Topic-1: Expanded Derivation of Mean Squared Error (MSE)

Bias, Variance and Induced Error Analysis

1. Introduction

Mean Squared Error (MSE) is one of the most widely used loss functions in machine learning, statistics, and neural networks. It measures the average squared difference between actual values and predicted values. MSE provides not only an optimization objective but also deep statistical insight into prediction quality through bias-variance decomposition.

2. Definition of MSE

For a dataset containing N samples, the Mean Squared Error is defined as:

$$\mathbf{MSE = (1/N)\ \Sigma\ (yi - \hat{y}i)^2}$$

where:
- yi = true target value
- ŷi = predicted value
- N = number of samples

The squaring operation penalizes larger errors more strongly, making MSE particularly suitable for regression problems.

3. Statistical Model

Assume that the observed target variable Y can be represented as:

$$\mathbf{Y = f(x) + \varepsilon}$$

where:
- f(x) is the true underlying function
- ε is random noise
- E[ε] = 0
- Var(ε) = σ^2

4. Mathematical Derivation of MSE Expansion

The expected prediction error can be written as:

$$\mathbf{MSE = E[(Y - \hat{f}(x))^2]}$$

Substituting Y = f(x) + ε:

$$\mathbf{MSE = E[(f(x) + \varepsilon - \hat{f}(x))^2]}$$

Expanding the square using $(a+b)^2 = a^2 + 2ab + b^2$:

$$\mathbf{MSE = E[(f(x)-\hat{f}(x))^2 + 2\varepsilon(f(x)-\hat{f}(x)) + \varepsilon^2]}$$

Applying expectation term-by-term:

$$MSE = E[(f(x)-\hat{f}(x))^2] + 2E[\varepsilon(f(x)-\hat{f}(x))] + E[\varepsilon^2]$$

Since noise ε is assumed independent and E[ε]=0:

$$E[\varepsilon(f(x)-\hat{f}(x))] = 0$$

Therefore:

$$MSE = E[(f(x)-\hat{f}(x))^2] + \sigma^2$$

5. Bias-Variance Decomposition

Introduce the expected prediction $E[\hat{f}(x)]$ inside the squared term:

$$f(x)-\hat{f}(x) = f(x)-E[\hat{f}(x)] + E[\hat{f}(x)]-\hat{f}(x)$$

Let:

$$A = f(x)-E[\hat{f}(x)]$$

$$B = E[\hat{f}(x)]-\hat{f}(x)$$

Then:

$$(A+B)^2 = A^2 + 2AB + B^2$$

Taking expectation and using E[B]=0:

$$E[(f(x)-\hat{f}(x))^2] = Bias^2 + Variance$$

Thus, the final decomposition becomes:

$$MSE = Bias^2 + Variance + Noise$$

6. Interpretation of Terms

Component	Meaning
$Bias^2$	Error due to incorrect assumptions or underfitting
Variance	Sensitivity of model predictions to training data
Noise / Induced Error	Irreducible uncertainty inherent in data

7. Neural Network Perspective

In neural networks:

- Small networks often exhibit high bias and low variance.
- Very deep or highly parameterized networks may show low bias but high variance.

- Regularization methods such as dropout, weight decay, and early stopping help control variance.

Numerical Example

Suppose the true function value is:

$f(x) = 10$

Predictions obtained from different training datasets are: **8, 9, 11, 12**

Step 1: Compute mean prediction:

$E[\hat{f}(x)] = (8+9+11+12)/4 = 10$

Step 2: Compute $Bias^2$

$Bias^2 = (10 - 10)^2 = 0$

Step 3: Compute Variance

$Variance = [(8-10)^2 + (9-10)^2 + (11-10)^2 + (12-10)^2]/4$

$= (4 + 1 + 1 + 4)/4 = 2.5$

Step 4: Assume noise variance

$\sigma^2 = 1$

Final MSE:

$MSE = 0 + 2.5 + 1 = 3.5$

Topic-2: Training of a Neural Network to Achieve a Given Distribution using Maximum Likelihood Estimation (MLE)

1. Introduction
In probabilistic neural networks, the objective is to predict a single value and to learn the complete probability distribution of the data.

Maximum Likelihood Estimation (MLE) is one of the most fundamental methods used for this purpose. The neural network learns parameters such that the observed data becomes highly probable under the learned model distribution.

2. Basic Idea of Maximum Likelihood
Suppose the training dataset is:

$D = \{x1, x2, ..., xN\}$

Assume the neural network models a probability distribution:

$p(x; \theta)$

where θ represents neural network parameters.

The objective is to find parameters that maximize the probability of observing the training data.

3. Likelihood Function
Assuming samples are independent, the likelihood function becomes:

$$\mathbf{p(D; \theta) = \prod p(x\_i; \theta)}$$

4. Log-Likelihood
Products are numerically unstable and difficult to optimize. Therefore logarithm is applied to convert products into sums.

$$\mathbf{\log p(D; \theta) = \log \prod p(x\_i; \theta)}$$

$$\mathbf{\log p(D; \theta) = \Sigma \log p(x\_i; \theta)}$$

The optimization objective becomes maximizing the total log-probability of observed samples.

5. Maximum Likelihood Objective
$\theta^* = \text{argmax}_\theta \Sigma \log p(x_i; \theta)$

Optimization frameworks typically minimize losses. Therefore we minimize the negative log-likelihood (NLL):

$L(\theta) = - \Sigma \log p(x_i; \theta)$

6. Neural Network Interpretation
The neural network predicts parameters of a probability distribution. Different applications use different distributions.

Application	**Distribution Output**
Regression	Gaussian Mean and Variance
Classification	Categorical Probabilities
Language Models	Token Probabilities
Generative AI	Complex Learned Distributions

7. Gaussian Likelihood Example
Suppose the target variable follows a Gaussian distribution:

$y \sim N(\mu, \sigma^2)$

The neural network predicts the mean:

$\mu = f_\theta(x)$

Gaussian probability density function:

$p(y|x; \theta) = (1/\sqrt{(2\pi\sigma^2)}) \exp(-(y-f_\theta(x))^2 / (2\sigma^2))$

8. Derivation of Negative Log-Likelihood
Taking logarithm of the likelihood:

$\log p(y|x; \theta) = \log(1/\sqrt{(2\pi\sigma^2)}) - (y-f_\theta(x))^2/(2\sigma^2)$

Using logarithm properties:

$\log p(y|x; \theta) = -½ \log(2\pi\sigma^2) - (y-f_\theta(x))^2/(2\sigma^2)$

Negating both sides gives the Negative Log-Likelihood:

$-\log p(y|x; \theta) = ½ \log(2\pi\sigma^2) + (y-f_\theta(x))^2/(2\sigma^2)$

If variance σ^2 is constant, minimizing NLL becomes equivalent to minimizing Mean Squared Error (MSE).

$MSE = (y - f_\theta(x))^2$

9. Neural Network Training Procedure

1. Step 1: Input training sample x
2. Step 2: Forward pass through network
3. Step 3: Predict output $\hat{y} = f\theta(x)$
4. Step 4: Compute likelihood $p(y|x; \theta)$
5. Step 5: Compute negative log-likelihood loss
6. Step 6: Compute gradients using backpropagation
7. Step 7: Update parameters using gradient descent

10. Gradient Descent Update

$\theta \leftarrow \theta - \eta \nabla_\theta L$

where η is the learning rate and $\nabla\theta L$ represents the gradient of the loss with respect to model parameters.

11. Numerical Example

Suppose:

True value $y = 5$

Predicted value $\hat{y} = 4$

$\sigma^2 = 1$

Compute Gaussian likelihood:

$p(y|x; \theta) = (1/\sqrt{(2\pi)}) \exp(-(5-4)^2/2)$

Since:

$\exp(-0.5) \approx 0.6065$

Thus:

$p(y|x; \theta) \approx 0.242$

Negative Log-Likelihood:

$L = -\log(0.242)$

$L \approx 1.418$

Now suppose prediction improves to $\hat{y} = 4.8$

$(5 - 4.8)^2 = 0.04$

The likelihood increases and the NLL decreases. Thus the optimizer pushes the neural network toward the target distribution.

12. Classification using MLE
For classification problems, neural networks typically use softmax probabilities.

$p(y=k|x; \theta) = Softmax(z_k)$

The corresponding loss becomes cross-entropy:

$L = - \Sigma\ y_i \log(\hat{y}_i)$

13. Advantages of Maximum Likelihood

- Provides probabilistic interpretation
- Strong statistical foundation
- Supports uncertainty estimation
- Differentiable and suitable for backpropagation
- Used extensively in modern AI systems

14. Applications

- Regression models
- Classification systems
- Large Language Models
- Diffusion models
- Variational Autoencoders
- Scientific simulations
- Probabilistic forecasting

Training a Neural Network to obtain Rayleigh Distribution in the predicted values

Rayleigh Distribution: The probability density function (PDF) of the Rayleigh distribution is:

$$p(y; \sigma) = (y/\sigma^2) \exp(-y^2/(2\sigma^2))$$

where:
- $y \geq 0$
- σ is the scale parameter
- Larger σ produces wider distributions

Neural Network Objective: The neural network does not directly predict y. Instead, it predicts the Rayleigh scale parameter σ.

$$\sigma\theta(x) = \text{Neural Network Output}$$

The training objective becomes learning the parameter θ such that:

$$y_i \sim \text{Rayleigh}(\sigma\theta(x_i))$$

Neural Network Architecture
Typical architecture:

- Input Layer: x
- Hidden Layers: Dense + ReLU
- Output Layer: Positive σ prediction

Since σ must remain positive, the output layer commonly uses Softplus or Exponential activation functions.

$$\sigma\theta(x) = \log(1 + e^z)$$

Likelihood Function
For one training sample:

$$p(y_i|x_i; \theta) = (y_i/\sigma_i^2) \exp(-y_i^2/(2\sigma_i^2))$$

For N independent samples:

$$p(D; \theta) = \prod p(y_i|x_i; \theta)$$

Log-Likelihood: Taking logarithm converts products into sums.

$$\log p(D; \theta) = \Sigma \log p(y_i|x_i; \theta)$$

Substituting the Rayleigh PDF:

$$\log p(D; \theta) = \Sigma [\log(y_i) - 2\log(\sigma_i) - y_i^2/(2\sigma_i^2)]$$

7Negative Log-Likelihood Loss

Neural networks minimize the negative log-likelihood (NLL):

$$L(\theta) = - \log p(D; \theta)$$

Ignoring terms independent of θ gives the final training loss:

$$L(\theta) = \Sigma [2\log(\sigma_i) + y_i^2/(2\sigma_i^2)]$$

Step-by-Step Training Procedure

1. Step 1: Prepare dataset (x_i, y_i)
2. Step 2: Design neural network
3. Step 3: Perform forward pass
4. Step 4: Predict σ_i = σθ(x_i)
5. Step 5: Compute Rayleigh NLL loss
6. Step 6: Compute gradients using backpropagation
7. Step 7: Update parameters using gradient descent or Adam optimizer

Gradient Descent Update

$$\theta \leftarrow \theta - \eta \nabla\theta L$$

where η is the learning rate.

Numerical Example

Suppose:

Observed value $y = 4$

Predicted $\sigma = 3$

The loss becomes:

$L = 2\log(3) + 4^2/(2\times3^2)$

Compute each term:

$2\log(3) \approx 2.197$

$16/18 \approx 0.889$

Therefore:

$L \approx 3.086$

11. Improved Prediction
Suppose the network improves prediction to:

$\sigma = 2.8$

The loss decreases further, indicating better distribution matching.