

# Bayesian Deep Learning

EE807: Recent Advances in Deep Learning

Lecture 8

Slide made by

Sungsoo Ahn and Kimin Lee

KAIST EE

### **1. Introduction**

- What is Bayesian inference?
- What is Bayesian neural network?

### **2. Variational Inference for Bayesian Neural Networks**

- Using Gaussian distribution
- Using multiplicative normalizing flows

### **3. Non-variational Inference for Bayesian Neural Networks**

- Laplace Approximation
- Markov Chain Monte Carlo

### **1. Introduction**

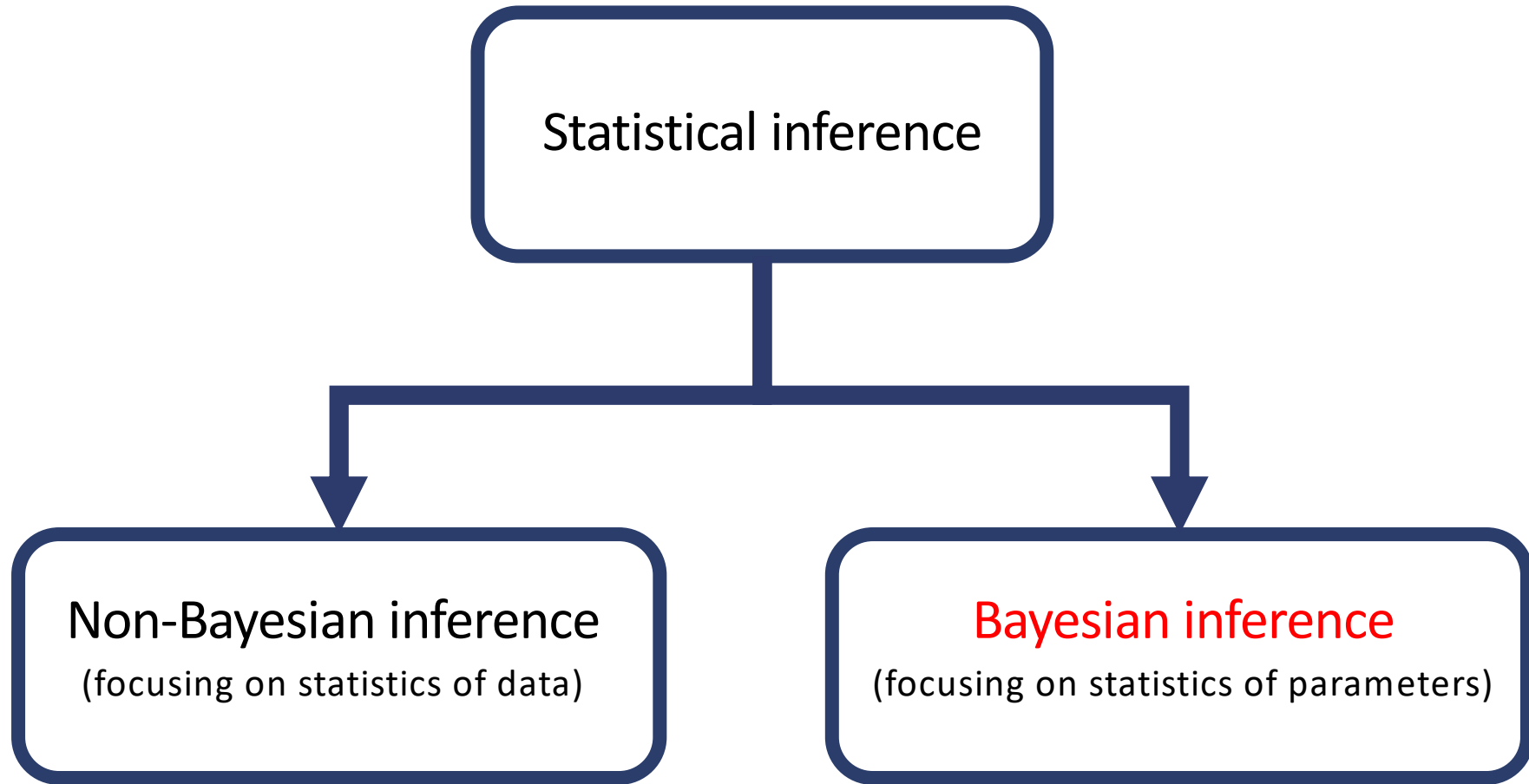
- What is Bayesian inference?
- What is Bayesian neural network?

### **2. Variational Inference for Bayesian Neural Networks**

- Using Gaussian distribution
- Using multiplicative normalizing flows

### **3. Non-variational Inference for Bayesian Neural Networks**

- Laplace Approximation
- Markov Chain Monte Carlo



(Quiz) But first... what is a statistical inference? (Hint: there are 2 keywords)

# What is a Statistical Inference?

“Using *data* analysis to deduce *properties* of underlying *probability distribution*.”

-Upton, G., Cook, I. (2008) *Oxford Dictionary of Statistics*, OUP.

images & labels for classification

$$\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$$

parametric model with independence assumption

$$\{y_n\}_{n=1}^N | \{\mathbf{x}_n\}_{n=1}^N \sim \prod_{n=1}^N p_{\theta}(y_n | \mathbf{x}_n)$$

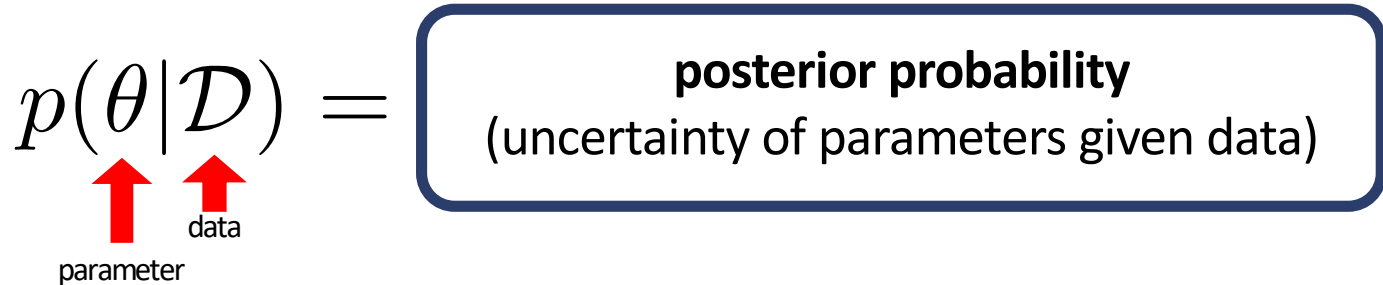
the “property”

## What is Bayesian Inference?

---

- Modeling **uncertainty/degree of belief** of parameters as **probability** given data.

$$p(\theta | \mathcal{D}) = \text{posterior probability} \\ \text{(uncertainty of parameters given data)}$$



- Remarks:**

- Bayesian inference is **NOT** about treating parameter as random variable.
- Instead, probability represents **degree of belief** or **uncertainty** on that value.
- Such interpretation of probability is called **Bayesian probability**.

## What is Bayesian Inference?

- Modeling **uncertainty/degree of belief** of parameters as **probability** given data.
  - Bayes' rule** is essential for description of posterior:

$$p(\theta | \mathcal{D}) = \frac{p(\theta, \mathcal{D})}{p(\mathcal{D})} = \frac{p(\mathcal{D} | \theta)p(\theta)}{p(\mathcal{D})}$$

↑      ↑  
parameter      data

$$\propto p(\mathcal{D} | \theta)p(\theta)$$



**likelihood**

(How likely is the data given parameter value?)

**prior probability**

(Initial belief on the parameter values)

Why do we care about the posterior?

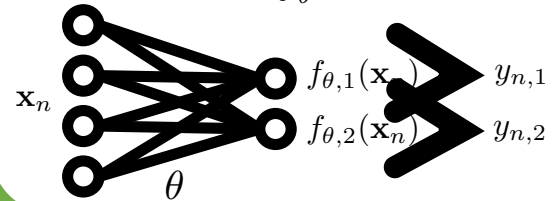
- **Bayesian prediction** for classification problem:

- What is test label, given test image, training image and training label?

$$\begin{aligned}
 & p(\mathcal{Y}^{(\text{test})} | \mathcal{X}^{(\text{test})}, \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})}) \\
 &= \frac{p(\mathcal{Y}^{(\text{test})}, \mathcal{Y}^{(\text{train})} | \mathcal{X}^{(\text{test})}, \mathcal{X}^{(\text{train})})}{p(\mathcal{Y}^{(\text{train})} | \mathcal{X}^{(\text{train})})} \\
 &= \int_{\theta} \frac{p(\mathcal{Y}^{(\text{test})}, \mathcal{Y}^{(\text{train})} | \mathcal{X}^{(\text{test})}, \mathcal{X}^{(\text{train})}, \theta) p(\theta)}{p(\mathcal{Y}^{(\text{train})} | \mathcal{X}^{(\text{train})})} \\
 &= \int_{\theta} \frac{p(\mathcal{Y}^{(\text{test})} | \mathcal{X}^{(\text{test})}, \theta) p(\mathcal{Y}^{(\text{train})} | \mathcal{X}^{(\text{train})}, \theta) p(\theta)}{p(\mathcal{Y}^{(\text{train})} | \mathcal{X}^{(\text{train})})} \\
 &= \int_{\theta} p(\mathcal{Y}^{(\text{test})} | \mathcal{X}^{(\text{test})}, \theta) p(\theta | \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})}) \\
 &= \mathbb{E}_{\theta \sim p(\theta | \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})})} [p(\mathcal{Y}^{(\text{test})} | \mathcal{X}^{(\text{test})}, \theta)]
 \end{aligned}$$

parametric modeling ( e.g.. neural network)

$$p(\mathcal{Y} | \mathcal{X}) = \int_{\theta} p(\mathcal{Y} | \mathcal{X}, \theta) p(\theta)$$



independence assumption

$$p(\{y_n\}_{n=1}^N | \{\mathbf{x}_n\}_{n=1}^N, \theta) = \prod_{n=1}^N p(y_n | \mathbf{x}_n, \theta)$$

$$p(\theta | \mathcal{X}, \mathcal{Y}) = \frac{p(\mathcal{Y} | \mathcal{X}, \theta) p(\theta | \mathcal{X})}{p(\mathcal{Y} | \mathcal{X})}$$

$$= \frac{p(\mathcal{Y} | \mathcal{X}, \theta) p(\theta)}{p(\mathcal{Y} | \mathcal{X})}$$

Bayes' rule

prediction requires  
sampling from posterior

Next, using neural network for Bayesian prediction



### **1. Introduction**

- What is Bayesian inference?
- What is Bayesian neural network?

### **2. Variational Inference for Bayesian Neural Networks**

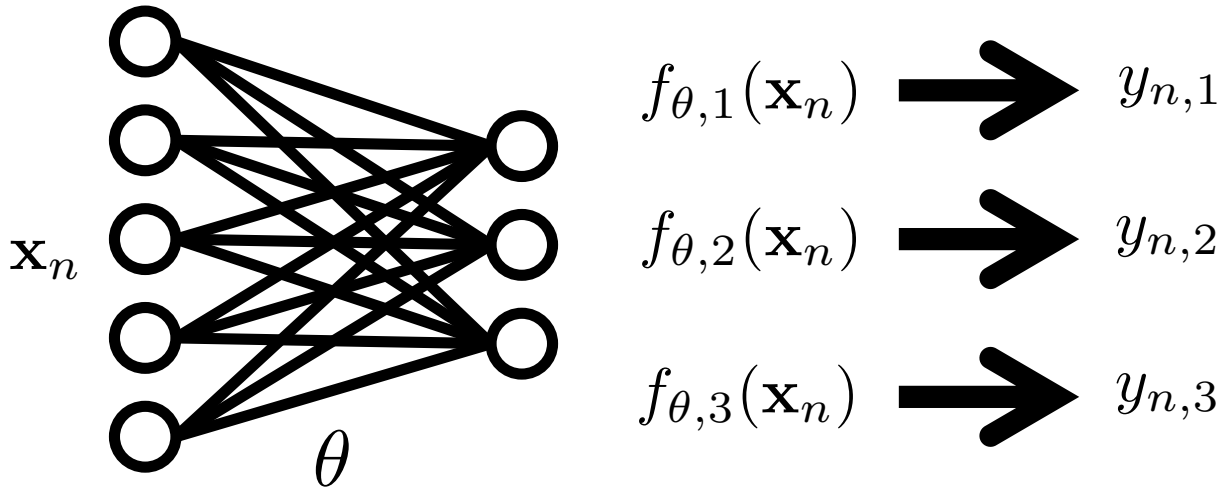
- Using Gaussian distribution
- Using multiplicative normalizing flows

### **3. Non-variational Inference for Bayesian Neural Networks**

- Laplace Approximation
- Markov Chain Monte Carlo

- **Bayesian neural network** is a neural network with **prior** on its weights.
  - Bayesian inference cannot be applied without priors.
- Two choices to make (i.e., choosing our models):
  1. **Log-likelihood** is expressed by neural networks:

$$\log p(\mathcal{Y}|\mathcal{X}, \theta) = \sum_{n=1}^N \log p(y_n|\mathbf{x}_n, \theta) = \sum_{n=1}^N \sum_{c \in \text{labels}} y_{n,c} \log f_{\theta,c}(\mathbf{x}_n)$$



- **Bayesian neural network** is a neural network with **prior** on its weights.
  - Bayesian inference cannot be applied without priors.
- Two choices to make (i.e., choosing our models):
  1. **Log-likelihood** is expressed by neural networks:

$$\log p(\mathcal{Y}|\mathcal{X}, \theta) = \sum_{n=1}^N \log p(y_n|\mathbf{x}_n, \theta) = \sum_{n=1}^N \sum_{c \in \text{labels}} y_{n,c} \log f_{\theta,c}(\mathbf{x}_n)$$


2. **Log-prior** is decided by our belief on the behavior of parameters:

$$\log p(\theta) = \log \mathcal{N}(\theta|0, \mathbb{I}) = \sum_{k=1}^K \log \mathcal{N}(\theta_k|0, 1) = \sum_{k=1}^K \frac{\theta_k^2}{2} + C$$

(we believe the parameters to be **normally** distributed when data is unseen.)

- Then log-posterior is expressed as follows:

$$\log p(\theta|\mathcal{X}, \mathcal{Y}) = \sum_{n=1}^N \log p(y_n|\mathbf{x}_n, \theta) + \log p(\theta) - \log p(\mathcal{Y}|\mathcal{X})$$



log-likelihood                      log-prior                      normalization factor

- Maximum Likelihood Estimation (MLE) recovers **cross-entropy loss**.

$$\begin{aligned}\theta_{\text{MLE}} &= \arg \max_{\theta} \sum_{n=1}^N \log p(y_n|\theta, \mathbf{x}_n) \\ &= \arg \min_{\theta} - \sum_{n=1}^N \sum_{c \in \text{labels}} y_{n,c} \log f_{\theta}(\mathbf{x}_n)\end{aligned}$$

- Then log-posterior is expressed as follows:

$$\log p(\theta|\mathcal{X}, \mathcal{Y}) = \sum_{n=1}^N \log p(y_n|\mathbf{x}_n, \theta) + \log p(\theta) - \log p(\mathcal{Y}|\mathcal{X})$$

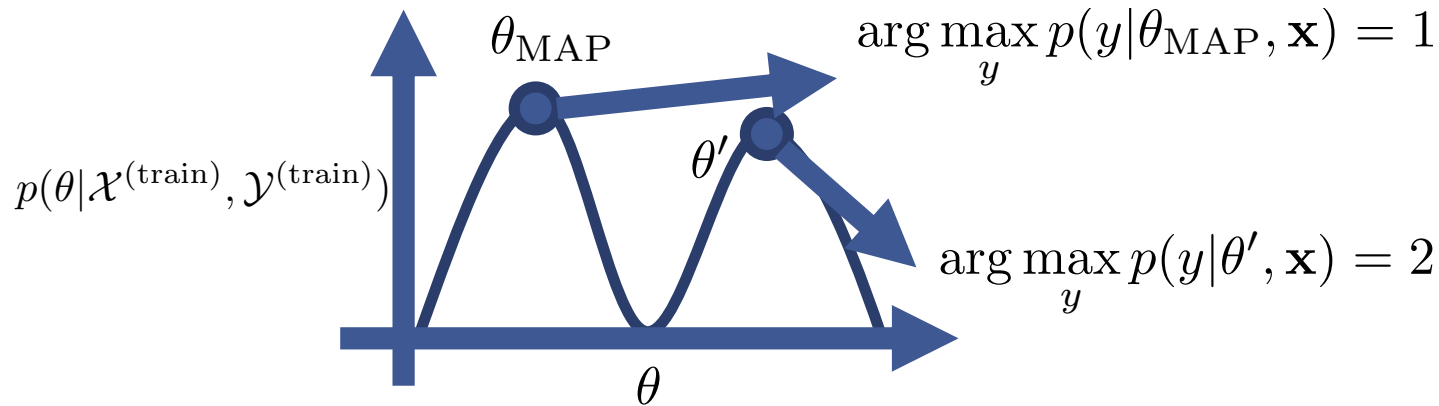
  
const. over parameter

- Maximum-a-Posteriori (MAP) recovers **cross-entropy** with **L2-regularization**.

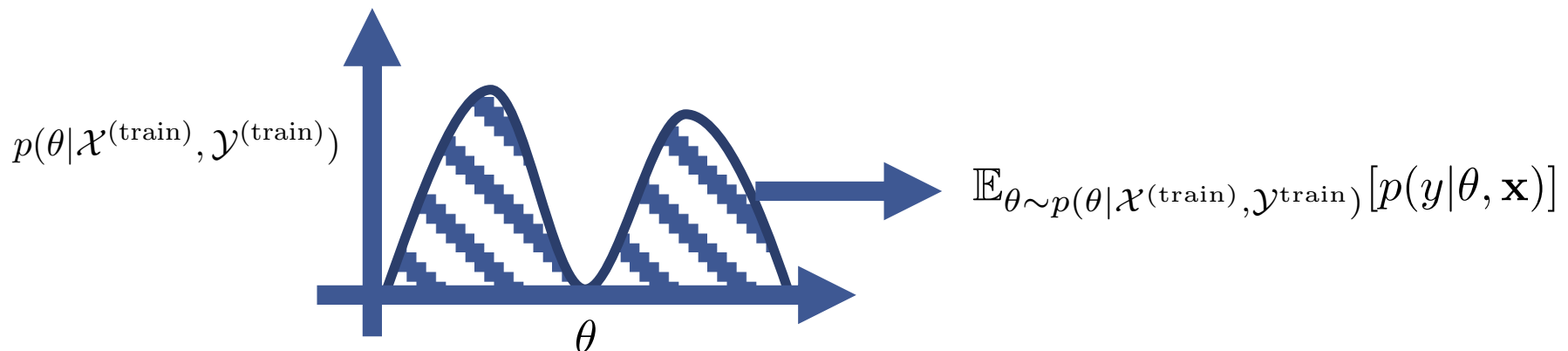
$$\begin{aligned}\theta_{\text{MAP}} &= \arg \max_{\theta} \sum_{n=1}^N \log p(y_n|\theta, \mathbf{x}_n) + \log p(\theta) \\ &= \arg \min_{\theta} - \sum_{n=1}^N \sum_{c \in \text{labels}} y_{n,c} \log f_{\theta}(\mathbf{x}_n) + \frac{\theta^2}{2}\end{aligned}$$

## MAP versus Bayesian Inference

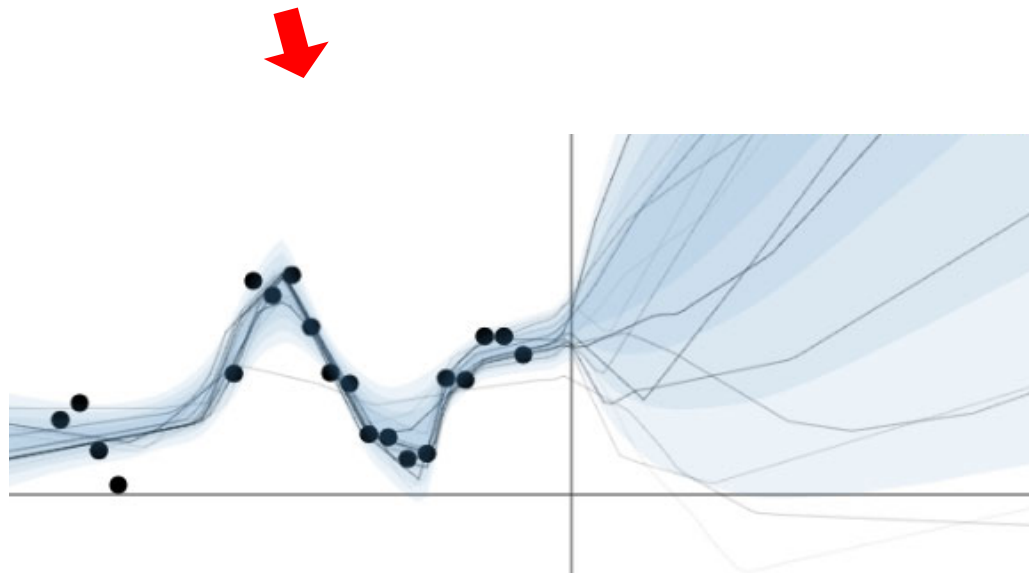
- Maximum-a-posteriori only considers a single point estimate:
  - Alternative parameter with similar score could exist.



- Bayesian inference allows to model 'uncertainty' over parameters:



- Bayesian NN is about modeling uncertainty in parameters.
- By modeling uncertainty, Bayesian NN provides:
  - Better **prediction accuracy** under same model.
  - Better **uncertainty estimation** for predictive distribution.



Given prediction of NN, how 'uncertain' are we on the expected performance?

## Difficulties of Bayesian Neural Network

- Bayesian NN lacks **scalability**, i.e., cannot be applied to large NNs in general:
  - Monte Carlo sampling is necessary for making predictions:

$$\begin{aligned} & p(\mathcal{Y}^{(\text{test})} | \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})}, \mathcal{X}^{(\text{test})}) \\ &= \mathbb{E}_{\theta \sim p(\theta | \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})})} [p(\mathcal{Y}^{(\text{test})} | \theta, \mathcal{X}^{(\text{test})})] \\ &\approx \frac{1}{S} \sum_{s=1}^S p(\mathcal{Y}^{(\text{test})} | \theta^{(s)}, \mathcal{X}^{(\text{test})}), \quad \theta^{(s)} \sim p(\theta | \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})}) \end{aligned}$$

- Furthermore, even sampling from the posterior is **intractable**.

$$p(\theta | \mathcal{X}, \mathcal{Y}) = \frac{p(\mathcal{Y} | \mathcal{X}, \theta) p(\theta)}{p(\mathcal{Y} | \mathcal{X})} = \frac{p(\mathcal{Y} | \mathcal{X}, \theta) p(\theta)}{\int_{\theta} p(\mathcal{Y} | \mathcal{X}, \theta) p(\theta)}$$

Intractable integration

- Instead, **approximate posterior distribution** can be used:

$$q(\theta) \approx p(\theta | \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})}), \quad \theta^{(s)} \sim q(\theta)$$

- This is called **approximate Bayesian inference**, or approximate Bayesian prediction.



- Approximate inference problem:

$$\begin{aligned} & p(\mathcal{Y}^{(\text{test})} | \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})}, \mathcal{X}^{(\text{test})}) \\ &= \mathbb{E}_{\theta \sim p(\theta | \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})})} [p(\mathcal{Y}^{(\text{test})} | \theta, \mathcal{X}^{(\text{train})})] \\ &\approx \frac{1}{S} \sum_{s=1}^S p(\mathcal{Y}^{(\text{test})} | \theta^{(s)}, \mathcal{X}^{(\text{test})}), \quad \theta^{(s)} \sim q(\theta) \approx p(\theta | \mathcal{X}^{(\text{train})}, \mathcal{Y}^{(\text{train})}) \end{aligned}$$



- Main obstacle: how to get the **approximate posterior**?
  1. **Variational inference (VI):**  
casting the inference / approximation as an optimization problem.
  2. **Laplace approximation:**  
pointwise estimation assisted with posterior curvature.
  3. **Markov chain Monte Carlo:**  
running Markov chains for Monte Carlo estimate of the posterior.

### 1. Introduction

- What is Bayesian inference?
- What is Bayesian neural network?

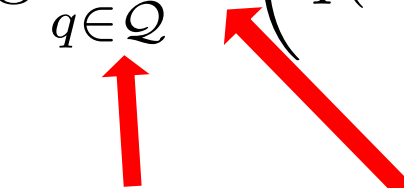
### 2. Variational Inference for Bayesian Neural Networks

- Using Gaussian distribution
- Using multiplicative normalizing flows

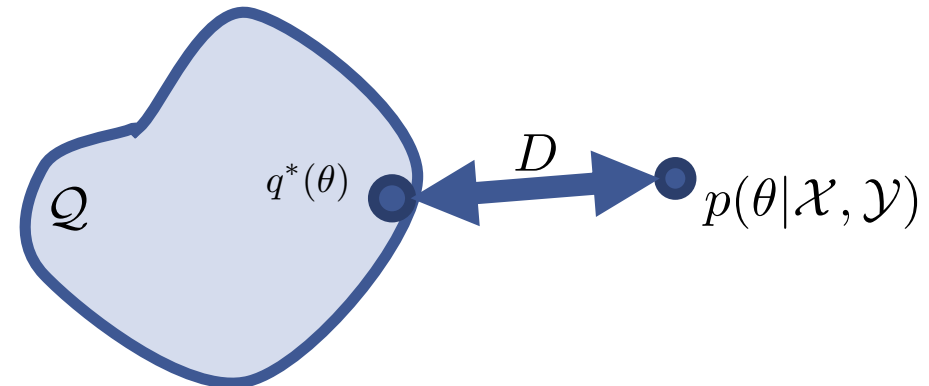
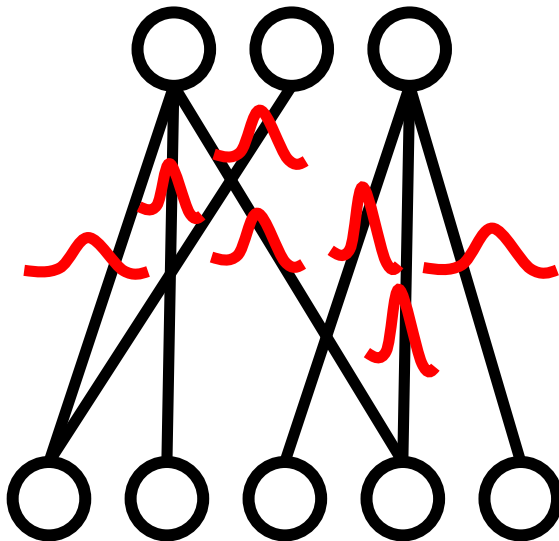
### 3. Non-variational Inference for Bayesian Neural Networks

- Laplace Approximation
- Markov Chain Monte Carlo

- Variational inference approximates posterior of Bayesian neural network by optimization:


$$q^* = \arg \min_{q \in \mathcal{Q}} D\left(q(\theta), p(\theta | \mathcal{X}, \mathcal{Y})\right)$$


- Choosing posterior from **certain family**, with **closest similarity** to exact posterior:



- Variational inference approximates posterior of Bayesian neural network by optimization:

$$q^* = \arg \min_{q \in \mathcal{Q}_{\text{FFG}}} D_{\text{KL}} \left( q(\theta), p(\theta | \mathcal{X}, \mathcal{Y}) \right)$$

  
fully factorized Gaussian                      KL divergence

- Next, we will study how to approximate the posterior by **fully factorized Gaussian (FFG)** [Blundel et al., 2015]:
  - Optimization with respect to **Kullback-Leibler (KL) divergence**.
  - Utilizing the **reparameterization trick** for efficient optimization of posterior approximations.

- We consider fully factorized Gaussians as **prior**:

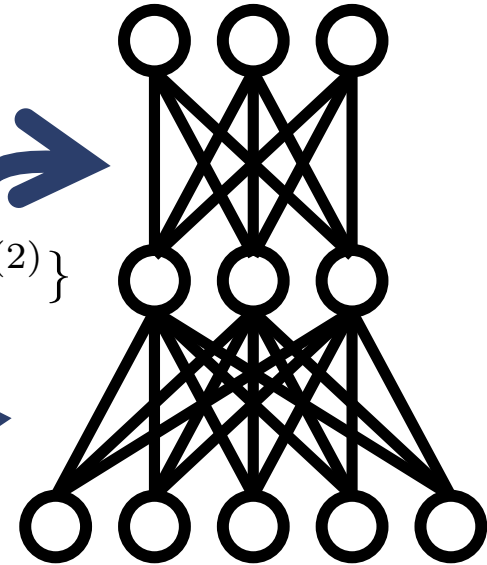
$$p(\theta) = \prod_{\ell=1}^L p(\mathbf{W}^{(\ell)})$$

elements of weight matrix

$$= \prod_{\ell=1}^L \prod_{i=1}^{D^{(\ell)}} \prod_{j=1}^{D^{(\ell-1)}} p(w_{ij}^{(\ell)})$$

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

$$p(w_{ij}^{(\ell)}) = \mathcal{N}(w_{ij}^{(\ell)}; 0, 1)$$



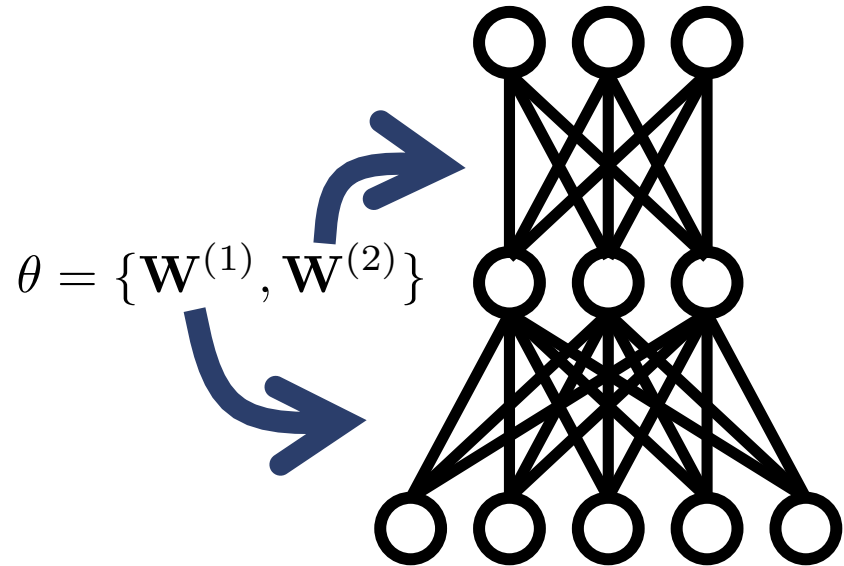
- Remark:** **exact** posterior is non-gaussian:

$$p(\theta | \mathcal{X}, \mathcal{Y}) \propto p(\mathcal{Y} | \mathcal{X}, \theta) p(\theta)$$

non-Gaussian   Gaussian

- We also consider fully factorized Gaussians as **approximate posterior** :

$$\begin{aligned} q(\theta) &= \prod_{\ell=1}^L q(\mathbf{W}^{(\ell)}) \\ &= \prod_{\ell=1}^L \prod_{i=1}^{D^{(\ell)}} \prod_{j=1}^{D^{(\ell-1)}} q(w_{ij}^{(\ell)}) \\ q(w_{ij}^{(\ell)}) &= \mathcal{N}(w_{ij}^{(\ell)}; \mu_{ij}^{(\ell)}, \sigma_{ij}^{(\ell)}) \end{aligned}$$



- We want to find parameters minimizing KL divergence to the exact posterior:

$$\phi = \{\mu_{ij}^{(\ell)}, \sigma_{ij}^{(\ell)}\}_{i,j,\ell}$$

$$\phi \leftarrow \arg \min_{\phi} D_{\text{KL}}(q_{\phi}(\theta), p(\theta | \mathcal{X}, \mathcal{Y}))$$

How to use gradient descent for optimization?

- Gradients for the KL divergence:

$$\begin{aligned} \nabla_{\phi} D_{\text{KL}}(q(\theta), p(\theta|\mathcal{X}, \mathcal{Y})) \\ = \nabla_{\phi} \mathbb{E}_{\theta \sim q_{\phi}(\theta)} [\log p(\mathcal{Y}|\mathcal{X}, \theta)] + \nabla_{\phi} D_{\text{KL}}(q_{\phi}(\theta), p(\theta)) \end{aligned}$$

2. Expected log-likelihood      1. KL-divergence to prior

- KL-divergence to prior (fixed form):

$$\nabla_{\phi} D_{\text{KL}}(q_{\phi}(\theta), p(\theta)) = \nabla_{\phi} \sum_{i,j,\ell} \left( \log \sigma_{i,j}^{(\ell)} + \frac{1 + \mu_{i,j}^{(\ell)}}{2\sigma_{i,j}^{(\ell)}} \right)$$

- Naïve estimation of expected log-likelihood:

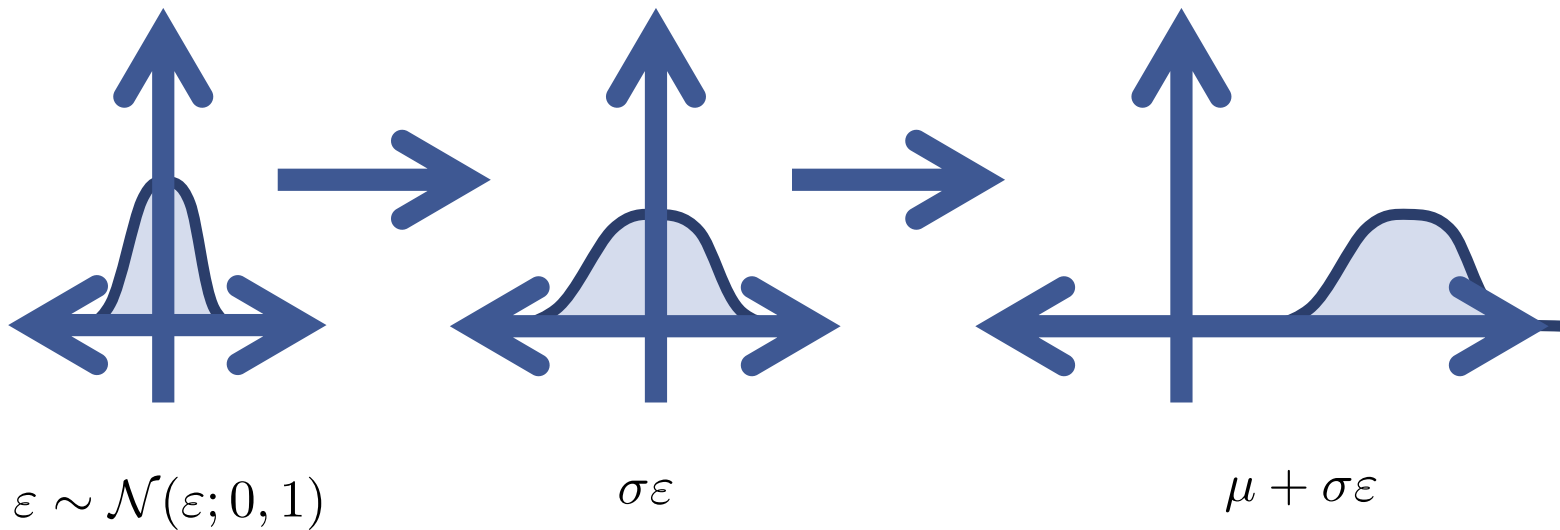
$$\nabla_{\phi} \mathbb{E}_{\theta \sim q_{\phi}(\theta)} [\log p(\mathcal{Y}|\mathcal{X}, \theta)] \approx \nabla_{\phi} \frac{1}{S} \sum_{s=1}^S \log p(\mathcal{Y}|\mathcal{X}, \theta^{(s)}), \quad \theta^{(s)} \sim q_{\phi}(\theta)$$

 zero gradient?

## The Reparameterization Trick

- Re-parameterizing random variables:

$$q(w_{ij}) = \mathcal{N}(w_{ij}; \mu_{ij}, \sigma_{ij}) \iff \begin{aligned} w_{ij} &= \mu_{ij} + \sigma_{ij}\varepsilon_{ij} \\ \varepsilon_{ij} &\sim \mathcal{N}(\varepsilon_{ij}; 0, 1) \end{aligned}$$





- Re-parameterizing random variables:

$$q(w_{ij}) = \mathcal{N}(w_{ij}; \mu_{ij}, \sigma_{ij}) \quad \longleftrightarrow \quad \begin{aligned} w_{ij} &= \mu_{ij} + \sigma_{ij}\varepsilon_{ij} \\ \varepsilon_{ij} &\sim \mathcal{N}(\varepsilon_{ij}; 0, 1) \end{aligned}$$

- Re-parameterized expected log-likelihood:

$$\begin{aligned} &\nabla_{\phi} \mathbb{E}_{\theta \sim q_{\phi}(\theta)} [\log p(\mathcal{Y} | \mathcal{X}, \theta)] \\ &\approx \nabla_{\phi} \frac{1}{S} \sum_{s=1}^S \log p(\mathcal{Y} | \mathcal{X}, \theta^{(s)}), \quad \theta^{(s)} \sim q_{\phi}(\theta) \\ &= \nabla_{\phi} \frac{1}{S} \sum_{s=1}^S \log p(\mathcal{Y} | \mathcal{X}, f_{\phi}(\varepsilon^{(s)})), \quad \varepsilon^{(s)} \sim \mathcal{N}(\varepsilon_{i,j}; 0, 1) \end{aligned}$$



differentiable!

Backpropagation for non-Bayesian neural network:

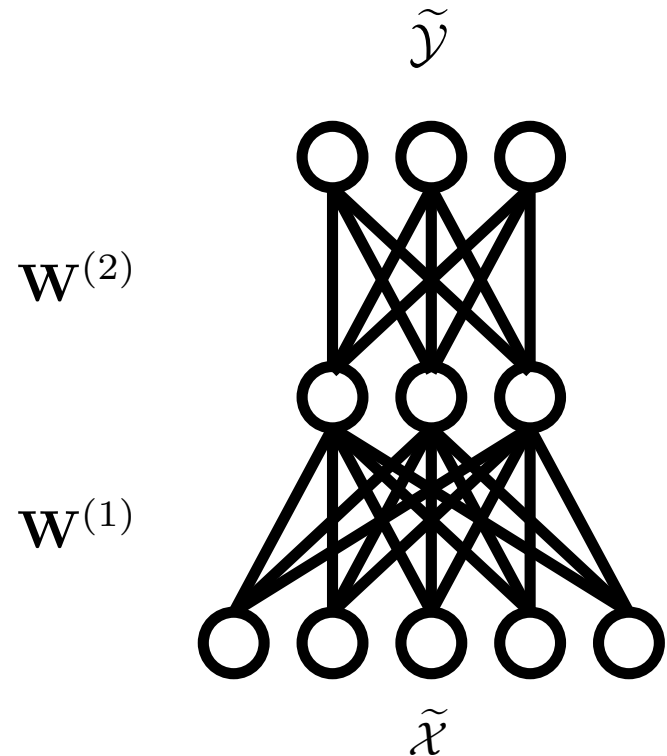
- For each step, perform gradient descent:

1. Sample mini-batch  $\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}$ .
2. Compute expected likelihood:

$$\mathcal{L}(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}) \leftarrow \log p(\tilde{\mathcal{Y}}|\tilde{\mathcal{X}}, \theta)$$

3. Back-propagate gradients.
4. Do gradient descent:

$$w_{ij}^{(\ell)} \leftarrow w_{ij}^{(\ell)} + \lambda \nabla_{w_{ij}^{(\ell)}} \mathcal{L}(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}})$$



Backpropagation for non-Bayesian neural network:

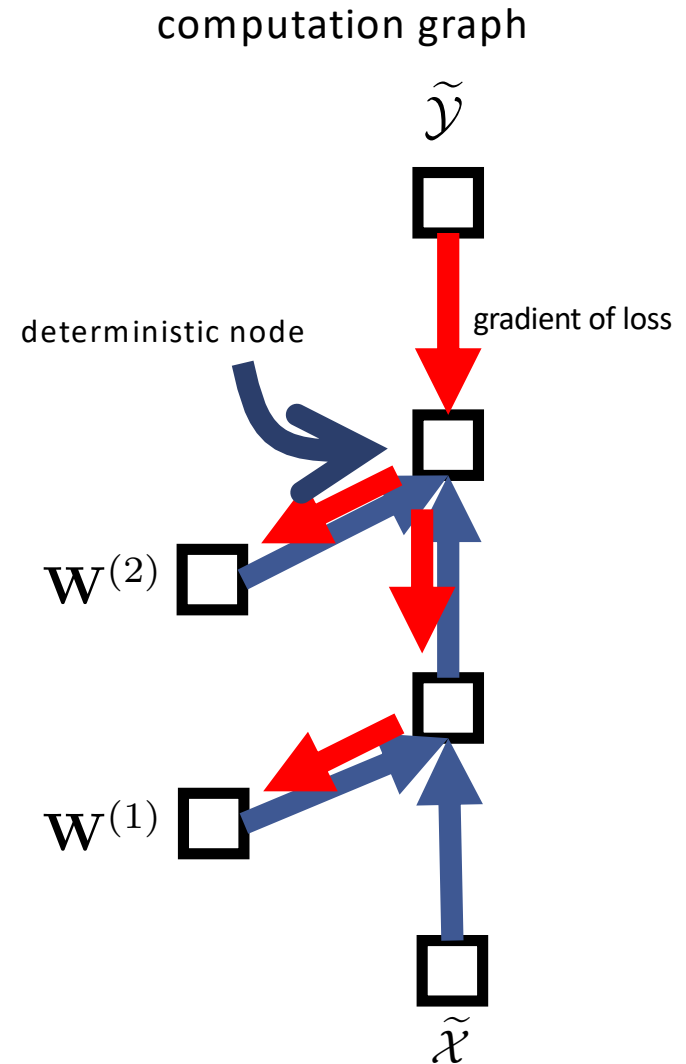
- For each step, perform gradient descent:

1. Sample mini-batch  $\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}$ .
2. Compute expected likelihood:

$$\mathcal{L}(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}) \leftarrow \log p(\tilde{\mathcal{Y}}|\tilde{\mathcal{X}}, \theta)$$

3. Back-propagate gradients.
4. Do gradient descent:

$$w_{ij}^{(\ell)} \leftarrow w_{ij}^{(\ell)} + \lambda \nabla_{w_{ij}^{(\ell)}} \mathcal{L}(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}})$$



### Backpropagation for Bayesian neural network:

- For each step, perform gradient descent:

1. Sample mini-batch  $\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}$  and noise  $\varepsilon$ .

2. Compute weights:

$$w_{ij}^{(\ell)} \leftarrow \mu_{ij}^{(\ell)} + \sigma_{ij}^{(\ell)} \varepsilon_{ij}^{(\ell)}$$

3. Compute KL divergence (usually  $S = 10$ ):

$$\mathcal{L}(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}, \varepsilon) = \frac{1}{S} \sum_{s=1} \log p(\tilde{Y} | \tilde{\mathcal{X}}, \theta^{(s)}) + D_{\text{KL}}(q_{\phi}(\theta), p(\theta))$$

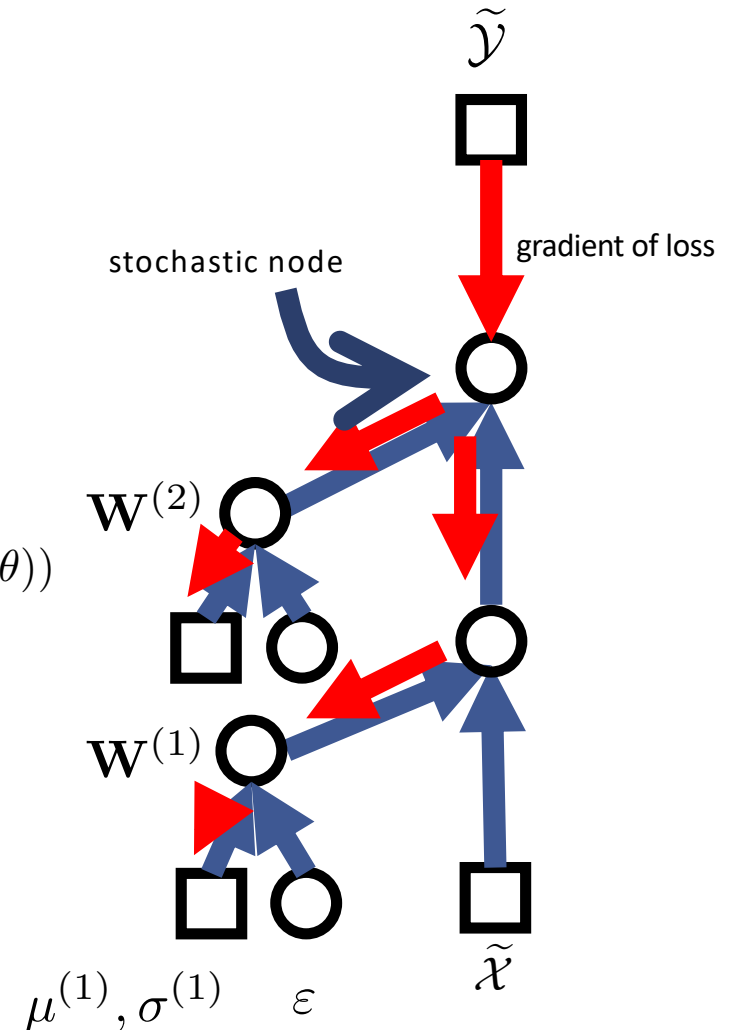
4. Back-propagate gradients.

5. Do gradient descent:

$$\mu_{ij}^{(\ell)} \leftarrow \mu_{ij}^{(\ell)} + \lambda \nabla_{\mu_{ij}^{(\ell)}} \mathcal{L}(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}, \varepsilon)$$

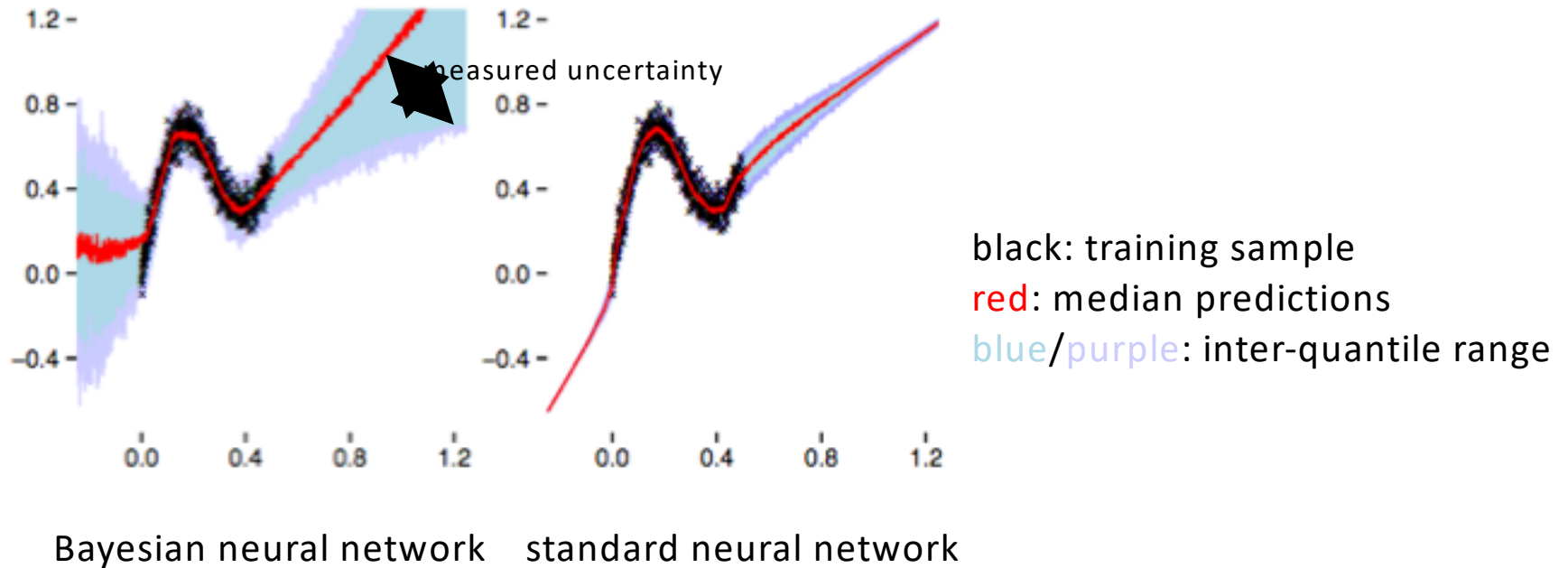
$$\sigma_{ij}^{(\ell)} \leftarrow \sigma_{ij}^{(\ell)} + \lambda \nabla_{\sigma_{ij}^{(\ell)}} \mathcal{L}(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}, \varepsilon)$$

### stochastic computation graph



## Experimental Results

- Regression task for toy data:



## Experimental Results

- MNIST classification using fully connected neural network:

Method	# Units/Layer	# Weights	Test Error
SGD, no regularisation (Simard et al., 2003)	800	1.3m	1.6%
SGD, dropout (Hinton et al., 2012)			≈ 1.3%
SGD, dropconnect (Wan et al., 2013)	800	1.3m	<b>1.2%*</b>
SGD	400	500k	1.83%
	800	1.3m	1.84%
	1200	2.4m	1.88%
SGD, dropout	400	500k	1.51%
	800	1.3m	1.33%
	1200	2.4m	1.36%
Bayes by Backprop, Gaussian	400	500k	1.82%
	800	1.3m	1.99%
	1200	2.4m	2.04%
Bayes by Backprop, Scale mixture	400	500k	1.36%
	800	1.3m	1.34%
	1200	2.4m	<b>1.32%</b>

simple extension from Gaussian  
using multiple noise source



Can we train more flexible (expressive) distribution for weights?

### **1. Introduction**

- What is Bayesian inference?
- What is Bayesian neural network?

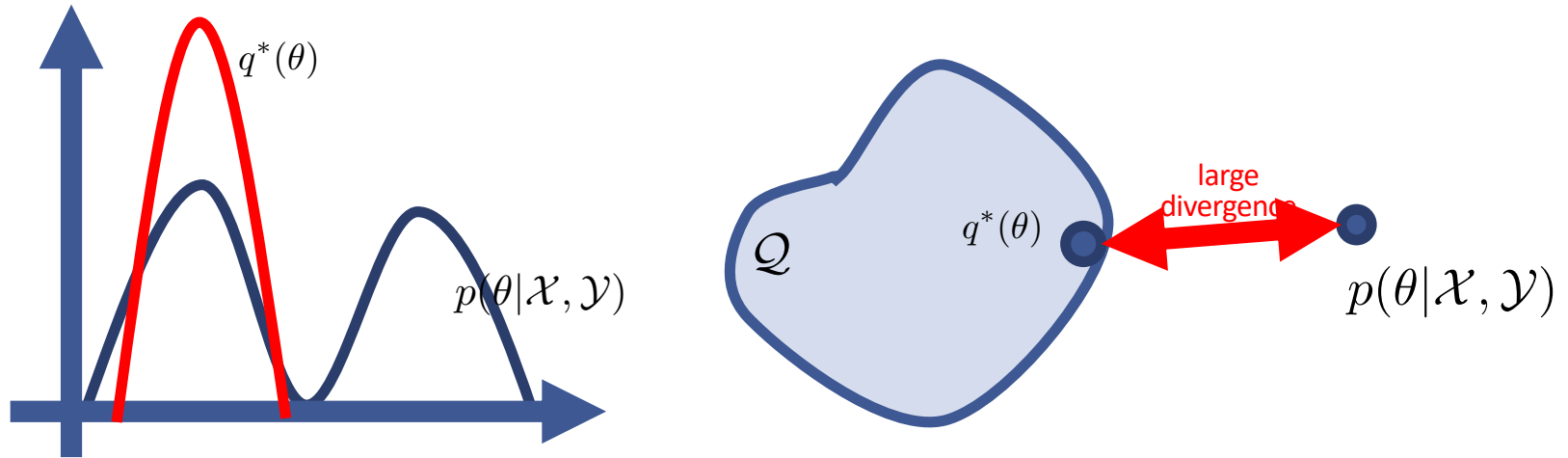
### **2. Variational Inference for Bayesian Neural Networks**

- Using Gaussian distribution
- Using multiplicative normalizing flows

### **3. Non-variational Inference for Bayesian Neural Networks**

- Laplace Approximation
- Markov Chain Monte Carlo

- Fully factorized Gaussian has limited expressive power, e.g., can only adjust to a single mode of the posterior.



- Multiplicative normalizing flow (MNF)** [Louizos et al., 2017] can be used to replace FFG for more expressive power:

$$q^* = \arg \min_{q \in \mathcal{Q}_{\text{MNF}}} D_{\text{KL}} \left( q(\theta), p(\theta | \mathcal{X}, \mathcal{Y}) \right)$$



- **Normalizing flows** [Rezende et al., 2015] family of flexible and tractable distribution made by sequence of **invertible transformations**:

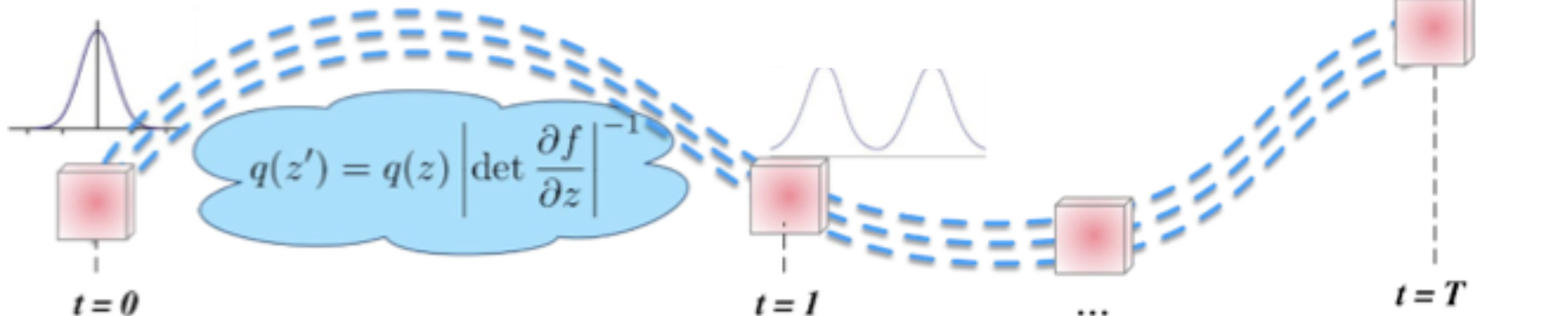
1. Sample initial distribution:  $\mathbf{z}_0 \sim q_0(\mathbf{z}_0)$
2. Warp the distribution through  $K$  invertible transformations:

$$\mathbf{z}_K = f_K \circ \dots \circ f_2 \circ f_1(\mathbf{z}_0)$$

3. Final variable  $\mathbf{z}_K$  is expressed as follows:

How to parameterize invertible transformations?

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \sum_{k=1}^K \log \det \left| \frac{\partial f_k}{\partial \mathbf{z}_k} \right|$$



- Parameterizing invertible transformations:

$$\mathbf{z}_K = f_K \circ \dots \circ f_2 \circ f_1(\mathbf{z}_0)$$

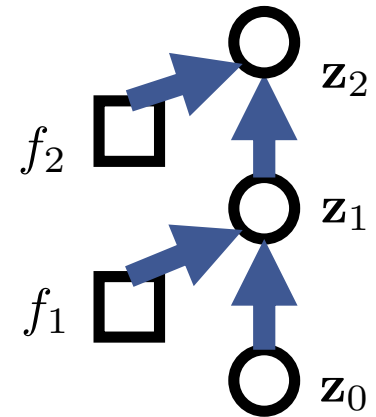
- Naïve (invertible) linear transformation:

$$\mathbf{z}_i = f_i(\mathbf{z}_{i-1}) = \mathbf{A}_i \mathbf{z}_{i-1}$$

- Planar flows [Rezende et al., 2015] with **non-linearity**:

$$f_i(\mathbf{z}_{i-1}) = \mathbf{z}_{i-1} + \mathbf{u}_i h(\mathbf{v}_i^\top \mathbf{z}_{i-1} + b_i)$$

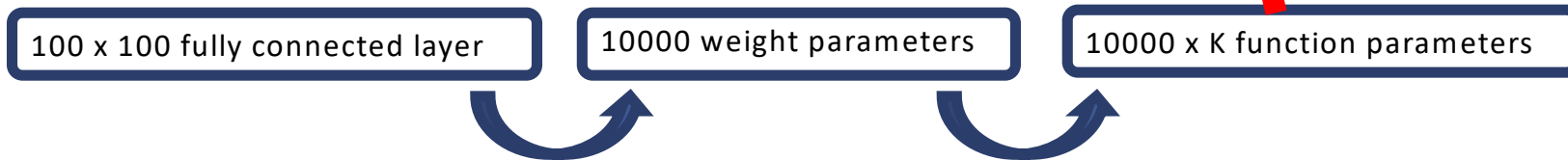
$\mathbf{u}_i$ : vector with same dimension as  $\mathbf{z}_{i-1}$   
 $h$ : non-linear activation function



- More advanced flows like **inverse autoregressive flow** [Kingma et al., 2016] exist.

## Multiplicative Normalizing Flows for Weight Generation

- Using normalizing flow for weights, e.g.,  $\mathbf{W}_K = f_K \circ \dots \circ f_1(\mathbf{W}_0)$  ?
  - However, weights are too high-dimensional for modeling...



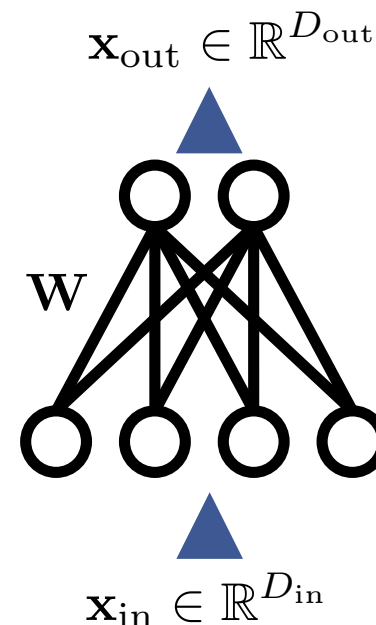
- Generation of NN weights from “multiplicative noise” [Louizos et al., 2017]:
  - Noise is sampled from normalizing flow:

$$\mathbf{z} = f_K \circ \dots \circ f_2 \circ f_1(\mathbf{z}_0) \quad \mathbf{z}_0 \sim q_0(\mathbf{z}_0)$$

- Generate weights by multiplication:

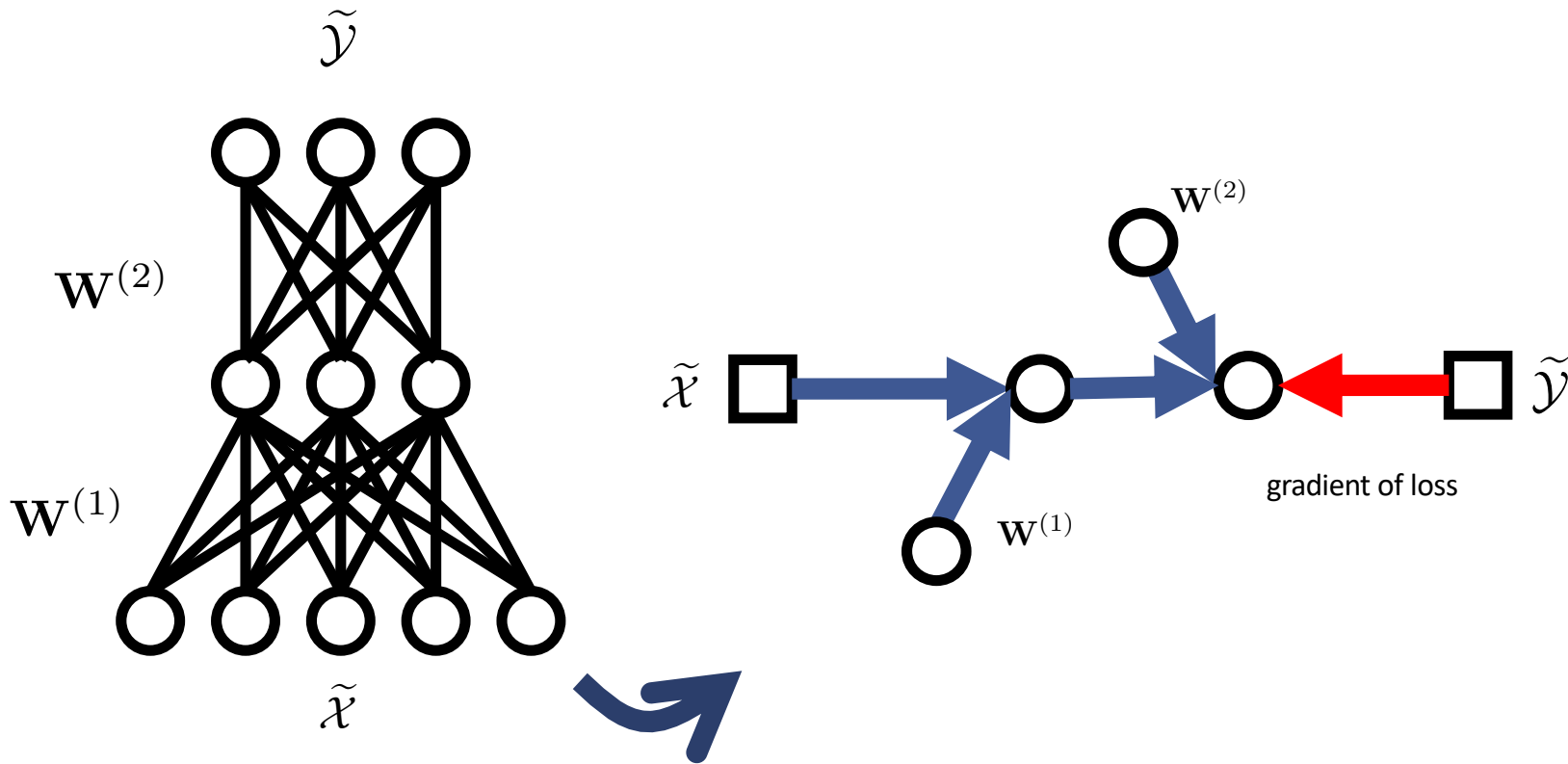
$$q(\mathbf{W}|\mathbf{z}) = \prod_{i=1}^{D_{in}} \prod_{j=1}^{D_{out}} \mathcal{N}(z_i \mu_{ij}, \sigma_{ij}^2)$$

Only requires  $D_{in} \times K$  function parameters



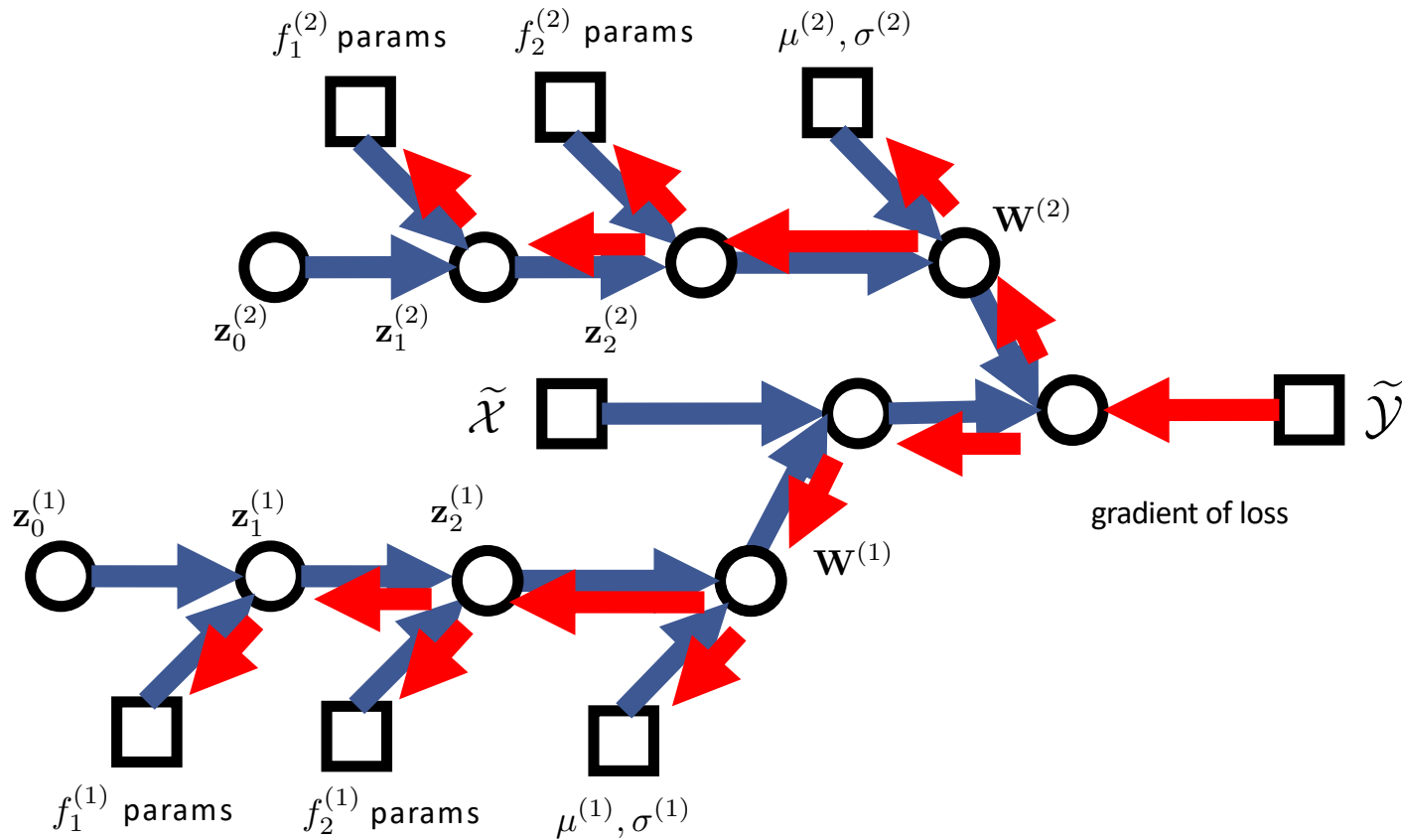
## Optimization by Stochastic Computational Graph

- For training, minimizing KL-divergence to the posterior.
- Again, weight is optimized from building stochastic computation graph.



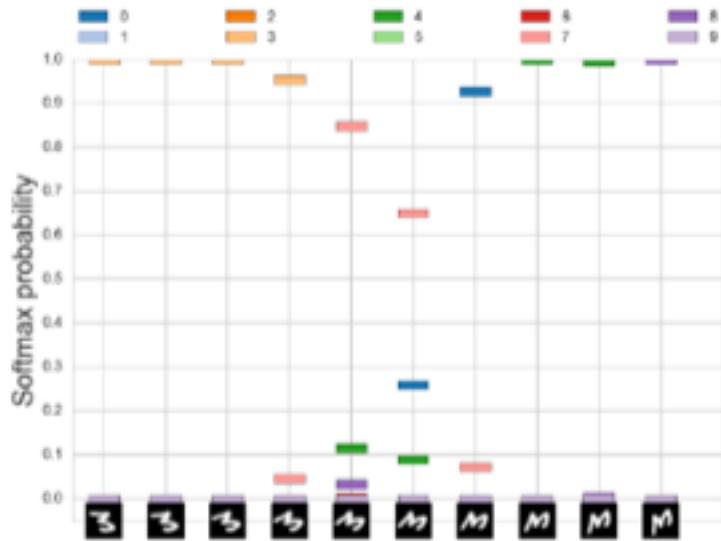
## Optimization by Stochastic Computational Graph

- For training, minimizing KL-divergence to the posterior.
- Again, weight is optimized from building stochastic computation graph.

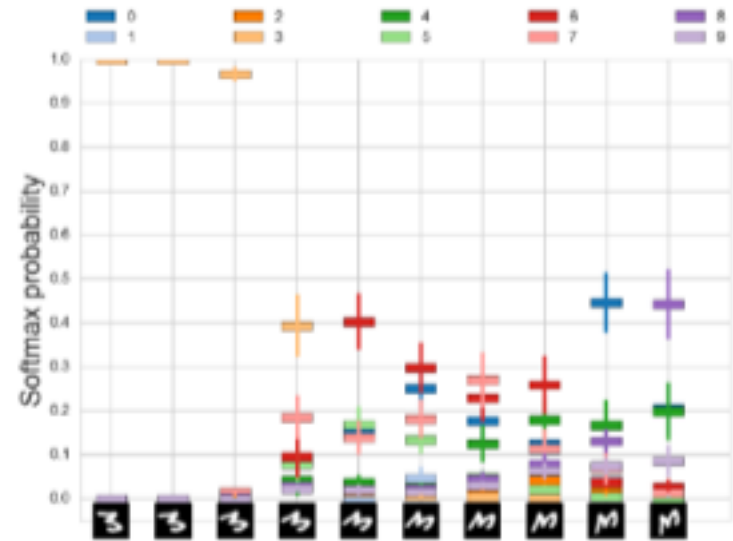


## Experimental Results

- Predictive uncertainty estimation for rotation of MNIST



(a) LeNet with weight decay



(b) LeNet with multiplicative normalizing flows

- Each color corresponds to a different class.
- Each bar denotes the assigned probability by the NN.

## Experimental Results

- Predictive uncertainty estimation for unobserved dataset:

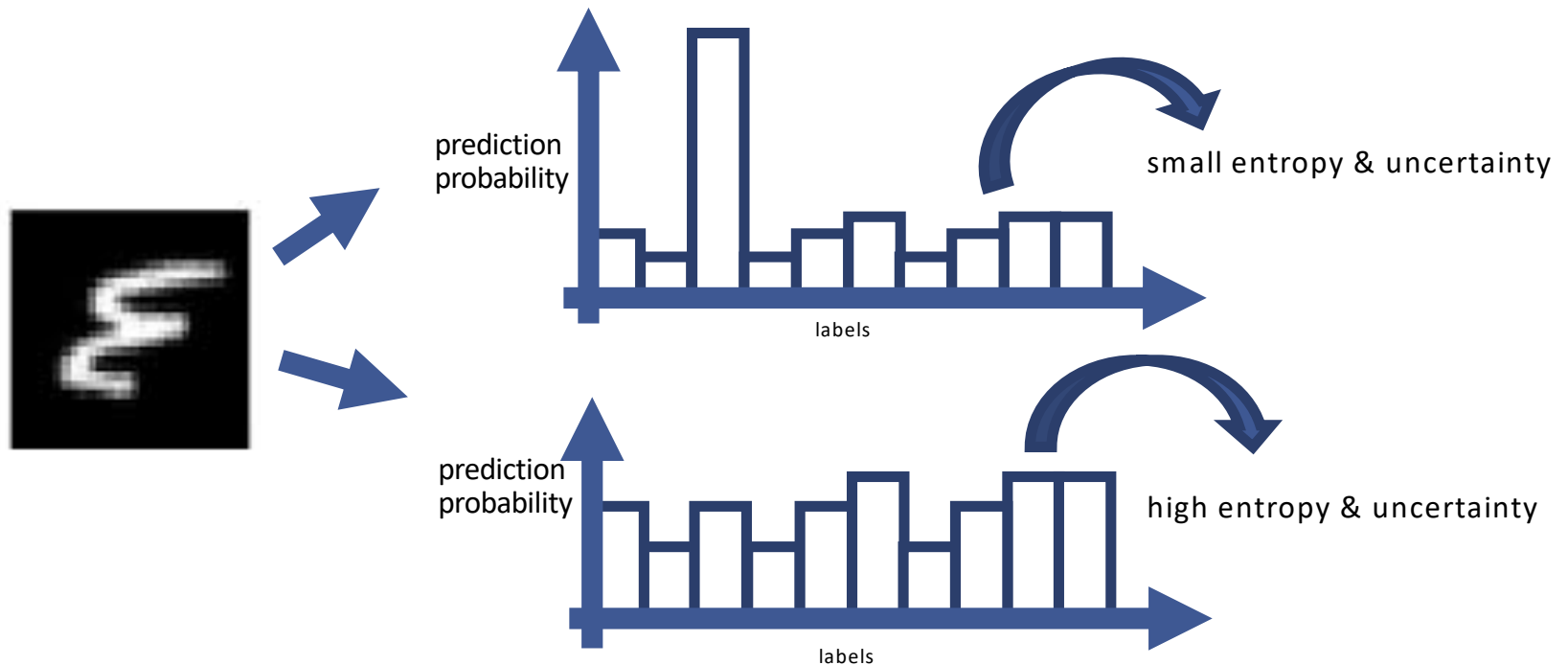
- Train model on CIFAR-10 using only 5 out of 10 classes.

test errors for each training methods using LeNet-5 architecture

Dataset	L2	Dropout	D.Ensem.	FFG	FFLU	MNFG
CIFAR 5	24	16	21	22	23	16

proposed algorithm

- Compute **entropy** ( $H(y) = -\sum_y p(y) \log p(y)$ ) for rest of 5 (unobserved) classes.



## Experimental Results

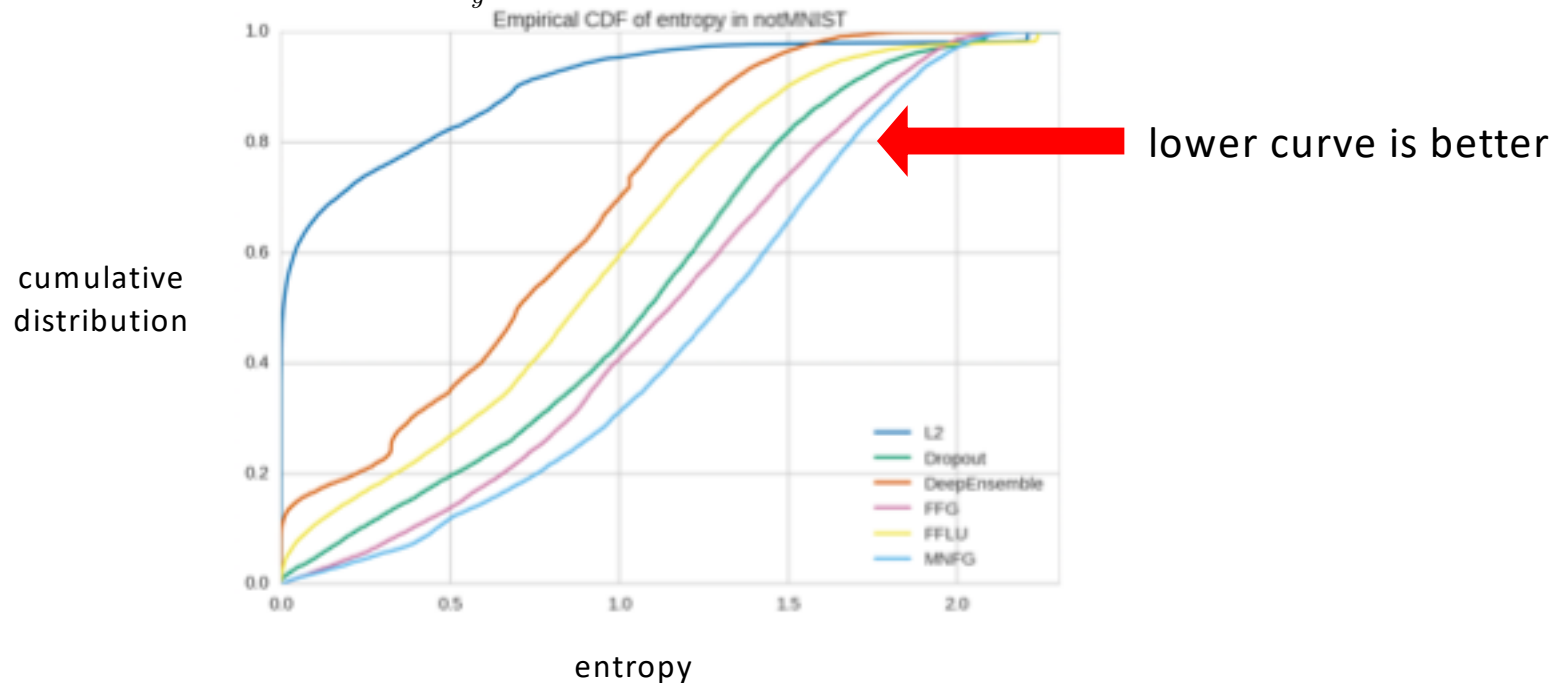
- Predictive uncertainty estimation for unobserved dataset:
  1. Train model on CIFAR-10 using only 5 out of 10 classes.

test errors for each training methods using LeNet-5 architecture

Dataset	L2	Dropout	D.Ensem.	FFG	FFLU	MNFG
CIFAR 5	24	16	21	22	23	16

proposed algorithm

2. Compute **entropy** ( $H(y) = -\sum_y p(y) \log p(y)$ ) for rest of 5 (unobserved) classes.





### **1. Introduction**

- What is Bayesian inference?
- Why use Bayesian neural networks?

### **2. Variational Inference for Bayesian Neural Networks**

- Using Gaussian distribution
- Using multiplicative normalizing flow

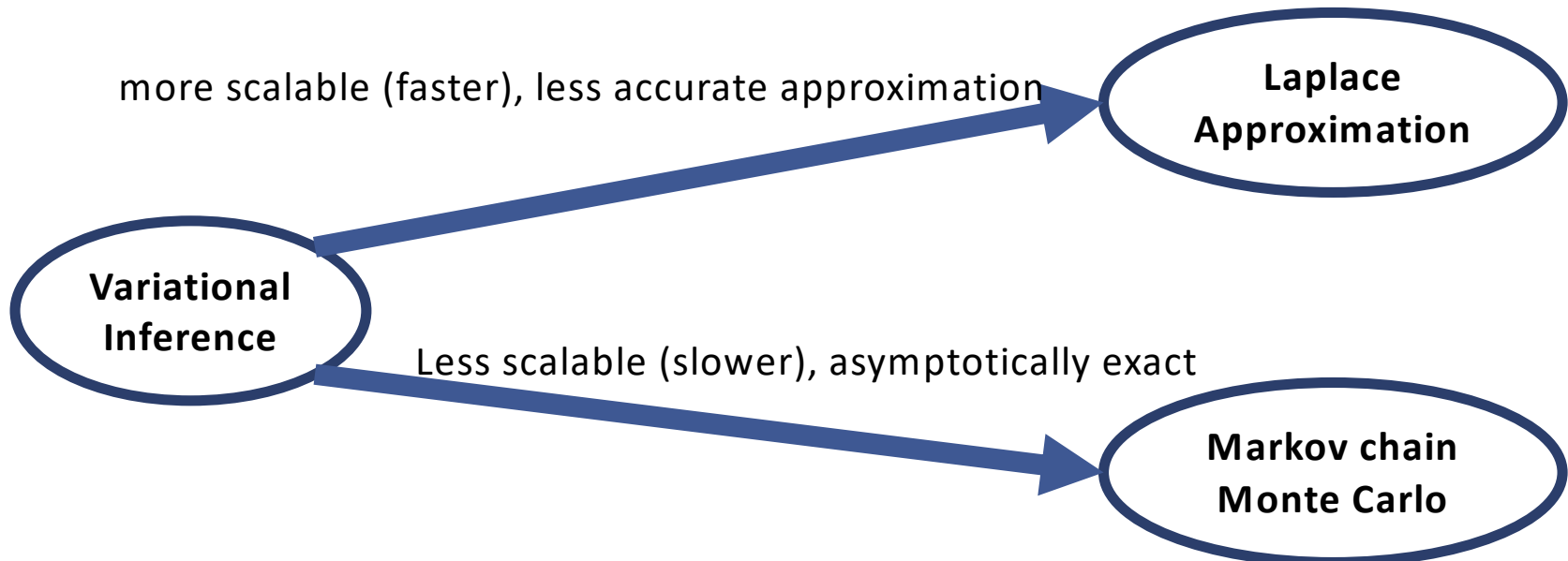
### **3. Non-variational Inference for Bayesian Neural Networks**

- Laplace approximation
- Markov Chain Monte Carlo

- So far, variational inference (VI) for approximating the posterior:

$$q(\theta) \approx p(\theta|\mathcal{X}, \mathcal{Y}) \quad \Rightarrow \quad q^* = \arg \min_{q \in \mathcal{Q}} D\left(q(\theta), p(\theta|\mathcal{X}, \mathcal{Y})\right)$$

- In this section, we describe two alternatives for VI:

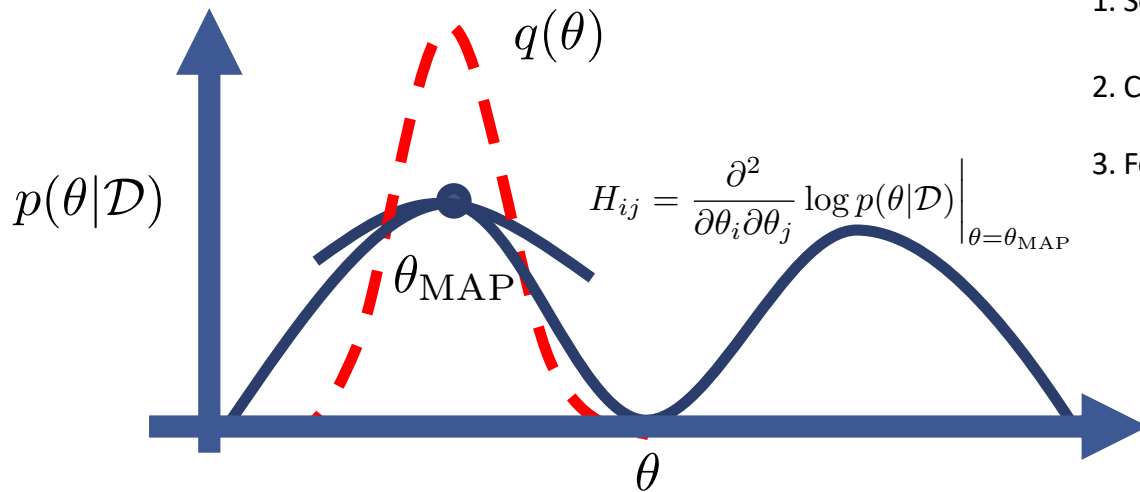


- **Laplace approximation** [MacKay, 1992]: pointwise estimation assisted with posterior curvature.

$$\log p(\theta|\mathcal{D}) \approx \log p(\theta_{\text{MAP}}|\mathcal{D}) - \frac{1}{2}(\theta - \theta_{\text{MAP}})^\top H(\theta - \theta_{\text{MAP}})$$

where  $H$  is the **Hessian** (second order derivative) of log-posterior.

- Equivalent to placing a Gaussian distribution with MAP estimation as mean.



1. Solve the MAP estimation of neural network.
2. Compute the Hessian of the posterior distribution.
3. Form the corresponding second-order approximation

However, Hessian requires  $O(D^2)$  (**too high-cost**) computation for  $\theta \in \mathbb{R}^D$ .

- Hessian is **too high-cost** to compute, so **diagonal** [Lecun et al., 1990] and **Kronecker product** [Ritter et al., 2017] are used approximate the Hessian.

- Diagonal approximation for the Hessian:

$$H = H_{\log\text{-likelihood}} + H_{\text{prior}} \quad \leftarrow \text{easily computable}$$
$$\approx \underline{-\text{diag}([g_1^2, \dots, g_D^2])} + H_{\text{prior}} \quad g_i = \mathbb{E}_{\mathcal{D} \sim p(\mathcal{D}|\theta)} \left[ \frac{\partial}{\partial \theta_i} \log p(\mathcal{D}|\theta) \right]$$

Diagonal of negative Fisher information matrix  $F = \mathbf{g}\mathbf{g}^\top$ .

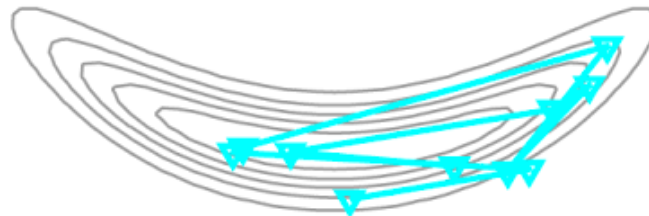
- This approximation reduce the complexity by  $O(D^2) \rightarrow O(D)$ .
- **Remark:** Fisher information matrix is average of  $H_{\log\text{-likelihood}}$  with respect to expectation over log-likelihood, i.e.,

$$F_{ij} = g_{ij}g_j = -\mathbb{E}_{\mathcal{D} \sim p(\mathcal{D}|\theta)} \left[ \frac{\partial^2}{\partial \theta_i \partial \theta_j} \log p(\mathcal{D}|\theta) \right]$$

- **Markov chain Monte Carlo (MCMC)**: running Markov chains for direct sampling of the **exact** posterior  $\theta \sim p(\theta|\mathcal{D})$ .
- **Stochastic Langevin gradient dynamics** [Welling et al., 2011] can be used to sample from the log posterior of the neural network:
  1. Initialize the parameter with  $\theta \leftarrow \theta_0$
  2. At each steps of Markov chain, do a **noisy gradient update**:

$$\theta \leftarrow \theta - \eta \left( \frac{\partial}{\partial \theta} \log p(\theta|\mathcal{D}) + \epsilon \right)$$

3. After repeating  $T$  steps, sample  $\theta$  from the posterior is obtained.



- Bayesian methods provide a probabilistic perspective for the uncertainty of NN.
  - It provides better prediction and estimate of uncertainty.
  - Efficient approximation of posterior is important for good performance.

## Additional Interesting Materials

- Deterministic NN regularizers can be re-interpreted as Bayesian inference.
  - **Dropout**: Gal et al., Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning, ICML 2016
  - **Batch Normalization**: Teye et al., Bayesian Uncertainty Estimation for Batch Normalized Deep Networks, ICML 2018
- Bayesian framework introduce new perspective for existing tasks.
  - **Compression**: Louizos et al., Bayesian Compression for Deep Learning, NIPS 2017
  - **Continual learning**: Nguyen et al., Variational Continual Learning, ICLR 2018

## References

---

[Blundel, 2015] Weight Uncertainty in Neural Networks, ICML 2015

link: <https://arxiv.org/abs/1505.05424>

[Louizos, 2017] Multiplicative Normalizing Flows for Variational Bayesian Neural Networks, ICML 2017

link: <https://arxiv.org/abs/1703.01961>

[Ritter et al., 2017] A Scalable Laplace Approximation for Neural Networks, ICLR 2017

link: <https://openreview.net/pdf?id=Skdyd2xAZ>

[Welling et al., 2011] Bayesian Learning via Stochastic Gradient Langevin Dynamics, ICML 2011

link: [https://www.ics.uci.edu/~welling/publications/papers/stoclangevin\\_v6.pdf](https://www.ics.uci.edu/~welling/publications/papers/stoclangevin_v6.pdf)

[Gal et al., 2016] Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning, ICML 2016

link: <https://arxiv.org/abs/1506.02142>

[Teye et al., 2018] Bayesian Uncertainty Estimation for Batch Normalized Deep Networks, ICML 2018

link: <https://arxiv.org/abs/1802.06455>

[Louizos et al., 2017] Bayesian Compression for Deep Learning, NIPS 2017

link: <https://arxiv.org/abs/1705.08665>

[Nguyen et al., 2018] Variational Continual Learning, ICLR 2018

link: <https://arxiv.org/abs/1710.10628>

[Kleijn et al., 2012] The Bernstein-Von-Mises Theorem under Misspecification, EJS 2012

link: <https://projecteuclid.org/euclid.ejs/1332162333>

[Kingma et al., 2015] Variational Dropout and Local Reparameterization Trick, NIPS 2015

link: <https://arxiv.org/abs/1506.02557>

- Bayesian inference makes assumptions about likelihood model.

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

- What happens when model is mis-specified?
  - Empirically saying, Bayesian model still output good results.
  - Theoretically, posterior distribution still converge to **maximum likelihood**:

$$p(\theta|\mathcal{D}) \rightarrow \delta(\theta_{\text{MLE}} - \theta) \quad \text{as} \quad |\mathcal{D}| \rightarrow \infty$$

$$\theta_{\text{MLE}} = \arg \max_{\theta} \log p(\mathcal{D}|\theta)$$

Maximum likelihood estimation still makes sense (takes best choice available):



- Fisher information matrix is average of  $H_{\log\text{-likelihood}}$  with respect to expectation over log-likelihood, i.e.,

$$\begin{aligned} & \mathbb{E}_{\mathcal{D} \sim p(\mathcal{D}|\theta)} \left[ \frac{\partial^2}{\partial \theta_i \partial \theta_j} \log p(\mathcal{D}|\theta) \right] \\ &= \mathbb{E}_{\mathcal{D} \sim p(\mathcal{D}|\theta)} \left[ \frac{\partial}{\partial \theta_i} \frac{\frac{\partial}{\partial \theta_j} p(\mathcal{D}|\theta)}{p(\mathcal{D}|\theta)} \right] \\ &= -\mathbb{E}_{\mathcal{D} \sim p(\mathcal{D}|\theta)} \left[ \frac{\frac{\partial}{\partial \theta_i} p(\mathcal{D}|\theta)}{p(\mathcal{D}|\theta)} \frac{\frac{\partial}{\partial \theta_j} p(\mathcal{D}|\theta)}{p(\mathcal{D}|\theta)} \right] + \mathbb{E}_{\mathcal{D} \sim p(\mathcal{D}|\theta)} \left[ \frac{\frac{\partial^2}{\partial \theta_i \partial \theta_j} p(\mathcal{D}|\theta)}{p(\mathcal{D}|\theta)} \right] \\ &= -g_i g_j \end{aligned}$$

Reduce to zero