithm that minimizes a function by:

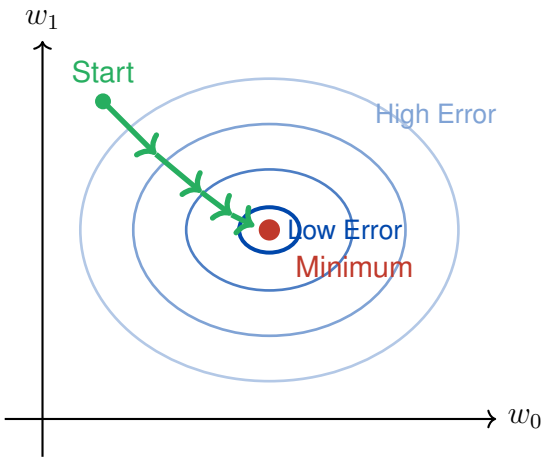
1. Starting with random initial weights
2. Computing the **gradient** (slope/direction of steepest increase)
3. Moving in the **opposite direction** (to decrease the error)
4. Repeating until convergence

Analogy — Think of It Like This

Imagine you're blindfolded on a hilly terrain trying to reach the lowest valley:

- You can feel the slope beneath your feet (gradient)
- You take a step in the direction that goes downhill (negative gradient)
- You keep walking until the ground feels flat (minimum reached)

The size of your steps is the **learning rate**.



Error Contour Map

Figure 4: Gradient Descent navigates the error surface toward the minimum

5.2 The Gradient

Definition

The **Gradient** $\nabla E(\mathbf{w})$ is a vector of partial derivatives that points in the direction of **steepest increase** of the error function.

$$\nabla E(\mathbf{w}) = \begin{bmatrix} \frac{\partial E}{\partial w_0} \\ \frac{\partial E}{\partial w_1} \\ \vdots \\ \frac{\partial E}{\partial w_n} \end{bmatrix}$$

5.3 Weight Update Rule

Key Formula

Gradient Descent Update Rule:

$$\boxed{\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla E(\mathbf{w})}$$

Component form:

$$\boxed{w_i \leftarrow w_i - \eta \cdot \frac{\partial E}{\partial w_i}}$$

Where:

- $\mathbf{w}$ = Weight vector
- η (eta) = **Learning rate** (step size)
- $\nabla E(\mathbf{w})$ = Gradient of error with respect to weights
- The **negative sign** moves us toward **decreasing** error

5.4 For Linear Unit with MSE

For a linear unit with MSE loss, the gradient simplifies to:

Key Formula

Delta Rule (Gradient for MSE):

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d) \cdot x_{id}$$

Weight Update:

$$\boxed{\Delta w_i = \eta \cdot \sum_{d \in D} (t_d - o_d) \cdot x_{id}}$$

Where:

- t_d = Target (actual) value for sample d
- o_d = Output (predicted) value for sample d
- x_{id} = Input i for sample d
- $(t_d - o_d)$ = Error for sample d

6 Batch vs Stochastic Gradient Descent

Why It Matters

There are two main ways to apply gradient descent: process all samples together (Batch) or one at a time (Stochastic). Each has trade-offs in speed, stability, and convergence.

6.1 Batch Gradient Descent (BGD)

Definition

Batch Gradient Descent:

1. Calculate error for **ALL** training samples
2. Sum up all weight updates (Δw)
3. Update weights **ONCE** at the end of the epoch

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) \cdot x_{id} \quad \text{(sum over ALL samples)}$$

6.2 Stochastic Gradient Descent (SGD)

Definition

Stochastic Gradient Descent:

1. For **EACH** training sample:
2. Calculate error for that sample
3. Update weights **IMMEDIATELY**

$$\Delta w_i = \eta (t_d - o_d) \cdot x_{id} \quad \text{(for EACH sample)}$$

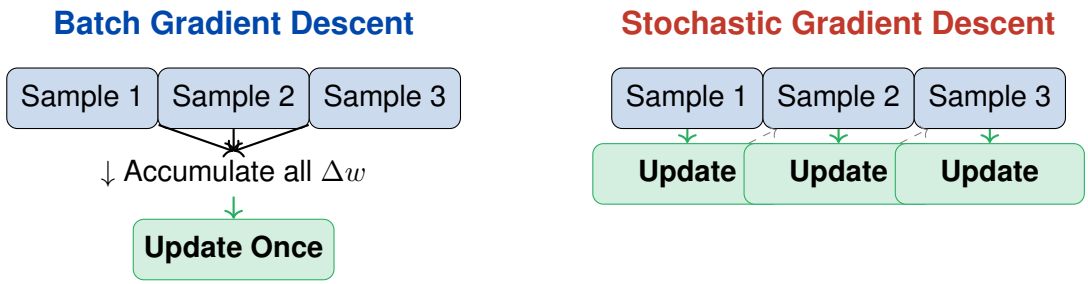


Figure 5: Batch GD updates once per epoch; SGD updates after every sample

Aspect	Batch GD	Stochastic GD
Update Frequency	Once per epoch	After every sample
Computation per Update	High (all samples)	Low (one sample)
Convergence Path	Smooth, direct	Noisy, zigzag
Memory Required	Higher	Lower
Local Minima	Can get stuck	Can escape (noise helps)
Learning Rate	Can use larger	Usually needs smaller

Table 1: Comparison of Batch GD vs Stochastic GD

7 Numerical Examples: Gradient Descent

Solved Example 5: Batch Gradient Descent (Complete Walkthrough)

Given Data:

- Learning rate: $\eta = 0.5$
- Initial weights: $w_0 = -0.3$ (bias), $w_1 = 0.5$, $w_2 = 0.5$
- Input format: $\mathbf{x} = [1, x_1, x_2]$ where first element is always 1 (for bias)

Training Data:

Sample	**Input** $\mathbf{x} = [1, x_1, x_2]$	**Target** t
1	[1, 1, 0]	1
2	[1, 1, 1]	0
3	[1, 0, 0]	0
4	[1, 0, 1]	1

Task: Perform ONE epoch of Batch Gradient Descent and find updated weights.

Solution:
Output formula: $o = w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2$
Update rule: $\Delta w_i = \eta \cdot (t - o) \cdot x_i$ (accumulate for all samples)

Step 1: Process Sample 1 — $\mathbf{x} = [1, 1, 0], t = 1$

$$o = (-0.3)(1) + (0.5)(1) + (0.5)(0) = -0.3 + 0.5 + 0 = \mathbf{0.2}$$
$$\text{Error} = t - o = 1 - 0.2 = \mathbf{0.8}$$
$$\Delta w_0 = 0.5 \times 0.8 \times 1 = \mathbf{0.4}$$
$$\Delta w_1 = 0.5 \times 0.8 \times 1 = \mathbf{0.4}$$
$$\Delta w_2 = 0.5 \times 0.8 \times 0 = \mathbf{0}$$

Step 2: Process Sample 2 — $\mathbf{x} = [1, 1, 1], t = 0$

$$o = (-0.3)(1) + (0.5)(1) + (0.5)(1) = -0.3 + 0.5 + 0.5 = \mathbf{0.7}$$
$$\text{Error} = t - o = 0 - 0.7 = \mathbf{-0.7}$$
$$\Delta w_0 = 0.5 \times (-0.7) \times 1 = \mathbf{-0.35}$$
$$\Delta w_1 = 0.5 \times (-0.7) \times 1 = \mathbf{-0.35}$$
$$\Delta w_2 = 0.5 \times (-0.7) \times 1 = \mathbf{-0.35}$$

Step 3: Process Sample 3 — $\mathbf{x} = [1, 0, 0], t = 0$

$$o = (-0.3)(1) + (0.5)(0) + (0.5)(0) = \mathbf{-0.3}$$
$$\text{Error} = t - o = 0 - (-0.3) = \mathbf{0.3}$$
$$\Delta w_0 = 0.5 \times 0.3 \times 1 = \mathbf{0.15}$$
$$\Delta w_1 = 0.5 \times 0.3 \times 0 = \mathbf{0}$$
$$\Delta w_2 = 0.5 \times 0.3 \times 0 = \mathbf{0}$$

Step 4: Process Sample 4 — $\mathbf{x} = [1, 0, 1], t = 1$

$$o = (-0.3)(1) + (0.5)(0) + (0.5)(1) = -0.3 + 0 + 0.5 = \mathbf{0.2}$$
$$\text{Error} = t - o = 1 - 0.2 = \mathbf{0.8}$$
$$\Delta w_0 = 0.5 \times 0.8 \times 1 = \mathbf{0.4}$$
$$\Delta w_1 = 0.5 \times 0.8 \times 0 = \mathbf{0}$$
$$\Delta w_2 = 0.5 \times 0.8 \times 1 = \mathbf{0.4}$$

Step 5: Accumulate ALL deltas (Batch GD)

$$\sum \Delta w_0 = 0.4 + (-0.35) + 0.15 + 0.4 = \mathbf{0.6}$$
$$\sum \Delta w_1 = 0.4 + (-0.35) + 0 + 0 = \mathbf{0.05}$$
$$\sum \Delta w_2 = 0 + (-0.35) + 0 + 0.4 = \mathbf{0.05}$$

Step 6: Update weights ONCE

$$w_0^{\text{new}} = w_0 + \sum \Delta w_0 = -0.3 + 0.6 = \mathbf{0.3}$$
$$w_1^{\text{new}} = w_1 + \sum \Delta w_1 = 0.5 + 0.05 = \mathbf{0.55}$$
$$w_2^{\text{new}} = w_2 + \sum \Delta w_2 = 0.5 + 0.05 = \mathbf{0.55}$$

Final Answer (After 1 Epoch of Batch GD):

$$\boxed{w_0 = 0.3, \quad w_1 = 0.55, \quad w_2 = 0.55}$$

Solved Example 6: Stochastic Gradient Descent (Complete Walkthrough)

Same Given Data as Example 5:

- Learning rate: $\eta = 0.5$
- Initial weights: $w_0 = -0.3$, $w_1 = 0.5$, $w_2 = 0.5$
- Same 4 training samples

Task: Perform ONE epoch of **Stochastic** Gradient Descent.
Key Difference: Update weights **AFTER EACH** sample!

Solution:
— Step 1: Sample 1 — $\mathbf{x} = [1, 1, 0], t = 1$
Current weights: $[w_0, w_1, w_2] = [-0.3, 0.5, 0.5]$

$$o = (-0.3)(1) + (0.5)(1) + (0.5)(0) = \mathbf{0.2}$$
$$\text{Error} = 1 - 0.2 = \mathbf{0.8}$$
$$\Delta w_0 = 0.5 \times 0.8 \times 1 = 0.4$$
$$\Delta w_1 = 0.5 \times 0.8 \times 1 = 0.4$$
$$\Delta w_2 = 0.5 \times 0.8 \times 0 = 0$$

Update immediately: $\boxed{[w_0, w_1, w_2] = [0.1, 0.9, 0.5]}$

— Step 2: Sample 2 — $\mathbf{x} = [1, 1, 1], t = 0$
Current weights: $[0.1, 0.9, 0.5]$ (updated from Step 1!)

$$o = (0.1)(1) + (0.9)(1) + (0.5)(1) = \mathbf{1.5}$$
$$\text{Error} = 0 - 1.5 = \mathbf{-1.5}$$
$$\Delta w_0 = 0.5 \times (-1.5) \times 1 = -0.75$$
$$\Delta w_1 = 0.5 \times (-1.5) \times 1 = -0.75$$
$$\Delta w_2 = 0.5 \times (-1.5) \times 1 = -0.75$$

Update immediately: $\boxed{[w_0, w_1, w_2] = [-0.65, 0.15, -0.25]}$

— Step 3: Sample 3 — $\mathbf{x} = [1, 0, 0], t = 0$
Current weights: $[-0.65, 0.15, -0.25]$

$$o = (-0.65)(1) + (0.15)(0) + (-0.25)(0) = \mathbf{-0.65}$$
$$\text{Error} = 0 - (-0.65) = \mathbf{0.65}$$
$$\Delta w_0 = 0.5 \times 0.65 \times 1 = 0.325$$
$$\Delta w_1 = 0.5 \times 0.65 \times 0 = 0$$
$$\Delta w_2 = 0.5 \times 0.65 \times 0 = 0$$

Update immediately: $\boxed{[w_0, w_1, w_2] = [-0.325, 0.15, -0.25]}$

— Step 4: Sample 4 — $\mathbf{x} = [1, 0, 1], t = 1$
Current weights: $[-0.325, 0.15, -0.25]$

$$o = (-0.325)(1) + (0.15)(0) + (-0.25)(1) = -0.325 - 0.25 = \mathbf{-0.575}$$
$$\text{Error} = 1 - (-0.575) = \mathbf{1.575}$$
$$\Delta w_0 = 0.5 \times 1.575 \times 1 = 0.7875$$
$$\Delta w_1 = 0.5 \times 1.575 \times 0 = 0$$
$$\Delta w_2 = 0.5 \times 1.575 \times 1 = 0.7875$$

Update immediately: $\boxed{[w_0, w_1, w_2] = [0.4625, 0.15, 0.5375]}$

Final Answer (After 1 Epoch of SGD):

$$\boxed{w_0 = 0.4625, \quad w_1 = 0.15, \quad w_2 = 0.5375}$$

Compare with Batch GD: $w_0 = 0.3, w_1 = 0.55, w_2 = 0.55$
Notice the weights are **different**! SGD follows a more erratic path because it updates immediately after each sample, using updated weights for subsequent calculations.

8 Summary: Calculation Steps Reference

Quick Reference: Step-by-Step Calculations

1. MSE Calculation:

1. Calculate error: $e_i = y_i - \hat{y}_i$
2. Square each error: e_i^2
3. Sum squared errors: $\sum e_i^2$
4. Divide by n: MSE $= \frac{1}{n}\sum e_i^2$

2. Binary Cross-Entropy:

1. For each sample: $L = -[y\log(p) + (1-y)\log(1-p)]$
2. If $y = 1$: $L = -\log(p)$
3. If $y = 0$: $L = -\log(1-p)$
4. Average all losses: BCE $= \frac{1}{N}\sum L_i$

3. Categorical Cross-Entropy:

1. Find predicted probability for TRUE class: $\hat{y}_{\text{true}}$
2. Calculate: CCE $= -\log(\hat{y}_{\text{true}})$
3. Average over samples if multiple

4. Gradient Descent (per sample):

1. Calculate output: $o = \sum w_i \cdot x_i$
2. Calculate error: $e = t - o$
3. Calculate delta: $\Delta w_i = \eta \cdot e \cdot x_i$
4. **Batch:** Accumulate all Δw, update once
5. **SGD:** Update immediately after each sample

Self-Test — Check Your Understanding

Practice Problems:

1. Calculate MSE for: Actual = [2, 4, 6], Predicted = [2.5, 3.5, 6.5]
2. Calculate BCE for: $y = 1$, $p = 0.6$
3. If $w = 0.5$, $x = 2$, $t = 3$, $o = 2$, $\eta = 0.1$, find Δw

Answers:

1. MSE $= \frac{(0.5)^2+(0.5)^2+(0.5)^2}{3} = \frac{0.75}{3} = 0.25$
2. BCE $= -\log(0.6) = 0.511$
3. $\Delta w = 0.1 \times (3-2) \times 2 = 0.2$

9 Glossary

Term	Definition
Loss Function	Measures error for a single training sample
Cost Function	Average loss over entire training dataset
MSE	Mean Squared Error — average of squared differences
BCE	Binary Cross-Entropy — loss for 2-class classification
CCE	Categorical Cross-Entropy — loss for multi-class classification
Gradient	Vector of partial derivatives indicating direction of steepest increase
Learning Rate (η)	Step size for weight updates
Epoch	One complete pass through all training data
Batch GD	Update weights once after processing all samples
SGD	Update weights after each individual sample
One-Hot Encoding	Binary vector representation of categorical labels