Lecture 6
Cost Functions

Contents

1 Training a Neural Network

Training a neural network involves three key stages:

1. **Forward Propagation:** The input data passes through the network layer by layer, producing an output. (We already covered this in the previous chapter.)
2. **Computing the Cost / Loss Function:** Measures how far the predicted output ($\hat{y}$) is from the true output (y).
3. **Backpropagation and Weight Update:** Gradients are calculated and propagated backward to update the weights and biases, minimizing the loss.

Mathematically, the linear combination at any neuron is:

$$z = w \cdot x + b$$

and the neuron's output is given by applying an activation function f:

$$\hat{Y} = f(z)$$

Activation Functions

Activation functions introduce non-linearity into the model, allowing the network to learn complex relationships between inputs and outputs.

- **Sigmoid:**
$$f(x) = \frac{1}{1 + e^{-x}}$$
- **Tanh (Hyperbolic Tangent):**
$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$
- **ReLU (Rectified Linear Unit):**
$$f(x) = \max(0, x)$$

Note: We will explore these activation functions in detail later.

1.1 Cost / Loss Function

After forward propagation, we measure how wrong the model is — that's where the **loss function** (or **cost function**) comes in.

Definition

The **Loss Function** quantifies the difference between the predicted output and the actual target value. It converts this difference into a single real number that represents how poorly the model is performing.

The goal of training is to minimize this loss value by adjusting the network's weights and biases through backpropagation.

1.2 Types of Cost Functions

Different tasks require different loss functions. We broadly classify them into:

- Regression Cost Functions
- Classification Cost Functions

1.3 Regression Losses

Mean Error (ME)

$$ME = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)$$

Intuition: simple average of signed errors. **Problem:** positives and negatives cancel; can be near zero even when predictions are bad. **Rarely used** beyond teaching.

Mean Squared Error (MSE)

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

Intuition: squares magnify larger mistakes; very smooth and differentiable. **Pros:** strong penalty on big errors; plays nicely with calculus. **Cons:** *not robust* to outliers (a few huge errors can dominate). **Gradient (w.r.t. prediction):**

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

Takeaway: the update size grows with the size of the error.

Mean Absolute Error (MAE)

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Intuition: measures typical absolute deviation; *robust* to outliers. **Pros:** less sensitive to spikes/noise; Median minimizes MAE. **Cons:** non-differentiable at 0; uses a subgradient. **Subgradient (w.r.t. prediction):**

$$\frac{\partial MAE}{\partial \hat{y}_i} = \frac{1}{N}\,\text{sign}(\hat{y}_i - y_i) \quad (\hat{y}_i \neq y_i)$$

Quick Numeric Intuition If one sample has error $e = 10$ and another $e = 1$:

$$\text{MSE penalty: } 10^2 + 1^2 = 101 \quad \text{vs} \quad \text{MAE penalty: } |10| + |1| = 11$$

MSE disproportionately punishes the big miss; MAE treats both linearly.

1.4 Classification: Cross-Entropy Family

Binary Cross-Entropy (with Sigmoid) Model outputs a probability $p = \sigma(z)$ with $\sigma(z) = \frac{1}{1+e^{-z}}$.

$$L_{\text{BCE}}(y, p) = -\big[y \log p + (1-y)\log(1-p)\big]$$

Intuition: negative log-likelihood of a Bernoulli; *confident and wrong* ⇒ huge penalty (because $\log(\cdot)$ blows up near 0). **Nice property:** using logits z (pre-sigmoid), the gradient is clean:

$$\frac{\partial L_{\text{BCE}}}{\partial z} = \sigma(z) - y = p - y$$

This is why many libraries provide `BCEWithLogitsLoss`: it's numerically stable and computes BCE directly from logits.

Categorical (Multi-Class) Cross-Entropy (with Softmax) For K classes, logits $\mathbf{z} \in R^K$, probabilities $p_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$, and one-hot label y_k:

$$L_{\text{CE}}(\mathbf{y}, \mathbf{p}) = -\sum_{k=1}^{K} y_k \log p_k$$

Intuition: penalizes probability mass assigned away from the true class. **Gradient (w.r.t. logits):**

$$\frac{\partial L_{\text{CE}}}{\partial z_k} = p_k - y_k$$

Why it works well: it directly pushes predicted distribution $\mathbf{p}$ toward the label distribution $\mathbf{y}$.

Multi-Label Note (common in CV/NLP) If classes are non-exclusive (e.g., "cat" *and* "outdoor"), use *independent* sigmoids per class with BCE, not softmax CE.

1.5 When to Use What (Practical Guide)

- **Regression without heavy outliers:** MSE (smooth gradients, fast convergence).
- **Regression with outliers/noise:** MAE (robust) or Huber/Smooth-L1 (MAE near outliers, MSE near small errors).
- **Binary classification:** BCE (prefer *with logits* for stability).
- **Multi-class (single label):** Softmax + Cross-Entropy.
- **Class imbalance:** use class weights, focal loss, or resampling.

1.6 Cross-Entropy: Numeric Feel

For a positive example ($y = 1$):

$$p = 0.9 \Rightarrow L \approx -\log(0.9) = 0.105, \quad p = 0.6 \Rightarrow 0.511, \quad p = 0.01 \Rightarrow 4.605$$

Confident and wrong is punished *much* more than being uncertain.

1.7 Loss vs Cost Function

While often used interchangeably, there is a subtle difference:

- The **Loss Function** is calculated per training example — it measures the model's performance for a single instance.
- The **Cost Function** is the *average loss across the entire dataset*, often including regularization terms.

$$\text{Cost Function} = \frac{1}{N}\sum_{i=1}^{N} \text{Loss}(y_i, \hat{y}_i) + \text{Regularization Penalty}$$

Summary

- The loss function measures how well the neural network predicts a single example.
- The cost function aggregates all those losses to estimate the model's overall performance.
- The training goal is to **minimize the cost function** using gradient descent and backpropagation.