KNU BAYESIAN LAB

# Introduction to Bayesian Neural Networks from Fundamentals of MLP to Bayesian Inference

**BK21 Study : 2nd Generation of MCMC**

**Gyeongmin Park**

Department of Statistics, Kyungpook National University,
Daegu, Republic of Korea

7 Jan, 2026

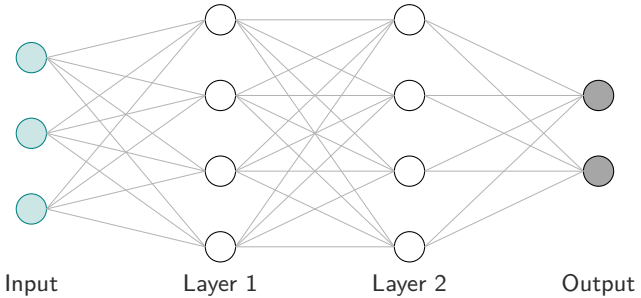
# 1. Deep Learning and Neural Network

# 1.1. Basic Concept of Neural Networks

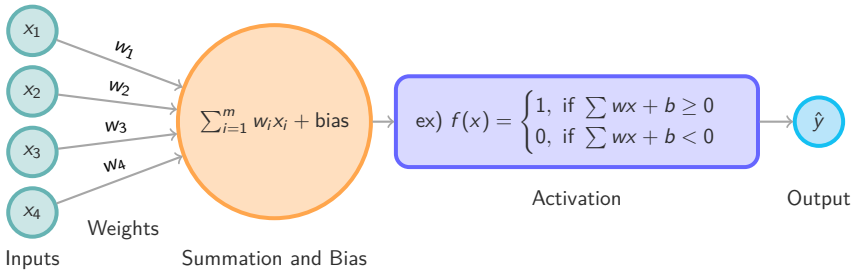
- **What is a Neural Network?** A machine learning model inspired by the human brain
  - Network structure with interconnected neurons (nodes) organized in layers
  - Each neuron receives inputs, performs calculations, and sends signals to the next layer
- **Basic Components**
  - Input Layer: Receives the input data
  - Hidden Layer: Where actual learning and computation occurs
  - Output Layer: Produces the final predictions
- **Training Data**: $K$ pairs of training samples $\{(\mathbf{x_i}, \mathbf{y_i})\}_{i=1}^{K}$
  - $\mathbf{x_i} \in \mathbb{R}^{N_x}$: Input data ($N_x$ dimensions)
  - $\mathbf{y_i} \in \mathbb{R}^{N_y}$: Target output ($N_y$ dimensions)

# 1.2. Deep Learning, Multi-Layer Networks

- **What is Deep Learning?**
  - Neural networks with multiple hidden layers
  - "Deep" refers to many layers of processing



# 1.3. Mathematical Foundation of Single Neuron

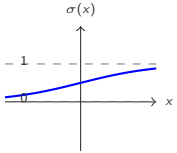



- A single neuron performs: **weighted sum → activation function → output**
- The activation function determines whether the neuron "fires" (outputs a value or 0)
- This is the fundamental building block of all neural networks
- The **summation and bias** part can be simply thought of as a statistical **regressor** or **classifier** from traditional statistics

# 1.4. Well Known Activation Functions (1/2)

### Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

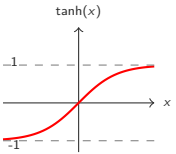



Output: [0,1]

Vanishing gradient

### Tanh

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

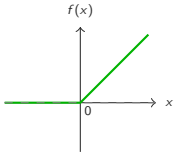



Output: [-1,1]

Zero-centered

### ReLU

$$\mathrm{ReLU}(x) = \max(0, x)$$



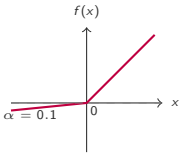

Output: $[0, \infty)$

Simple & efficient

# 1.4. Well Known Activation Functions (2/2)

### Leaky ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

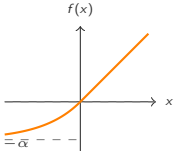



Prevents dying ReLU

Small negative slope

### ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

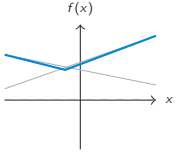



Smooth negative part

Faster convergence

### Maxout

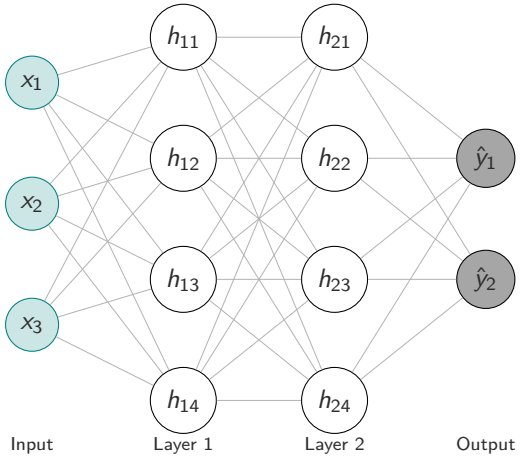
$$f(x) = \max_i (w_i^T x + b_i)$$



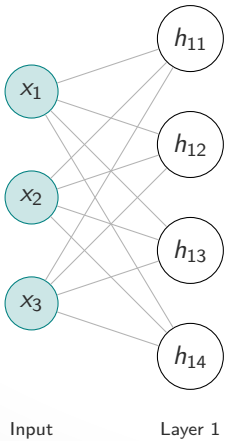

Learnable activation

Generalizes ReLU

# 1.5. Overview of Multi-Layer Perceptron Network Structure (Feedforward)



# 1.6. Mathematical Structure (1/3): Input to Hidden Layer 1




$$h_{11} = f(w_{111}x_1 + w_{112}x_2 + w_{113}x_3 + b_{11})$$
$$h_{12} = f(w_{121}x_1 + w_{122}x_2 + w_{123}x_3 + b_{12})$$
$$h_{13} = f(w_{131}x_1 + w_{132}x_2 + w_{133}x_3 + b_{13})$$
$$h_{14} = f(w_{141}x_1 + w_{142}x_2 + w_{143}x_3 + b_{14})$$

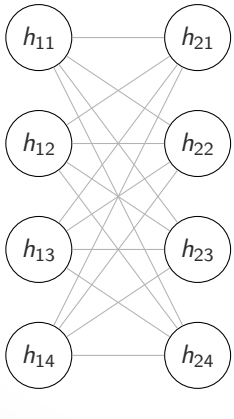
**Matrix Form:**

$$\mathbf{h}^{(1)} = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

where:

- $\mathbf{W}^{(1)} \in \mathbb{R}^{4\times 3}$ (weight matrix)
- $\mathbf{x} \in \mathbb{R}^3$ (input vector)
- $\mathbf{b}^{(1)} \in \mathbb{R}^4$ (bias vector)
- $f(\cdot)$ is the activation function

# 1.6. Mathematical Structure (2/3): Hidden Layer 1 to Hidden Layer 2




$$h_{21} = f(w_{211}h_{11} + w_{212}h_{12} + w_{213}h_{13} + w_{214}h_{14} + b_{21})$$
$$h_{22} = f(w_{221}h_{11} + w_{222}h_{12} + w_{223}h_{13} + w_{224}h_{14} + b_{22})$$
$$h_{23} = f(w_{231}h_{11} + w_{232}h_{12} + w_{233}h_{13} + w_{234}h_{14} + b_{23})$$
$$h_{24} = f(w_{241}h_{11} + w_{242}h_{12} + w_{243}h_{13} + w_{244}h_{14} + b_{24})$$

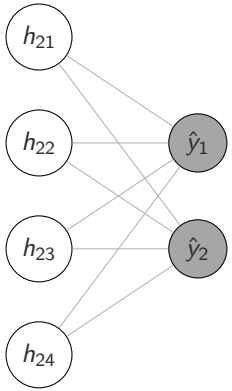
**Matrix Form:**

$$\mathbf{h}^{(2)} = f(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

where:

- $\mathbf{W}^{(2)} \in \mathbb{R}^{4\times4}$ (weight matrix)
- $\mathbf{h}^{(1)} \in \mathbb{R}^4$ (hidden layer 1 output)
- $\mathbf{b}^{(2)} \in \mathbb{R}^4$ (bias vector)

# 1.6. Mathematical Structure (3/3): Hidden Layer 2 to Output




$$\hat{y}_1 = g(w_{o11}h_{21} + w_{o12}h_{22} + w_{o13}h_{23} + w_{o14}h_{24} + b_{o1})$$
$$\hat{y}_2 = g(w_{o21}h_{21} + w_{o22}h_{22} + w_{o23}h_{23} + w_{o24}h_{24} + b_{o2})$$

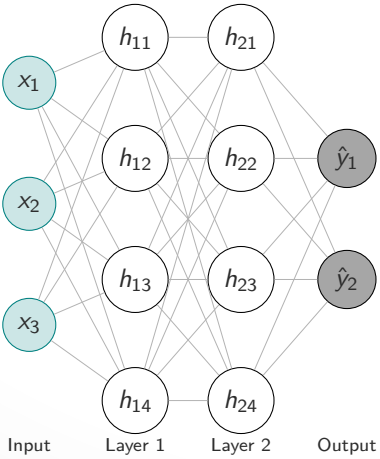
**Matrix Form:**

$$\hat{\mathbf{y}} = g(\mathbf{W^{(o)}}\mathbf{h^{(2)}} + \mathbf{b^{(o)}})$$

where:

- $\mathbf{W^{(o)}} \in \mathbb{R}^{2\times 4}$ (output weight matrix)
- $\mathbf{h^{(2)}} \in \mathbb{R}^4$ (hidden layer 2 output)
- $\mathbf{b^{(o)}} \in \mathbb{R}^2$ (output bias vector)
- $g(\cdot)$ is the output activation function

# 1.7. Mathematical Structure: Complete Network Overview




**Compact Forms of Nodes:**

$$\mathbf{h}^{(1)} = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}^{(2)} = f(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$
$$\hat{\mathbf{y}} = g(\mathbf{W}^{(o)}\mathbf{h}^{(2)} + \mathbf{b}^{(o)})$$
$$\hat{\mathbf{y}} = g(\mathbf{W}^{(o)}f(\mathbf{W}^{(2)}f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(o)})$$

**Functions:**

- $f(\cdot)$: Hidden activation (e.g., ReLU)
- $g(\cdot)$: Output activation (e.g., softmax)

# 2. Training Multi-Layer Perceptron Networks

## 2.1. What is a Loss Function?

- **Goal**: Find parameters $\theta$ that minimize the loss function $\mathcal{L}(\theta)$
- **Training Process**: Adjust weights and biases to reduce prediction errors

**Mathematical Formulation:**

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta)$$

where $\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \mathbf{W}^{(o)}, \mathbf{b}^{(o)}\}$

**Key Concept**:

- **Lower loss** = Better predictions
- **Higher loss** = Worse predictions
- Training = Finding the "best" parameters that minimize loss

## 2.2. Common Loss Functions

**1. Mean Squared Error (Regression):**

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2$$

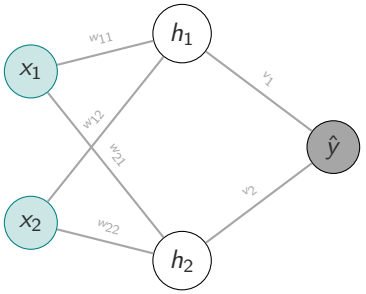
**2. Cross-Entropy Loss (Classification):**

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij})$$

**Notation:**

- $N$: Number of training samples
- $C$: Number of classes
- $\mathbf{y}_i$: True label for sample $i$
- $\hat{\mathbf{y}}_i$: Predicted output for sample $i$

# 2.3. Simple Case of MLP Network Architecture




**Hidden Layer Computations:**

$$h_1 = f(w_{11}x_1 + w_{12}x_2 + b_1)$$
$$h_2 = f(w_{21}x_1 + w_{22}x_2 + b_2)$$

**Output Layer Computation:**

$$\hat{y} = g(v_1 h_1 + v_2 h_2 + b_{out})$$

**Complete Network Function:**

$$\hat{y} = g\left(v_1 f(w_{11}x_1 + w_{12}x_2 + b_1) + v_2 f(w_{21}x_1 + w_{22}x_2 + b_2) + b_{out}\right)$$

# 2.4. Loss Function and Optimization

### Mean Squared Error (MSE) Loss Function

From our complete network function:

$$L(\theta) = \frac{1}{2N} \sum_{i=1}^{N} [y_i - g\left(v_1 f(w_{11} x_{1i} + w_{12} x_{2i} + b_1)\right. \\ \left. + v_2 f(w_{21} x_{1i} + w_{22} x_{2i} + b_2) + b_{out})]^2$$

### Gradient Descent Update Rule

$$\theta^{(t+1)} = \theta^t - \lambda \nabla L(\theta^t)$$

where $\theta = \{w_{11}, w_{12}, w_{21}, w_{22}, v_1, v_2, b_1, b_2, b_{out}\}$

## 2.5. Training for $w_{11}$ using Gradient Descent

$$w_{11}^{(t+1)} = w_{11}^{t} - \lambda \frac{\partial L(\theta^t)}{\partial w_{11}}$$

**Step 1: Expanding** $\frac{\partial L(\theta^t)}{\partial w_{11}}$

$$\frac{\partial L(\theta^t)}{\partial w_{11}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i) \cdot \left( - \frac{\partial \hat{y}_i}{\partial w_{11}} \right)$$

**Step 2: Computing** $\frac{\partial \hat{y}_i}{\partial w_{11}}$

$$\frac{\partial \hat{y}_i}{\partial w_{11}} = g'(\cdot) \cdot v_1 \cdot f'(w_{11}x_{1i} + w_{12}x_{2i} + b_1) \cdot x_{1i}$$

**Step 3: Final Update Formula**

$$w_{11}^{(t+1)} = w_{11}^{t} + \frac{\lambda}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i) \cdot g'(\cdot) \cdot v_1 \cdot f'(w_{11}x_{1i} + w_{12}x_{2i} + b_1) \cdot x_{1i}$$

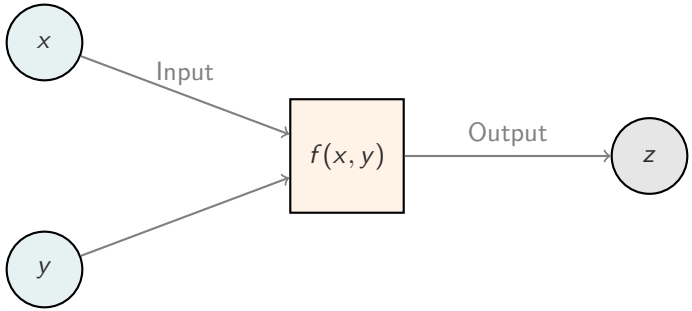
# 3. Backpropagation

# 3.1. The Problem with Manual Calculation

- **Recall our previous calculation:**
  - We calculated $\frac{\partial L}{\partial w_{11}}$ for a **single weight** manually.
  - We used the chain rule: Output $\rightarrow$ Activation $\rightarrow$ Summation $\rightarrow$ Input.
- **The Scale Problem:**
  - Modern Deep Neural Networks have **millions or billions** of parameters (weights).
  - Computing gradients one by one for every single weight independently is computationally impossible.
  - We need a more efficient algorithm that can compute gradients for **all weights at once**.

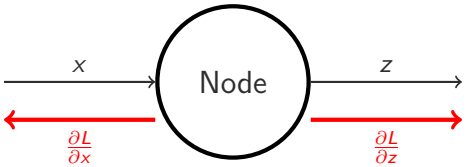
## 3.2. Computational Graphs of nodes

- To understand Backpropagation, we represent equations as **Graphs**.
- **Nodes**: Mathematical operations (add, multiply, max, activation function).
- **Edges**: Flow of data (values).



# 3.3. The Key Insight: Local Gradient vs. Upstream Gradient

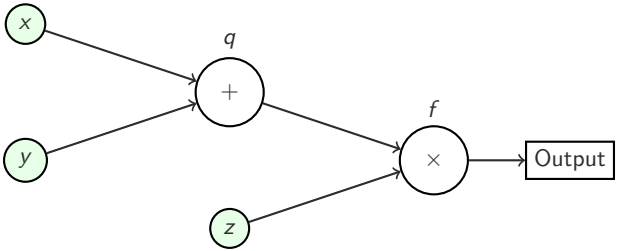
- Every node in the graph only needs to know two things to calculate its gradient:




- By the Chain Rule: $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \times \frac{\partial z}{\partial x}$
- **Upstream Gradient ($\partial L/\partial z$)**: Passed back from the end.
- **Local Gradient ($\partial z/\partial x$)**: Can be calculated immediately during the forward pass.

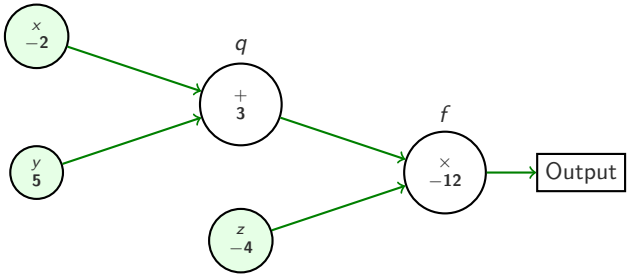
# 3.4. Step-by-Step Example (1/4): Problem Setup

- Let's compute gradients for a simple function: $f(x, y, z) = (x + y) \cdot z$
- We want to find $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$.
- Let inputs be: $x = -2, \quad y = 5, \quad z = -4$
- We introduce an intermediate variable $q = x + y$, so $f = q \cdot z$.



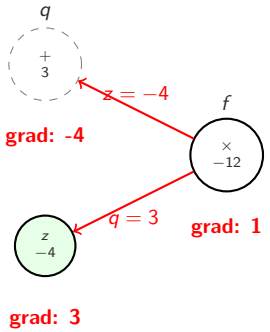
# 3.5. Step-by-Step Example (2/4): Forward Pass

- Compute values from left to right (Green numbers).
- $q = x + y = -2 + 5 = 3$
- $f = q \cdot z = 3 \cdot (-4) = -12$



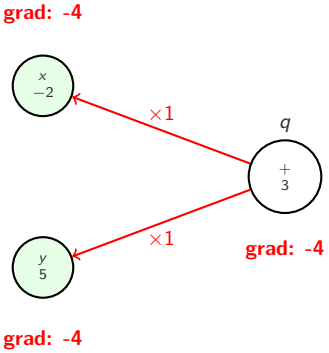
# 3.6. Step-by-Step Example (3/4): Backward Pass (First Steps)

- Start from the end. The gradient of a variable with respect to itself is 1. $\frac{\partial f}{\partial f} = 1$
- **At the multiplication node ($f = q \cdot z$):**
  - Local gradients: $\frac{\partial f}{\partial q} = z = -4, \quad \frac{\partial f}{\partial z} = q = 3$
  - Apply chain rule (Upstream $\times$ Local): $\frac{\partial f}{\partial q} = 1 \times -4 = \mathbf{-4} \quad , \quad \frac{\partial f}{\partial z} = 1 \times 3 = \mathbf{3}$



# 3.7. Step-by-Step Example (4/4): Backward Pass (Final Steps)

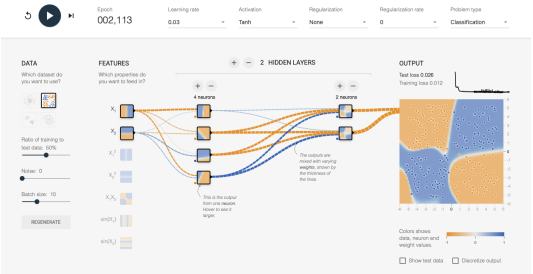
- **At the addition node ($q = x + y$):**
  - Local gradients: $\frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1$
  - Upstream gradient (from $q$) is $-\mathbf{4}$.
  - Apply chain rule: $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} = -4 \times 1 = \mathbf{-4} \quad , \quad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y} = -4 \times 1 = \mathbf{-4}$



# 3.8. Interactive Neural Network Playground

**Practice with Real Neural Networks**

- Visualize neural network training in real-time
- Experiment with different architectures
- Test various activation functions
- Observe how parameters affect learning




**Try it yourself:** TensorFlow Playground

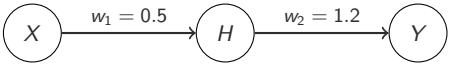
# 4. Bayesian Neural Networks

# 4.1. The Bayesian Perspective

- **Frequentist View (Standard Deep Learning):**
  - Parameters (Weights $W$) are fixed, unknown constants.
  - We try to find the "best" single value (Point Estimate) using MLE.
  - $\hat{W} = \arg\max_W f(y|X, W)$.
- **Bayesian View:**
  - Parameters $W$ are treated as **Random Variables**.
  - We do not find a single value; we find a **distribution** of values.
  - We aim to find the **Posterior Distribution** $\pi(W|y)$.

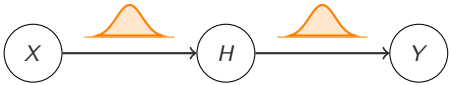
# 4.2. Visualizing the Difference

**Standard Neural Network**




Weights have **fixed values**.

**Bayesian Neural Network**




Weights are assigned **distributions**.

## 4.3. Uncertainty Quantification of BNNs

- The biggest advantage of BNNs is that they know **"what they don't know"**.
- We can quantify predictive uncertainty using the variance:

$$Var(y_{pred}|x_{pred}, y)$$

- Standard networks might be "confidently wrong," but BNNs will report high uncertainty.

## 4.4. Finding the Posterior

- Our goal is to compute the posterior distribution of weights given the data:

$$\pi(W|y) \propto f(y|X, W) \times \pi(W)$$

- **Components:**
  - $\pi(W|y)$: **Posterior** (What we want to know).
  - $f(y|X, W)$: **Likelihood** (How well weights fit the data).
  - $\pi(W)$: **Prior** (Initial belief about weights).
- **Challenge:** Calculating this exactly is often impossible due to the complex integral in the denominator (evidence).

# 4.5. Two Main Approaches to Solve

- Since we cannot compute the posterior analytically, we use approximation methods.
- **1. Markov Chain Monte Carlo (MCMC)**
  - Can compute the posterior exactly (in the limit).
  - **Cons:** Becomes computationally expensive as the number of parameters $W$ increases.
- **2. Variational Inference (VI)**
  - Approximates the posterior with a simpler distribution.
  - **Pros:** Much faster compared to MCMC.
  - **Cons:** Provides an approximate posterior, not the exact one.

# 4.6. The Concept of Variational Inference (VI)

- **Idea:** Instead of finding the exact $\pi(W|y)$, let's find a "proxy" distribution $q_\lambda(W)$ that best approximates the posterior.
- We choose a family of distributions for $q_\lambda(W)$, usually a Gaussian:

$$q_\lambda(W) \sim \mathcal{N}_L(\mu, \Sigma)$$

  where $\lambda = (\mu, \Sigma)$ are the parameters we want to learn.
- **Analogy:**
  - The true posterior is a complex shaped cloud.
  - We try to fit a simple oval (Gaussian) that best overlaps with that cloud.

# 4.7. Challenges in MCMC for High-Dimensional Problems (1/3)

- **Recall:** We need to sample from the posterior $\pi(W|y)$ where $W$ contains millions of parameters.
- **Random Walk Metropolis-Hastings (RWMH)**
  - Proposal: $W^{(t+1)} = W^{(t)} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$
  - Accept/Reject based on Metropolis ratio
- **Problems in High Dimensions:**
  - Random exploration without direction guidance
  - Proposal steps must be very small to maintain reasonable acceptance rate
  - Moves very slowly through the parameter space
  - Takes exponentially many iterations to explore the target distribution

# 4.7. Challenges in MCMC for High-Dimensional Problems (2/3)

- **Effective Sample Size (ESS)**
  - Measures the quality of MCMC samples
  - Definition: $\text{ESS} = \frac{N}{1+2\sum_{k=1}^{\infty} \rho_k}$
    - $N$: Total number of samples
    - $\rho_k$: Autocorrelation at lag $k$
  - **Interpretation:** How many "independent" samples we effectively have
  - Higher ESS = Better mixing = More efficient sampling

- **RWMH in High Dimensions:**
  - ESS becomes extremely low ($\text{ESS} \ll N$)
  - High autocorrelation between consecutive samples
  - Need millions of iterations to get a few hundred effective samples
  - Computationally prohibitive for BNNs

# 4.7. Challenges in MCMC for High-Dimensional Problems (3/3)

- **Gibbs Sampler**
  - Samples each parameter conditionally: $W_i^{(t+1)} \sim \pi(W_i | W_{-i}^{(t)}, y)$
  - Requires explicit conditional distributions
  - For Neural Networks: Conditionals are **intractable**
  - Not applicable to general BNN architectures
- **Metropolis Adjusted Langevin Algorithm (MALA)**
  - Uses gradient information: $W^{(t+1)} = W^{(t)} + \frac{\epsilon^2}{2} \nabla \log \pi(W^{(t)}|y) + \epsilon Z$
  - Better than RWMH, but still has slow convergence in high dimensions
  - Step size $\epsilon$ must be very small for acceptance
  - Scaling issues: $\epsilon \propto d^{-1/6}$ where $d$ is dimension
- **Solution:** We need a more sophisticated method $\rightarrow$ **Hamiltonian Monte Carlo (HMC)**

# Introduction to Bayesian Neural Networks from Fundamentals of MLP to Bayesian Inference

**BK21 Study : 2nd Generation of MCMC**

**Gyeongmin Park**

Department of Statistics, Kyungpook National University,
Daegu, Republic of Korea

7 Jan, 2026