Probabilistic Machine Learning:

12. Bayesian Neural Networks

ESC 2024 Spring Session 4주차

Contents

1

Uncertainty quantification

Introduction to BNN

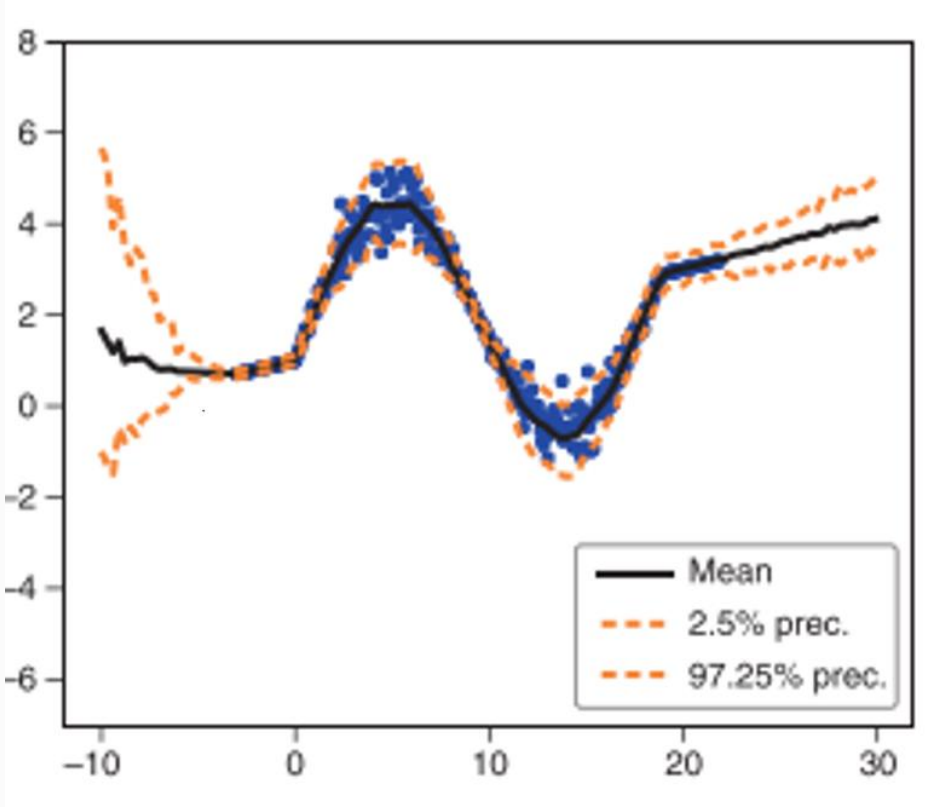
Aleatoric vs Epistemic

Aleatoric uncertainty (data inherent)

- 데이터에 내재된 노이즈로 인해 이해하지 못하는 정도
- 더 많은 데이터가 학습되더라도 줄일 수 없음
- 측정 정밀도를 높이면 줄일 수 있음

Epistemic uncertainty (model inherent)

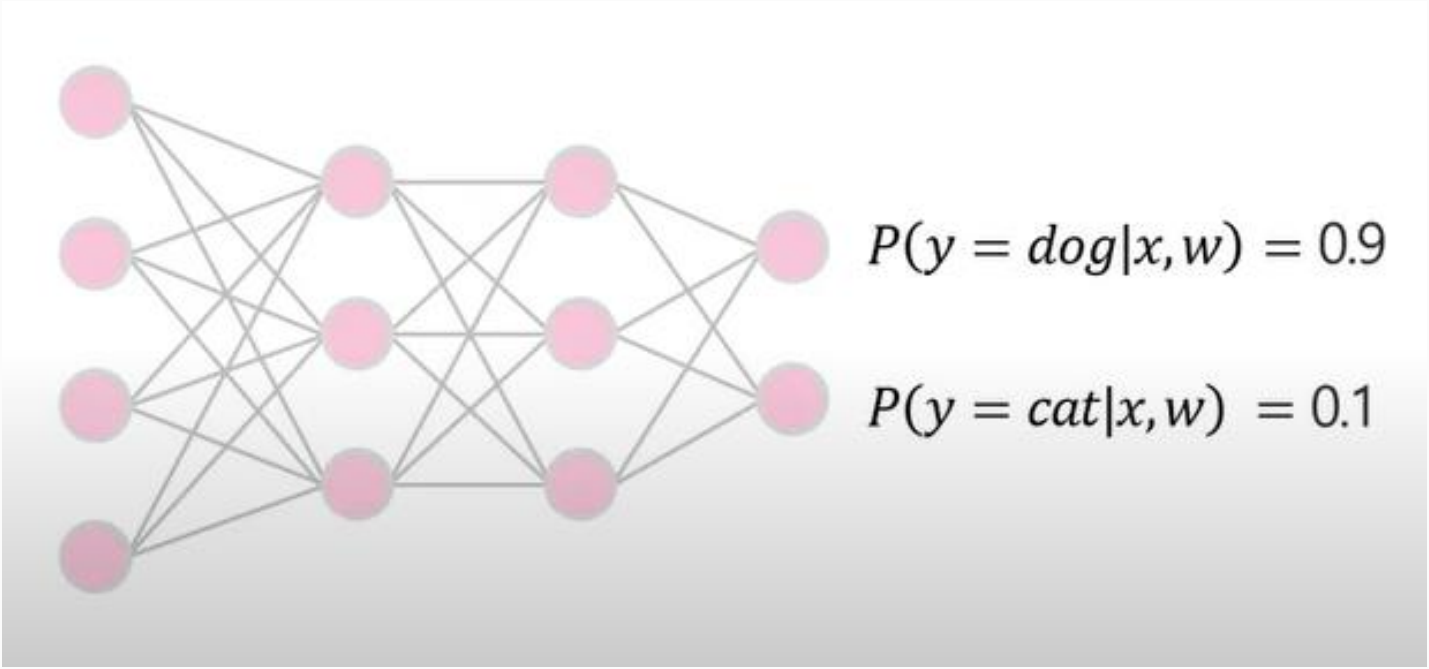
- 모델 파라미터가 데이터에 대해 얼마나 적합하게 구축되었는지 모르는 정도
- 더 많은 데이터가 학습된다면 줄일 수 있음



Non-Bayesian Neural Network

- CNN의 fully connected layer 구조라고 가정

- Softmax를 적용해 logit 값을 각 class에 대한 예측확률 값으로 변환

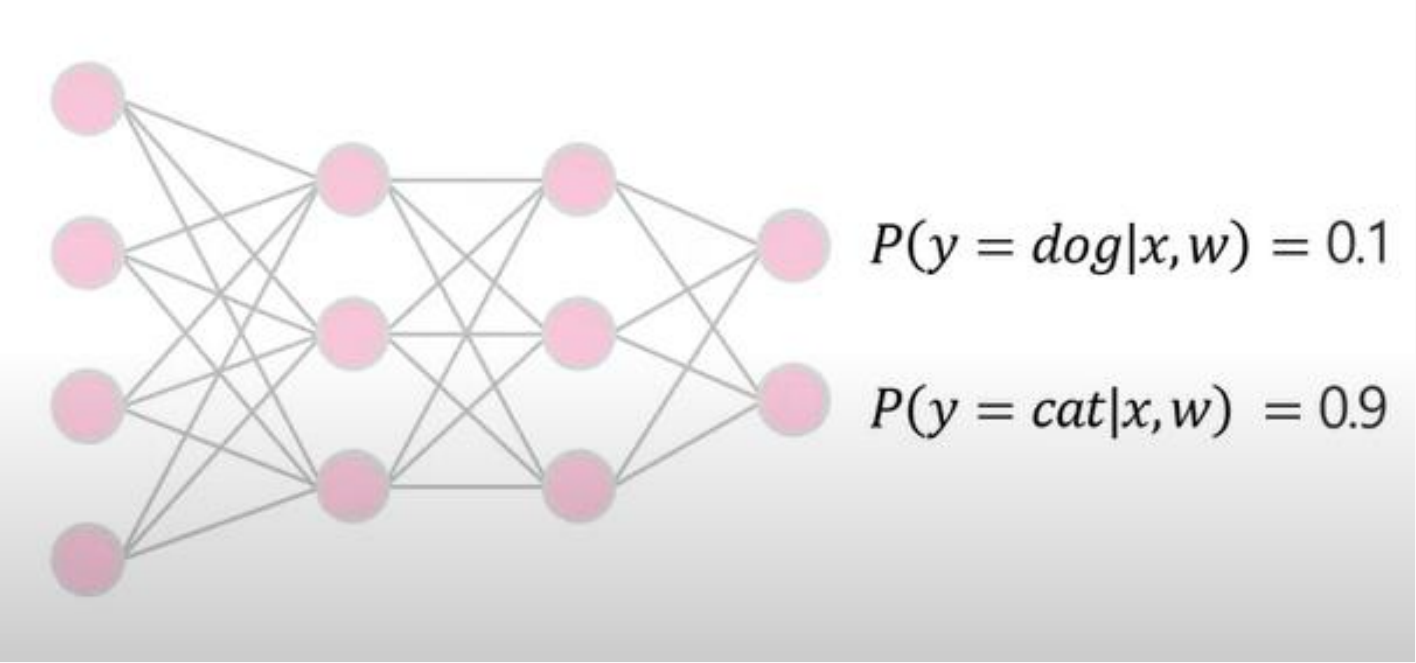


- Input data에 대한 예측 확률이 0.9로 1에 가까우므로 uncertainty가 낮다고 해석할 수 있음

Non-Bayesian Neural Network

- CNN의 fully connected layer 구조라고 가정

- Softmax를 적용해 logit 값을 각 class에 대한 예측확률 값으로 변환

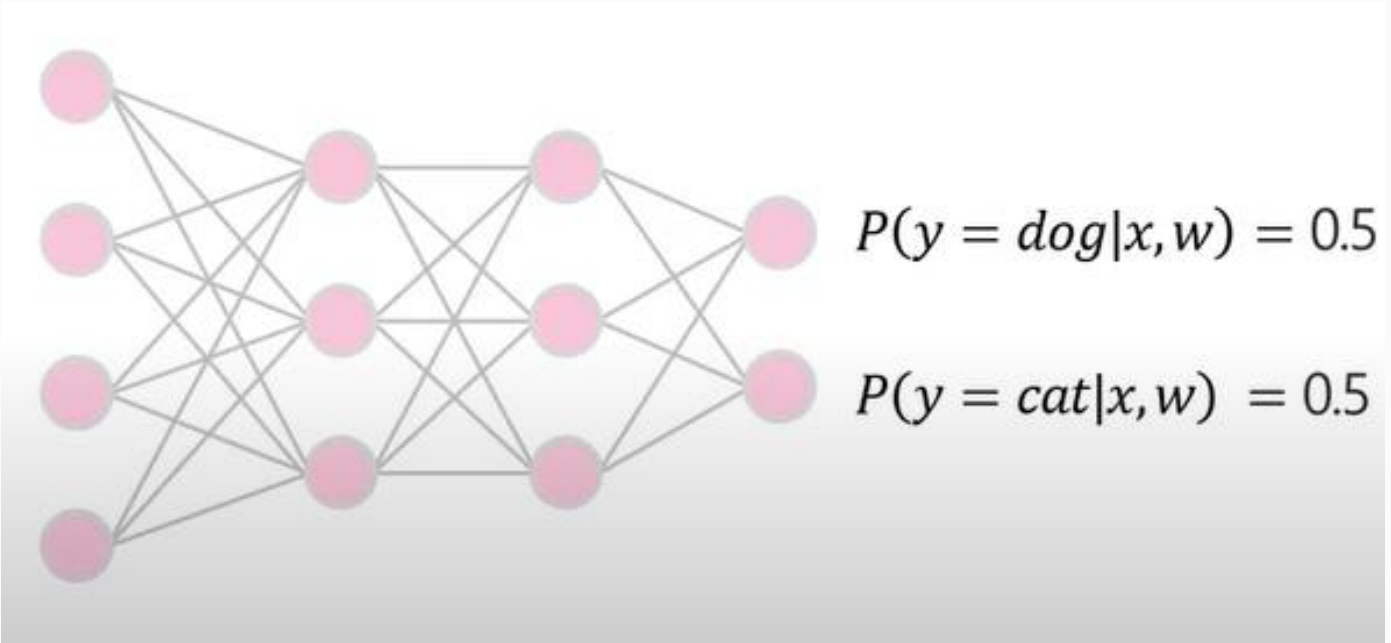


- Input data에 대한 예측 확률이 0.9로 1에 가까우므로 uncertainty가 낮다고 해석할 수 있음

Non-Bayesian Neural Network

- CNN의 fully connected layer 구조라고 가정

- Softmax를 적용해 logit 값을 각 class에 대한 예측확률 값으로 변환



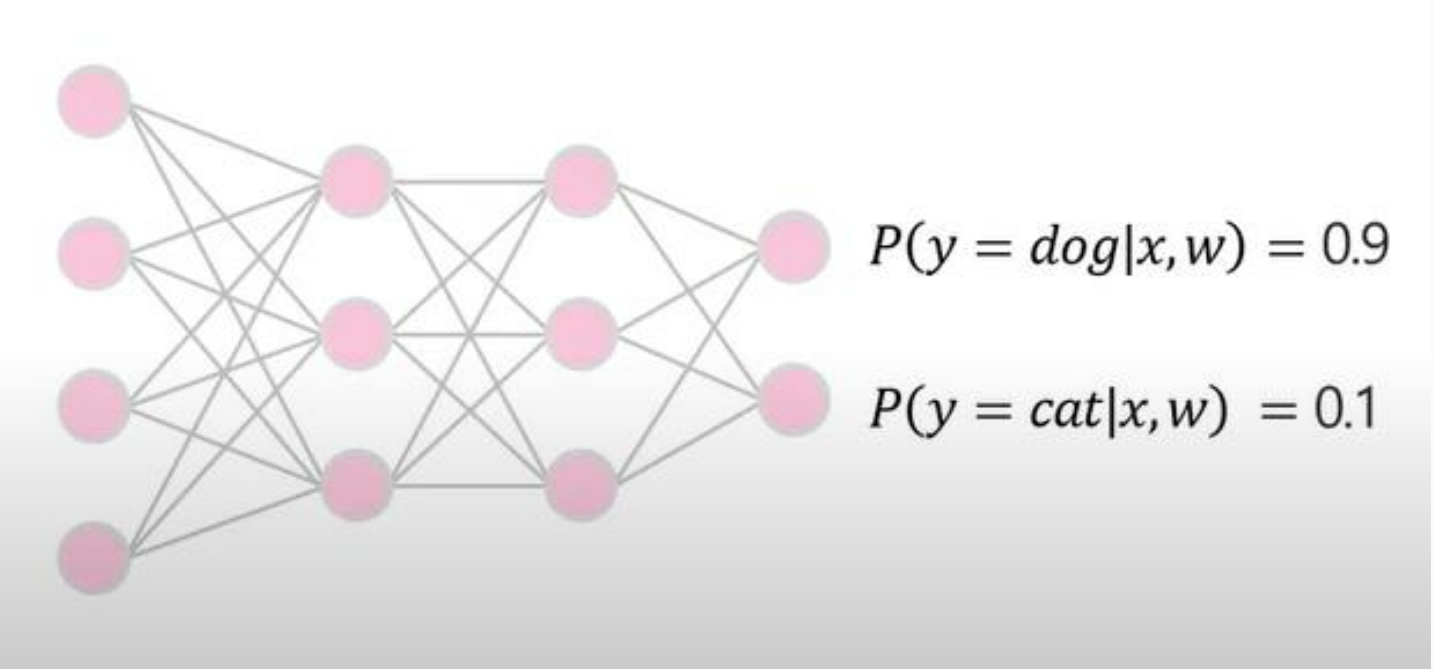
- Input data에 대한 예측 확률이 0.5이므로 uncertainty가 높다고 해석할 수 있음

Q. 그렇다면 uncertainty는 얼마만큼 높다고 할 수 있겠는가? (uncertainty quantification problem)

Non-Bayesian Neural Network

- CNN의 fully connected layer 구조라고 가정

- Softmax를 적용해 logit 값을 각 class에 대한 예측확률 값으로 변환



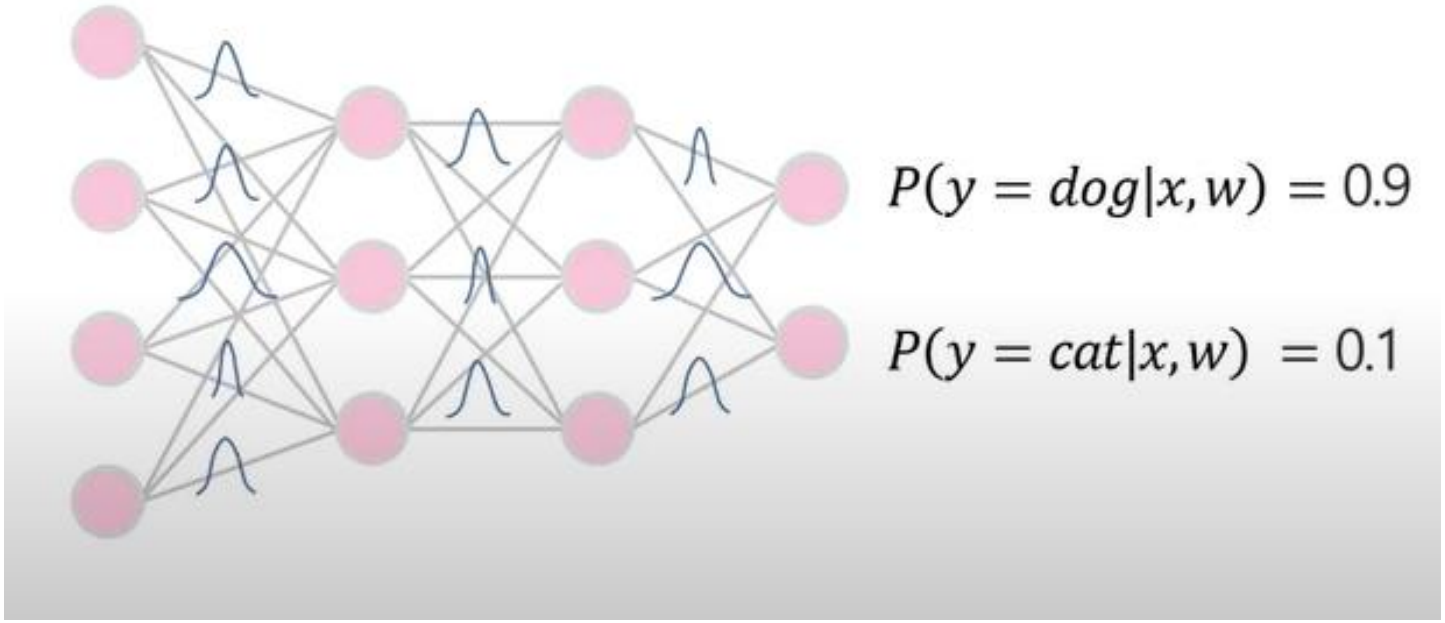
Q. Input data에 대한 예측 확률이 0.9로 1에 가까우므로 uncertainty가 낮다고 해석할 수 있겠는가?

- 새로운 data(out of distribution)에 대한 예측력이 현저히 떨어짐 (overconfidence problem in DL)

Bayesian Neural Network

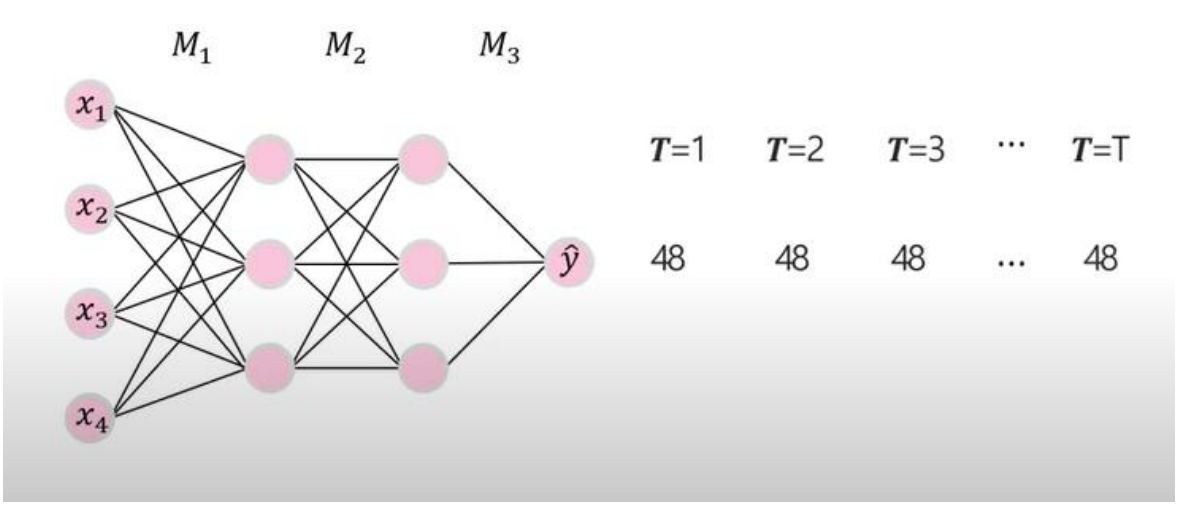
- Parameter w를 분포를 따르는 확률변수로 가정하고, 예측 값을 분포로 추정할 수 있다.

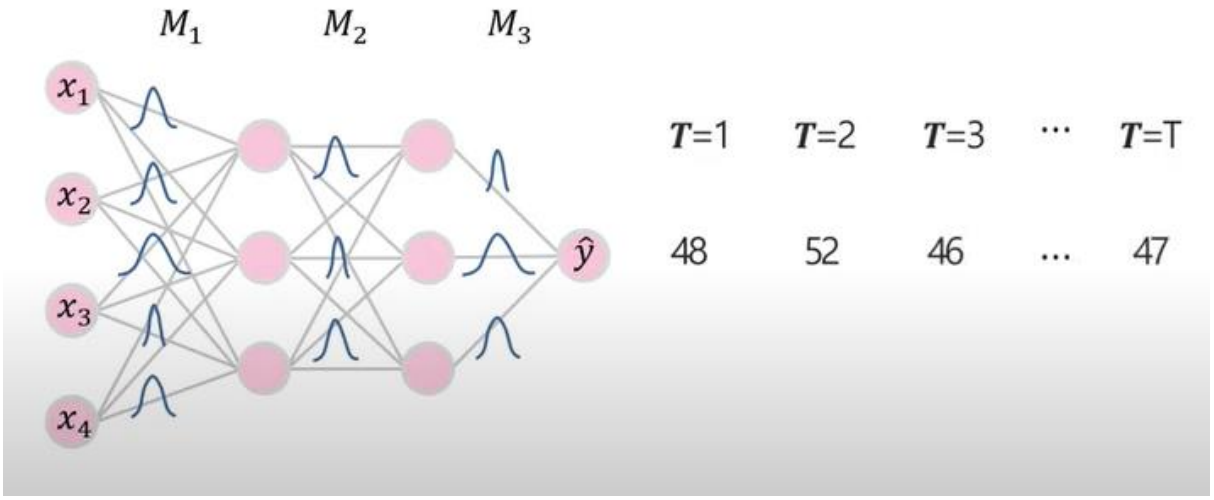
- ‘Uncertainty’를 예측 확률의 분산으로 정량화 하고자 함



Non-Bayesian Neural Network vs Bayesian Neural Network

- 같은 test data에 대해 T번 output을 산출했을 때의 그림 (왼쪽이 Standard NN, 오른쪽이 BNN)





Bayesian Neural Network

Properties

- Can detect novel situations by expressing a larger epistemic uncertainty

- Can train with less training data than traditional models

- Computational cost is intensive than traditional models because of the use of prior rather than a single parameter

Q. Then how to construct BNN wisely?

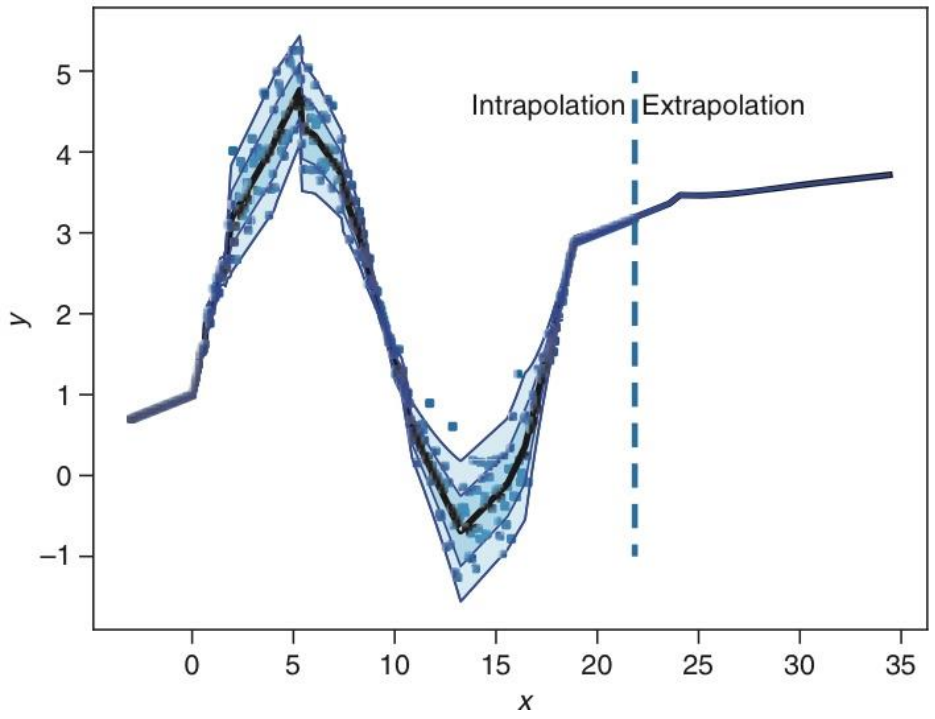
- With the help of VI or MC dropout

2

Variational inference

- Why use Bayesian Models? (With Ex.)
- VI approach
- Applying VI to BNN

Why use Bayesian Models? (with examples)



Bayesian Models can capture epistemic uncertainty
➔ Important in extrapolation
(situations not seen during training)

1. **Deterministic Models**
2. **Probabilistic Models**
3. **Bayesian Models**

Aleatoric Uncertainty
데이터 자체의 불확실성

Epistemic Uncertainty
모델의 불확실성

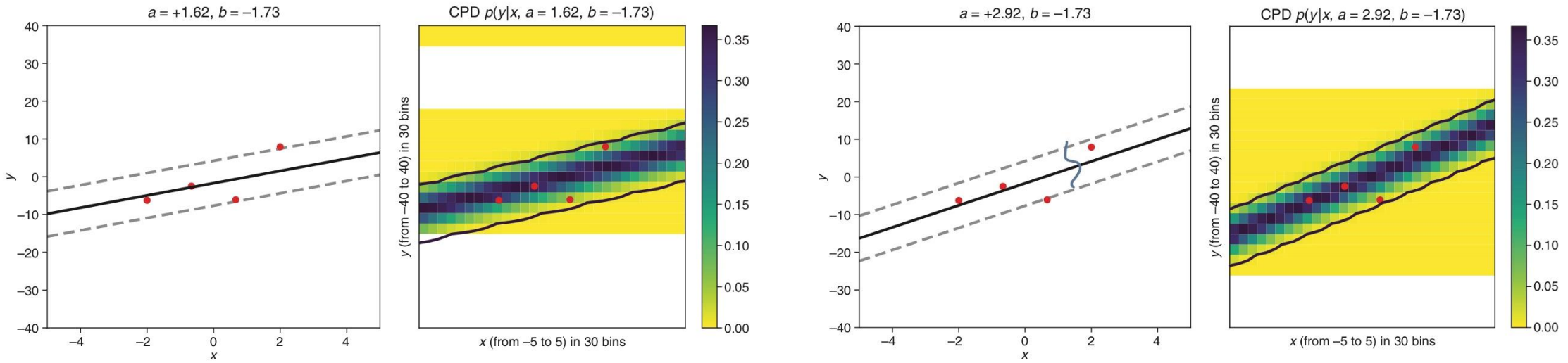
Why use Bayesian Models? (with examples)

How does Bayesian Models capture Epistemic Uncertainty?

Ex) $p(y|x,(a,b)) = N(y;\mu = a \cdot x + b, \sigma = 3)$

- Probabilistic Linear Model을 적용한 예시

파라미터가 무엇인지에 따라
데이터가 없는 부분(extrapolation)에서의 차이가 크다.



보통 Likelihood를 Maximize하는 파라미터 선택 -> 단점은 epistemic uncertainty를 고려 못함

다른 파라미터들도 가중치를 적게 해서 고려하면 epistemic uncertainty 고려 가능!

Why use Bayesian Models? (with examples)

다른 파라미터들도 가중치를 적게 해서 고려하면 epistemic uncertainty 고려 가능!

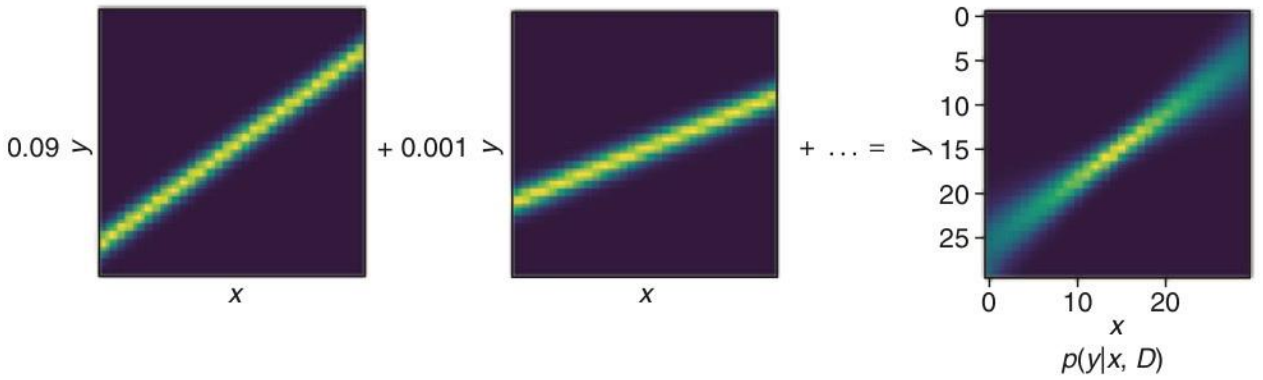
➔ Likelihood에 의한 가중평균으로 생각

(a_1, b_1)에 대한 Likelihood: $p(D|(a_1, b_1))$

(a_1, b_1)에 대한 Likelihood weight: $p_n(D|(a_1, b_1)) = \frac{p(D|a_1, b_1)}{\sum_a \sum_b p(D|(a, b))}$ where $\sum_a \sum_b p_n(D|(a, b)) = 1$

$$p(y|x, D) = \sum_a \sum_b p(y|x, (a, b)) \cdot p_n(D|(a, b))$$

특정 분포 분포에 대한 가중치



Epistemic uncertainty를 잘 표현한다.

Why use Bayesian Models? (with examples)

$$p(y|x,D) = \sum_a \sum_b p(y|x,(a,b)) \cdot p_n(D|(a,b))$$

↓ For continuous weight

$$p(y|x,D) = \int_w p(y|x,w) \cdot p_n(D|w)\, dw$$

Weight이 따르는 distribution

↓ Prior까지 고려한다면

$$p(y|x,D) = \int_w p(y|x,w) \cdot p(w|D) dw$$

posterior

결론: A Bayesian Model is a weighted ensemble model that captures epistemic uncertainty!

VI approach

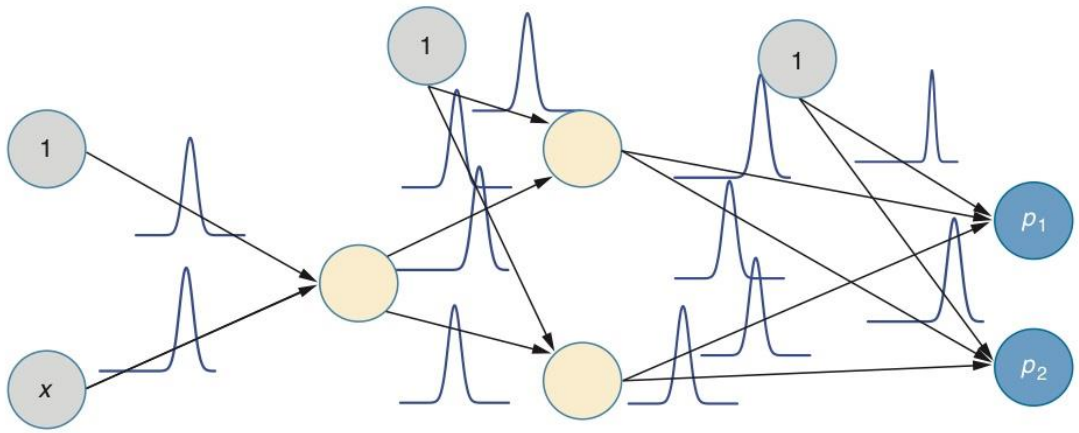
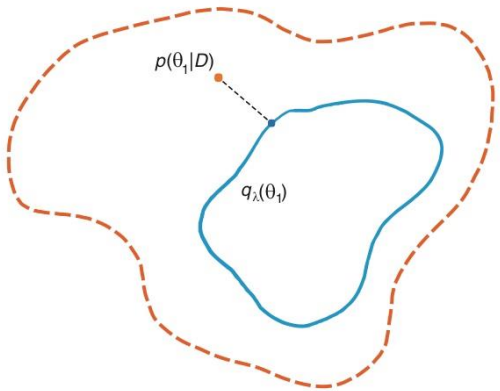


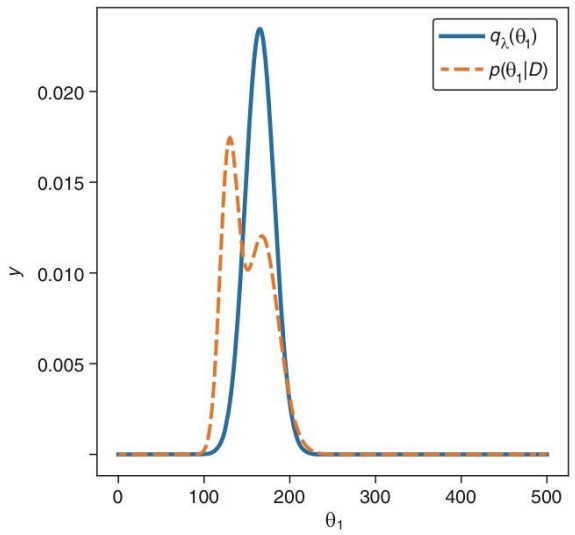
Figure 8.2 A Bayesian network with two hidden layers. Instead of fixed weights, the weights now follow a distribution.

BNN: each weight is replaced by a distribution -> but complicated & not independent

Idea of VI: approximate posterior distributions with variational distribution

Variational Inference





$p(\theta|D)$ 을 직접 구하기 어려움 - high dimension

Variational distribution

$$q_\lambda(\theta) \rightarrow p(\theta|D)$$

$$(if\ Gaussian)\ \ \lambda = (\mu, \sigma)$$

Variational parameters

VI의 목표: Find λ so $q_\lambda(\theta)$ gets close to $p(\theta|D)$

= Find λ that minimizes KL divergence

Variational Inference

$$\lambda^* = argmin\{KL[q_\lambda(\theta)||p(\theta|D)]\} = argmin\{KL[q_\lambda(\theta)||p(\theta)] - E_{\{\theta \sim q_\lambda\}}[log(p(D|\theta))]\}$$

KL divergence: q & posterior　　KL divergence: q & prior　　Negative Log Likelihood

<proof>

$$KL[q_\lambda\theta||p(\theta|D)] = \int q_\lambda(\theta)\, log\, \frac{q_\lambda(\theta)}{p(\theta|D)}\, d\theta$$

$$= \int q_\lambda(\theta)\, log\, \frac{q_\lambda(\theta)}{p(\theta,D)/p(D)}\, d\theta$$

$$= \int q_\lambda(\theta)\, log\, p(D)\, d\theta - \int q_\lambda(\theta)\, log\, \frac{p(\theta,D)}{q_\lambda(\theta)}\, d\theta$$

$$= log\, p(D) \cdot \int q_\lambda(\theta)\, d\theta - \int q_\lambda(\theta)\, log\, \frac{p(\theta,D)}{q_\lambda(\theta)}\, d\theta$$

$$= log\, p(D) - \int q_\lambda(\theta)\, log\, \frac{p(\theta,D)}{q_\lambda(\theta)}\, d\theta$$

$$\lambda^* = argmin\{-\int q_\lambda(\theta)\, log\, \frac{p(\theta,D)}{q_\lambda(\theta)}\, d\theta\}$$

$$= argmin\{-\int q_\lambda(\theta)\, log\, \frac{p(D|\theta)\cdot p(\theta)}{q_\lambda(\theta)}\, d\theta\}$$

$$= argmin\{\int q_\lambda(\theta)\, log\, \frac{q_\lambda(\theta)}{p(\theta)}\, d\theta - \int q_\lambda(\theta)\, log\, p(D|\theta)\, d\theta\}$$

$$= argmin\{KL[q_\lambda(\theta)||p(\theta)] - E_{\{\theta \sim q_\lambda\}}[log(p(D|\theta))]\}$$

→ Posterior를 몰라도 KL Divergence를 구할 수 있다.

Variational Inference in BNN

$$\lambda^* = argmin\{KL[q_\lambda(\theta)||p(\theta|D)]\} = \underbrace{argmin\{KL[q_\lambda(\theta)||p(\theta)]}_{①} \underbrace{- E_{\{\theta \sim q_\lambda\}}[log(p(D|\theta))]\}}_{②}$$

① Choose good prior

- In BNN, prior is usually chosen to be around zero → $q_\lambda\theta$ (variational dist) is good if centered around small values

② Averaged NLL

- 정확히 말하자면 Expectation of Negative Log Likelihood
- q_λ에서 sampling한 θ들로 Negative Log Likelihood의 평균을 구한 것이라 생각
  - 이를 minimize하는 것은 원래 NN의 목표와 부합

목표 : Minimize NLL averaged according to the probability of θ,
with the restriction that the variational distribution of θ is not too far away from the prior.

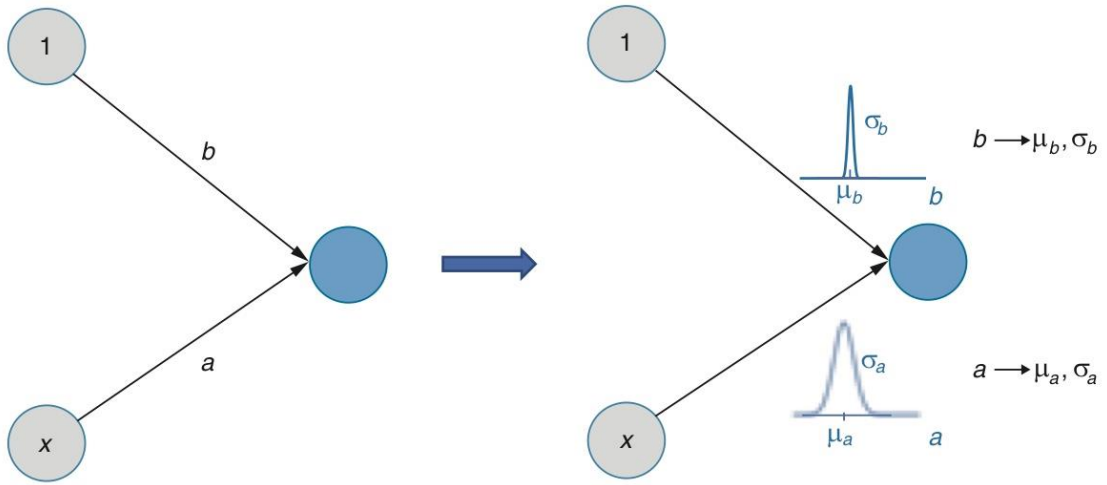
Applying VI

Define Problem

Ex) $p(y|x,(a,b)) = N(y;\mu = a \cdot x + b, \sigma = 3)$

① Define priors for (a,b) : $N(0,1)$

② Define Variational Distribution : $a \sim N(\mu_a, \sigma_a), b \sim N(\mu_b, \sigma_b)$



Optimize $\lambda = (\mu_a, \sigma_a, \mu_b, \sigma_b)$

⇔ Optimize $w = (\mu_a, w_1, \mu_b, w_3)$ where $\sigma_a = f(w_1), \sigma_b = f(w_3)$

Loss: $\{KL[q_\lambda(\theta)||p(\theta)] - E_{\{\theta \sim q_\lambda\}}[log(p(D|\theta))]\}$

〈CODE〉

>> sigma_a = tf.math.softplus(w[1])
>> sigma_a = tf.math.softplus(w[3])

Applying VI

Loss 살펴보기

Ex) $p(y|x,(a,b)) = N(y;\mu = a \cdot x + b, \sigma = 3)$

Optimize $w = (\mu_a, w_1, \mu_b, w_3)$ to minimize loss: $\{KL[q_\lambda(\theta)||p(\theta)] - E_{\{\theta \sim q_\lambda\}}[log(p(D|\theta))]\}$

$loss_{VI} = \{KL[q_\lambda(\theta)||p(\theta)] - E_{\{\theta \sim q_\lambda\}}[log(p(D|\theta))]\} = loss_{KL} + loss_{NLL}$

In this example,

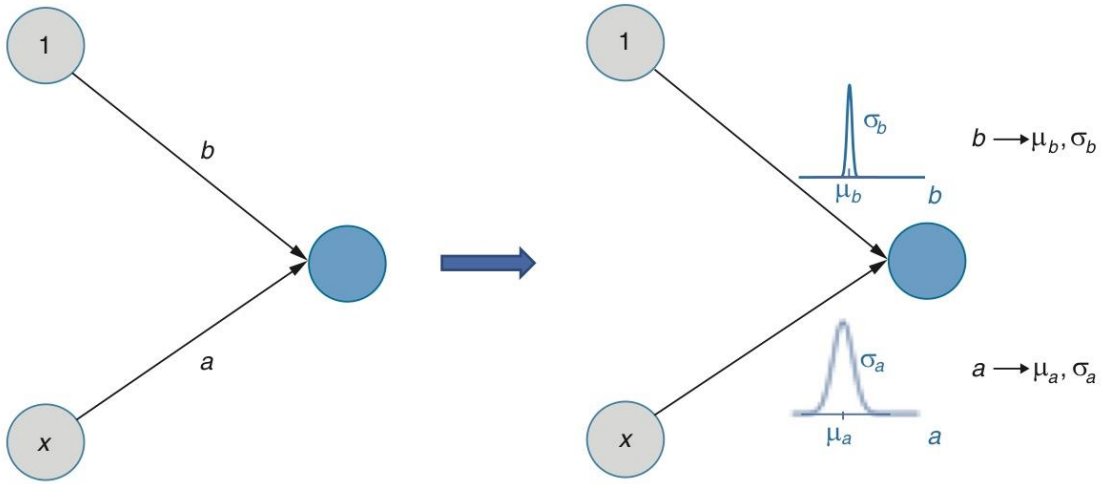
1) $loss_{KL} = KL[q_\lambda(w)||p(w)] = KL[N(\mu,\sigma)||N(0,1)] = -\frac{1}{2}(1 + log(\sigma^2) - \mu^2 - \sigma^2)$

2) $loss_{NLL}$ has no closed form → approximate again.

- Sample θ from $\theta \sim q_\lambda$ and calculate empricial mean of $-log(p(D|\theta))$
- How many θ is needed? One is enough!
  - Gradient may be noisy but it still finds its way to the minimum
  - θ 여러 개 sampling하면 direction would be more accurate, but computation↑

Applying VI

Forward Pass



1. Fixed vector $w = (\mu_a, w_1, \mu_b, w_3)$
2. $\lambda = (\mu_a, \sigma_a = sp(w_1), \mu_b, \sigma_b = sp(w_3))$
3. With fixed variational dist, Calculate Loss

$$\text{Loss}_{\text{VI}} = loss_{KL} + loss_{NLL}$$

$$\text{Loss}_{\text{KL}} = -\frac{1}{2}(1 + log(\sigma^2) - \mu^2 - \sigma^2)$$

$$\text{Loss}_{\text{NLL}} \approx -\frac{\sum_{i=1}^{n} log\ (N(y_i; a \cdot x + b, \sigma))}{n}$$ with one sampled (a,b)

〈CODE〉

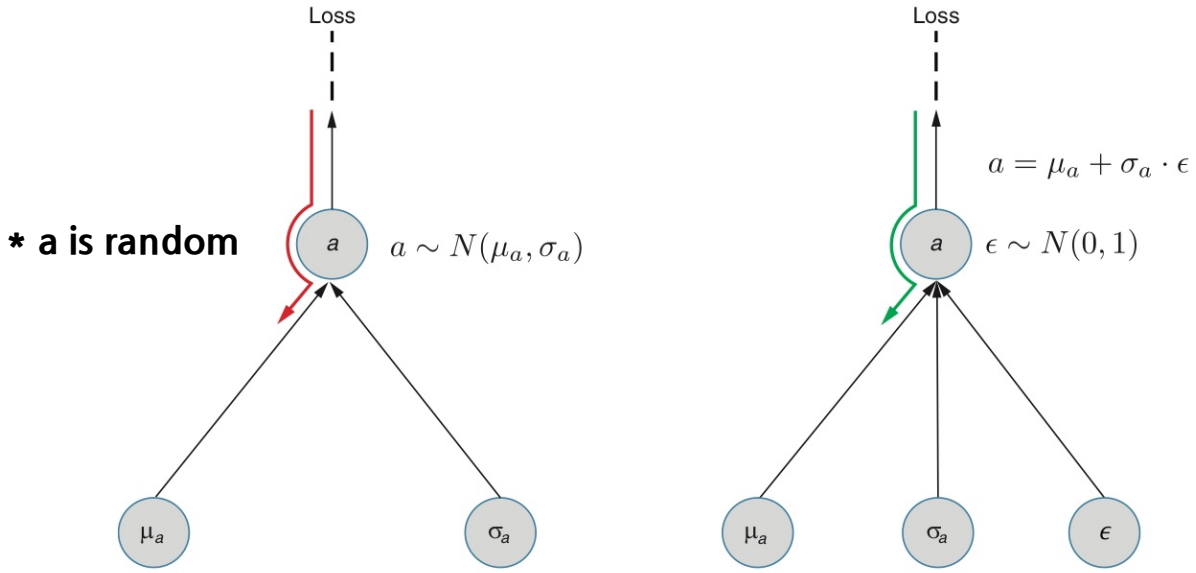
```
>> y_prob = tfd.Normal(loc = x * a + b, scale = sigma)
>> loss_nll = -tf.reduce_sum(y_prob.log_prob(ytensor)
```

Applying VI

Gradient Descent

Problem: How do you calculate the gradient of a sampling process?

→ **Reparameterization**



* a is random

Sample $a \sim N(\mu_a, \sigma_a)$

↕

Reparameterize $a = \mu_a + \sigma_a \cdot \epsilon,\ \epsilon \sim N(0,1)$

You still have the sample of N(0,1) but can ignore ϵ

(ϵ에 대해 미분할 필요가 없기 때문)

→ Able to backpropagate to get $\frac{\partial}{\partial \mu_a}, \frac{\partial}{\partial \sigma_a}$

Applying VI

Full Code

Listing 8.1 Using VI for the simple linear regression example (full code)

```
w_0=(1.,1.,1.,1.)
log = tf.math.log
w = tf.Variable(w_0)
e = tfd.Normal(loc=0., scale=1.)
ytensor = y.reshape([len(y),1])
for i in range(epochs):
    with tf.GradientTape() as tape:

    mu_a = w[0]
    sig_a = tf.math.softplus(w[1])

    mu_b = w[2]
    sig_b = tf.math.softplus(w[3])

    l_kl = -0.5*(1.0 +
        log(sig_a**2) - sig_a**2 - mu_a**2 +
        1.0 + log(sig_b**2) - sig_b**2 - mu_b**2)

    a =  mu_a + sig_a * e.sample()
    b =  mu_b + sig_b * e.sample()

    y_prob = tfd.Normal(loc=x*a+b, scale=sigma)
    l_nll = \
        -tf.reduce_sum(y_prob.log_prob(ytensor))

    loss = l_nll + l_kl
grads = tape.gradient(loss, w)
logger.log(i, i, w, grads, loss, loss_kl, loss_nll)
w = tf.Variable(w - lr*grads)
```

The initial condition of the vector w

The noise term, needed for the variational trick

Controls the center of parameter a

Controls the spread of parameter a

Controls the spread of parameter b

Controls the center of parameter b

KL divergence with Gaussian priors

Samples b ~ N(mu_b, sigma_b)

Samples a ~ N(mu_a, sigma_a) with the reparameterization trick

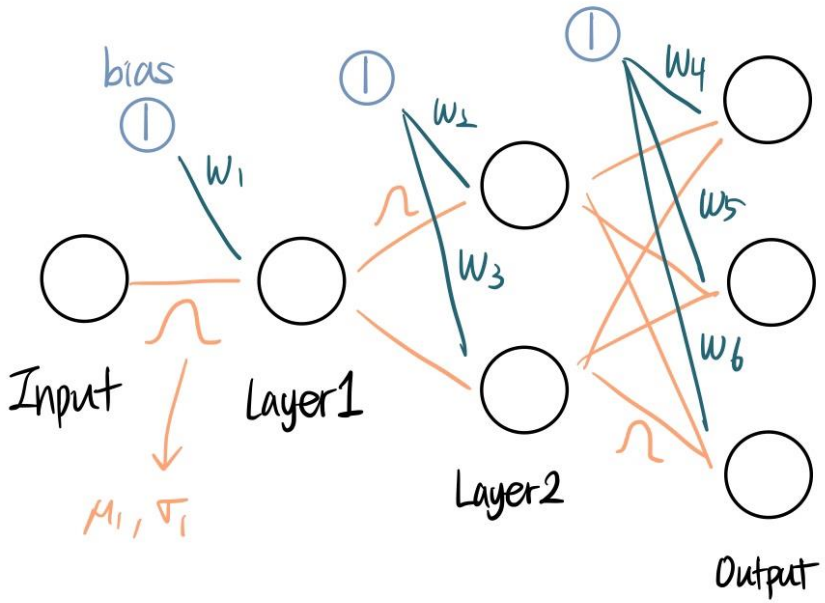
Calculates the NLL

Gradient descent

VI BNN with Multiple Layers

Three Layer VI BNN → construct layer with tfp.layers.DenseReparameterization(#num of node)

```
model = tf.keras.Sequential([
  tfp.layers.DenseReparameterization(1, input_shape=(None,1)),
  tfp.layers.DenseReparameterization(2),
  tfp.layers.DenseReparameterization(3)
])
```



Default: Bias has no variational dist, only one weight

How many parameters?

1. **Input → Layer1: $2 \cdot 1 + 1 = 3$**
2. **Layer 1 → Layer2: $2 \cdot 2 + 2 = 6$**
3. **Layer 2 → Layer3(Output): $2 \cdot 6 + 3 = 15$**

<CODE> - Add bias distribution

>> bias_prior_fn = normal
>> bias_posterior_fn = normal

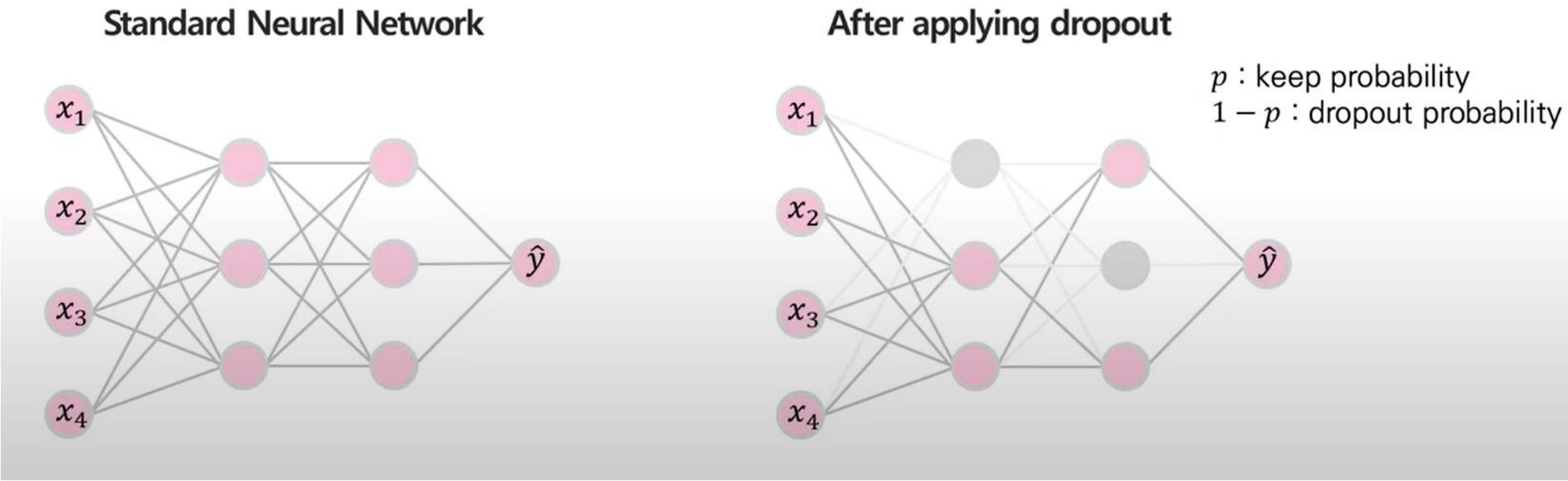
3

Monte Carlo Dropout

Algorithm to implement Bayesian approach

Dropout

- Dropout was introduced as a way to avoid overfitting from fully connected NN
- In each update run, Randomly pick a node and disconnect them (set the output of node as zero)
- Dropout probability $1 - p$ is user determined
- Only conducted in training phase

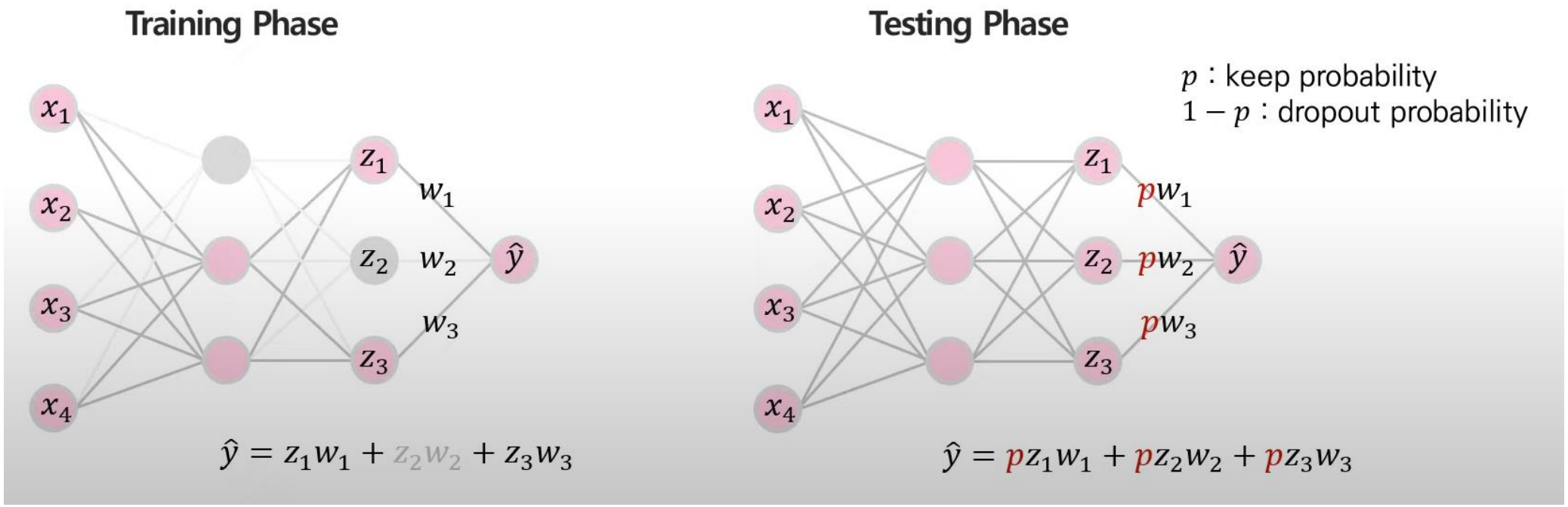


Dropout

- At testing phase, parameters are deterministically modeled

- Downweigh the learned weight value via formula

$$w^* = pw$$



Why dropout prevents overfitting?

- encourages the network to learn multiple independent representations of the same data, reducing the risk of overfitting to specific features of the training set

- Ensemble effect of several thinned networks, since at each training iteration, different neurons are dropped out

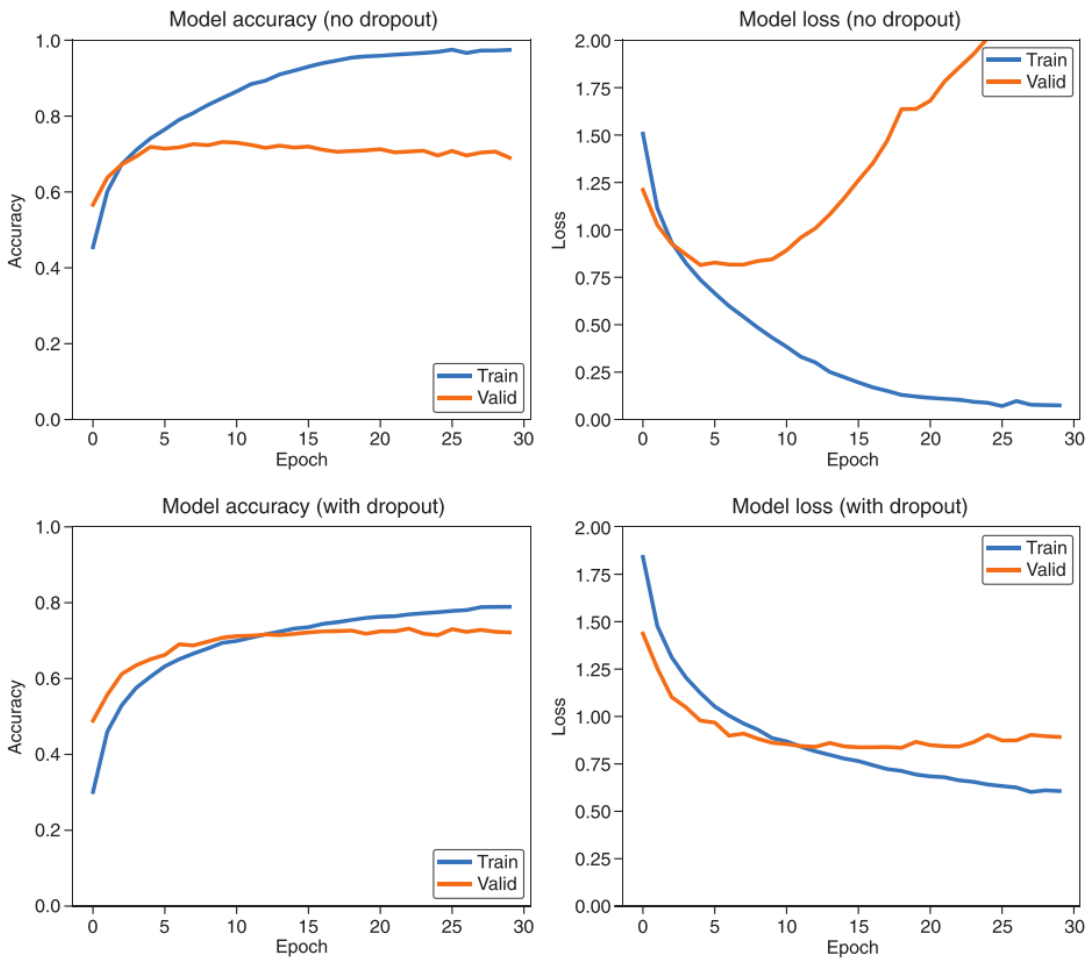


Figure 8.10 CIFAR-10 results achieved with and without using dropout during training. During testing, the weights are fixed (no MC dropout). When using dropout, the accuracy on the validation data is higher (left panel), and the distance between validation and trained loss is much smaller (right panel). This indicates that dropout prevents overfitting.

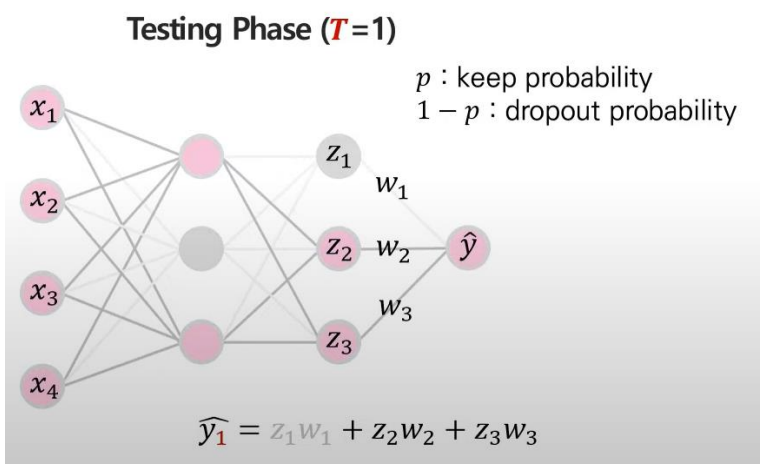
Motivations of MC dropout

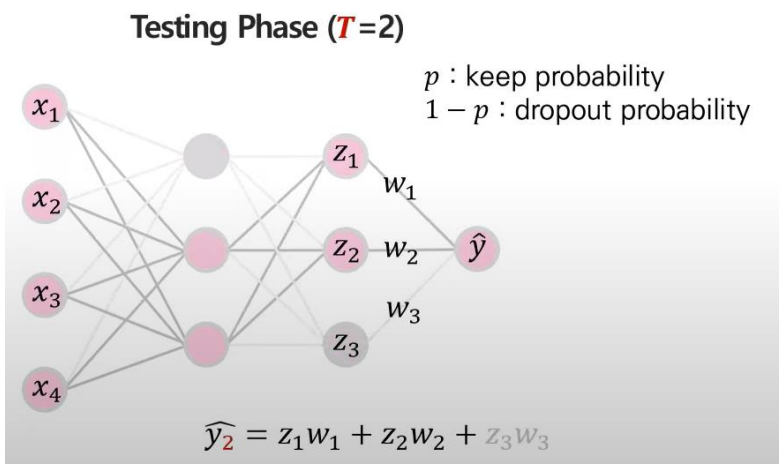
- BNN with VI : should estimate parameter twice then traditional neural network

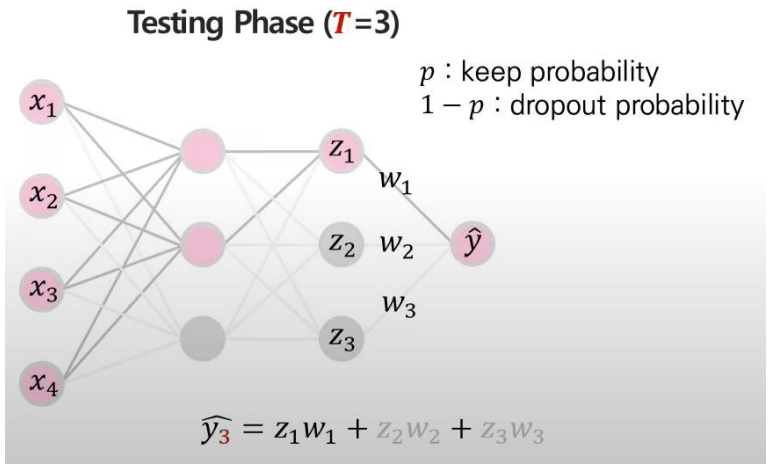
- it would be even nicer if the number of parameters wouldn't double when going from a non-Bayesian NN to its Bayesian variant

MC dropout

- Not only conducted in training phase but also in testing phase
- Each sample for the same test data yields different outcomes







MC dropout with Bayesian approach

- Distribution of each weight follows a Bernoulli distribution

- Since dropout probability $(1-p)$ is fixed

- The only parameter in this distribution is W

- Compared to distribution of each weight in VI, one parameter is required in MC dropout method

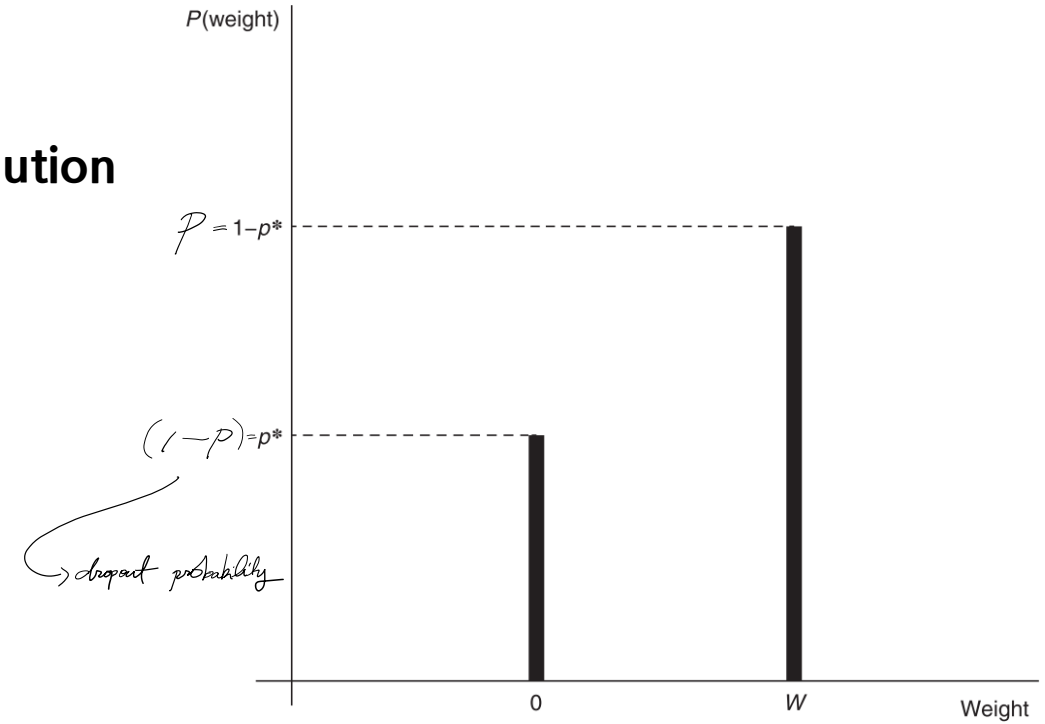


Figure 8.11 The (simplified) weight distribution with MC dropout. The dropout probability p^* is fixed when defining the NN. The only parameter in this distribution is the value of w.

MC dropout with Bayesian approach

- Bayesian predictive distribution:

$$P(y|x_{test}, D) = \sum_i P(y|x_{test}, w_i) \cdot p(w_i|D)$$

- With sampling T times for given data x_{test}, approximated predictive distribution:

$$P(y|x_{test}, D) \cong \frac{1}{T} \sum_{t=1}^{T} P(y|x_{test}, w_t)$$

- The resulting predictive distribution captures the epistemic and the aleatoric uncertainties

- Yarin Gal, was able to show that the MC dropout method was similar to VI, allowing us to approximate a BNN

4

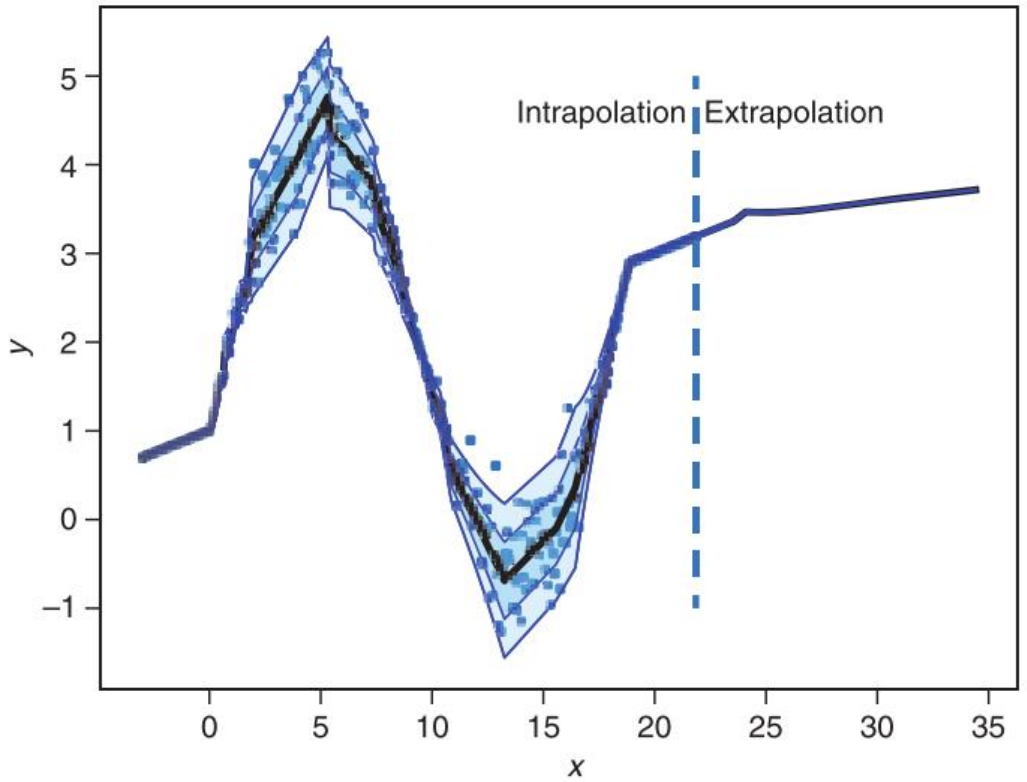
Case Studies

Regression & Classification

BNN for regression

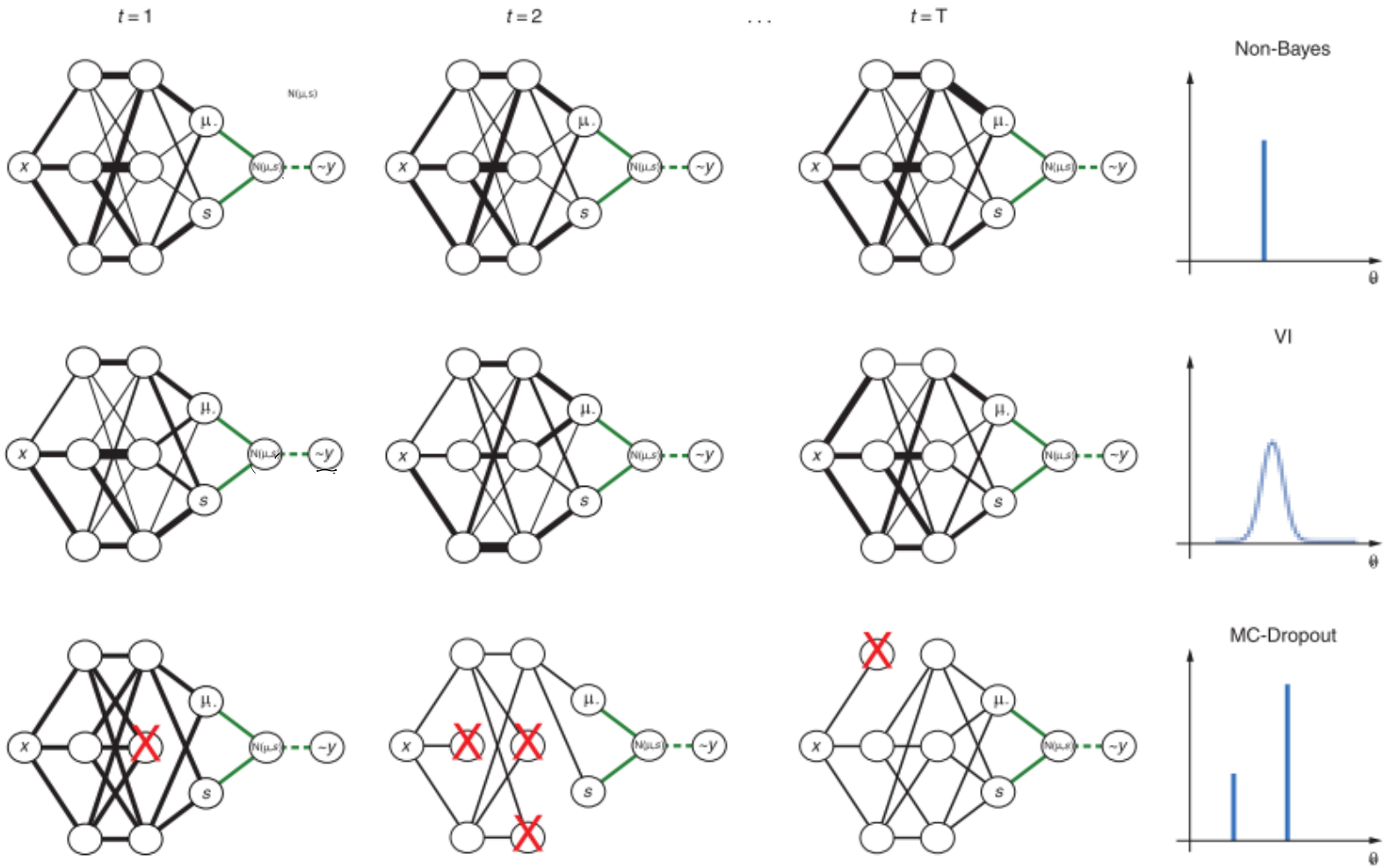
Training data range: -3 to 22 but Validation data range: -10 to 40

Ideal expectation:
Since validation data range doesn't only cover the range of training data, model should estimate the parameters with high uncertainty in the extrapolation region rather than intrapolation region. (high epistemic uncertainty in extrapolation region)



BNN for regression

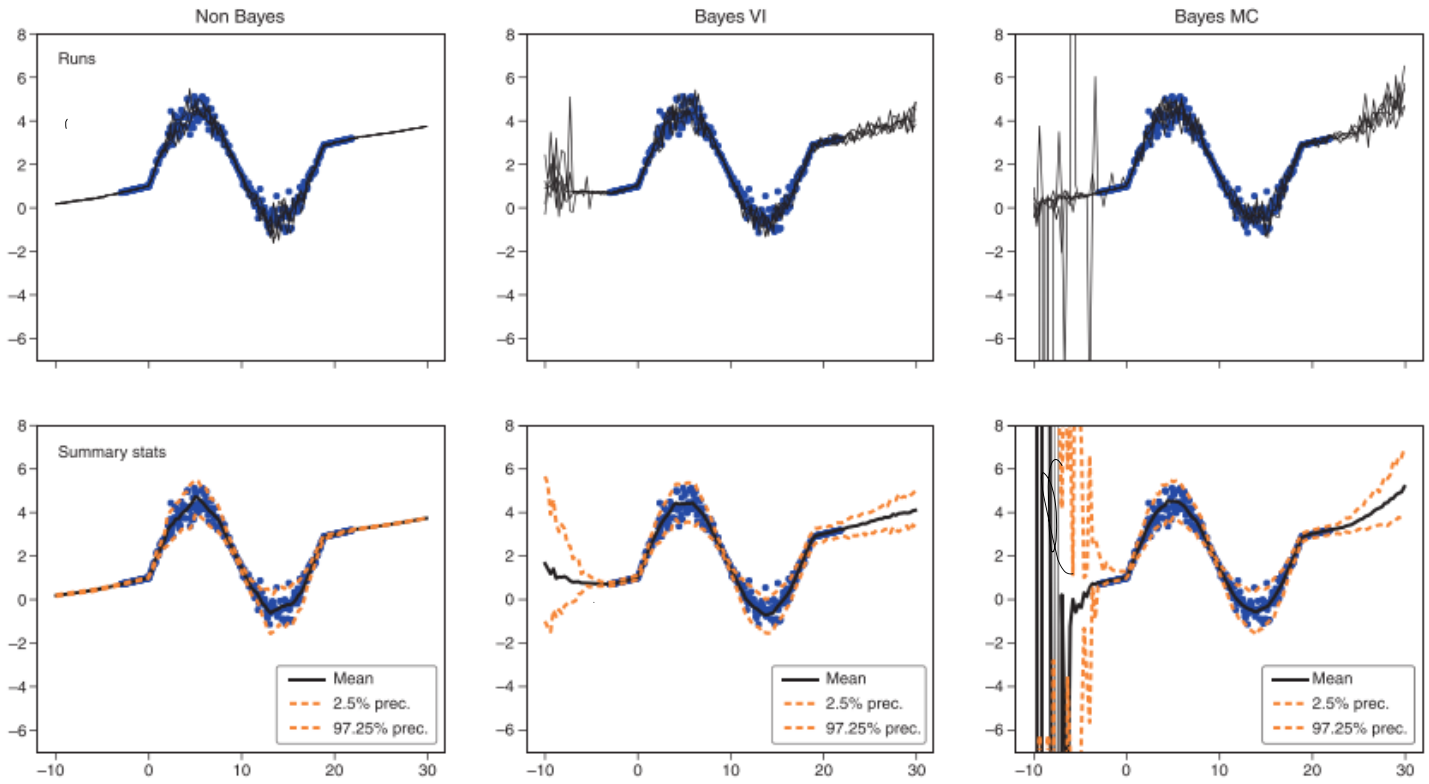
Then how to determine the CPD; predictive distribution of $p(y|x_i, D)$ for a given input x_i?



BNN for regression

The uncertainty captured by the spread of the CPD is large in regions where the data spread is also large.
-> able to quantify aleatoric uncertainty in all models

When we go into the extrapolation region, the non-Bayesian approach fails. It assumes 95% of the data in an unrealistically narrow region.
-> Overconfidence problem in non-Bayesian network



BNN for classification

CIFAR-10 dataset:

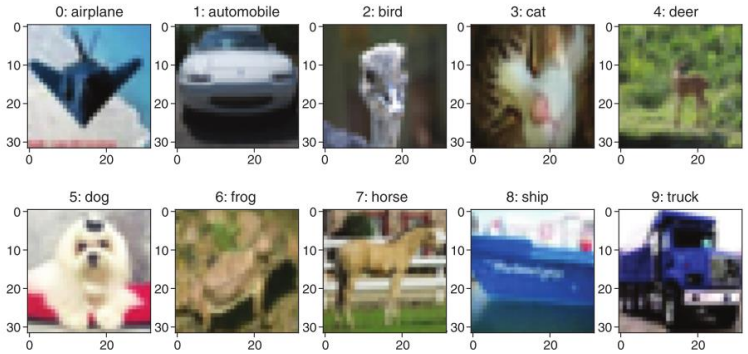


Figure 8.15 Example images for the ten classes in the CIFAR-10 data set (same as figure 8.9)

Known class:
all networks correctly classify the image of an airplane.
Especially In the two BNNs, little variation in the plots indicates that the classification is quite certain.

Unknown class:
all predictions are wrong. But BNNs can better express their uncertainty by sampling the outcome T times for each classes.

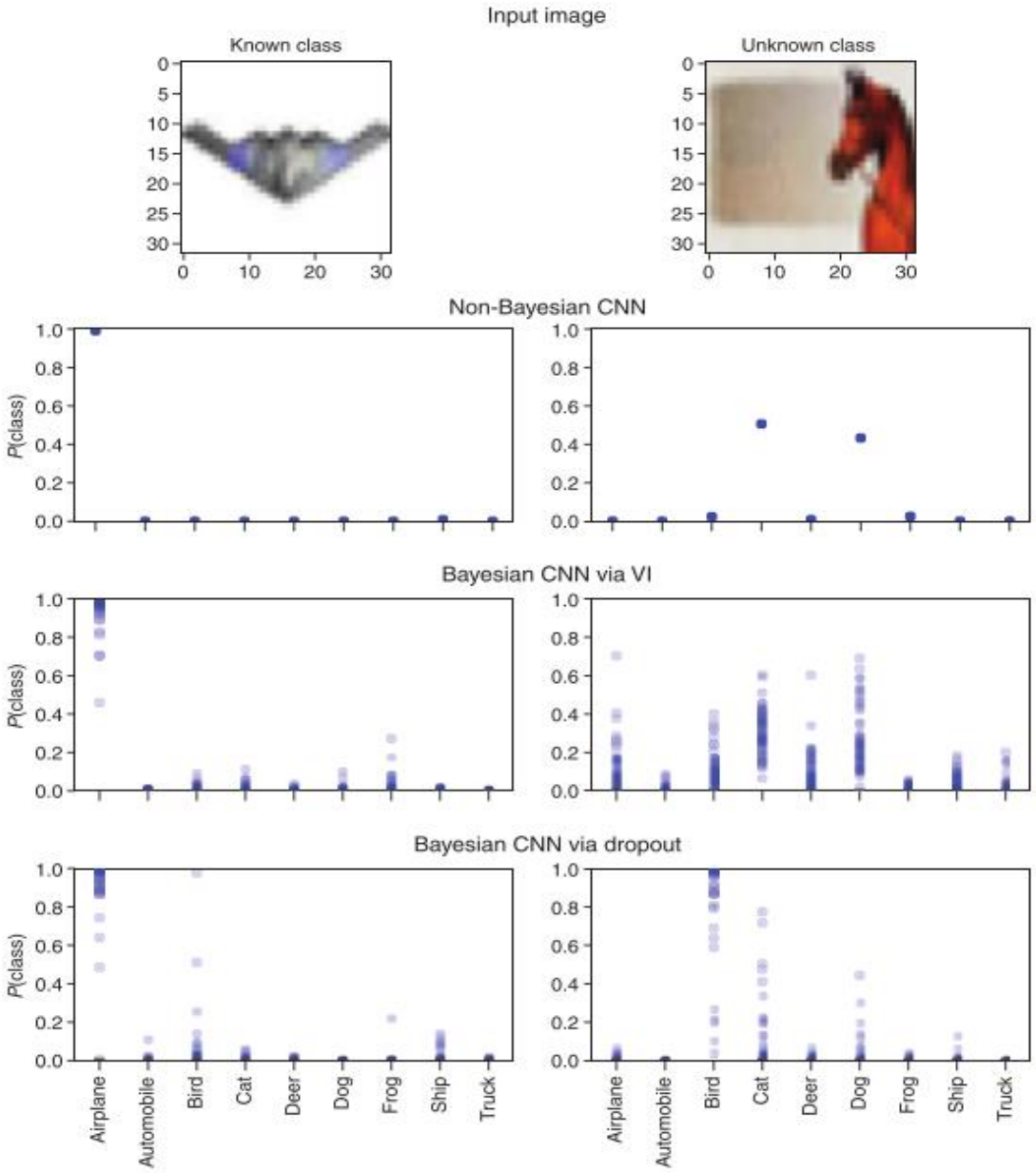


Figure 8.17 Upper panel: images presented to train CNNs included an image from the known class *airplane* (left) and an image from the unknown class *horse* (right). Second row of plots: corresponding predictive distributions resulting from non-Bayesian NN. Third row of plots: corresponding predictive distributions resulting from BNN via VI. Fourth row of plots: corresponding predictive distributions resulting from BNN via MC dropout.

감사합니다