

What is Binary Cross-Entropy Loss?

It is the most common **loss function for binary classification** (two classes: 0 or 1).
It measures **how far the predicted probability is from the actual label**.

◆ Formula

For target $y \in \{0, 1\}$ and predicted probability \hat{y} :

$$\text{BCE Loss} = -[y \cdot \ln(\hat{y}) + (1 - y) \cdot \ln(1 - \hat{y})]$$

◆ Step-by-Step Explanation

Case 1: If ” Ask ChatGPT $y = 1$

- Loss = $-\ln(\hat{y})$
- If prediction is good (\hat{y} close to 1), loss is small.
- If prediction is bad (\hat{y} close to 0), loss is very large.

✓ Example:

True = 1, Predicted = 0.9

— $\ln(0.9) = 0.105 \rightarrow$ small loss (good).

True = 1, Predicted = 0.1

— $\ln(0.1) = 2.302 \rightarrow$ large loss (bad).

Case 2: If the true label is $y = 0$

- Loss = $-\ln(1 - \hat{y})$
- If prediction is good (\hat{y} close to 0), loss is small.
- If prediction is bad (\hat{y} close to 1), loss is very large.

✓ Example:

True = 0, Predicted = 0.1

— $\ln(0.9) = 0.105 \rightarrow$ small loss (good).

True = 0, Predicted = 0.9

— $\ln(0.1) = 2.302 \rightarrow$ large loss (bad).

Intuition

- Loss is low when prediction matches truth.
- Loss is high when prediction is far from truth.
- It's like a **penalty**: the more wrong you are, the bigger the penalty.

In short:

Binary Cross-Entropy loss tells us **how close the predicted probability is to the actual label (0 or 1)**.

- Correct & confident \rightarrow **small loss**.
- Wrong & confident \rightarrow **huge loss**.

1. Forward Pass

- You give input → network processes through layers → produces an output (prediction).
 - Example: The network predicts 0.72 when the true answer is 1.
-

2. Loss Calculation

- Use **Binary Cross-Entropy (BCE)** to measure **how wrong the prediction is**.
 - In this example, loss ≈ 0.33 (not perfect).
-

3. Backpropagation (Error Flowing Backward)

- The network asks: “*Which weights caused this error?*”
- It computes **gradients** (slopes) using the chain rule:
 - If increasing a weight would reduce the loss, gradient is **negative** → weight should increase.
 - If decreasing a weight would reduce the loss, gradient is **positive** → weight should decrease.
- This happens layer by layer, from output → hidden → input.

◆ 4. Optimization (Weight Update)

- Once gradients are known, the optimizer (e.g., **Gradient Descent**) updates weights:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

- Here, η = learning rate (step size).
- This shifts the weights slightly in the direction that makes the loss smaller next time.

5. Repeat

- With updated weights, the next forward pass gives a prediction closer to the true label.
- Over many iterations, the loss keeps decreasing, and predictions improve.

In short:

- **Backpropagation** → figures out *how much each weight contributed to the error*.

- **Optimization** → adjusts the weights in the right direction to reduce the error.