

---

# EM Algorithm to Train Neural Networks

ESC 2024 Spring Final Project 3조 – 최운형, 김상민, 김효은, 심재윤, 이현우



# Contents

---

**1. EM Algorithm**

**2. Training Multilayer Perceptron Network**

**3. Training MLP : with Python**

**4. Discussion & Conclusion**

# 1

## EM Algorithm

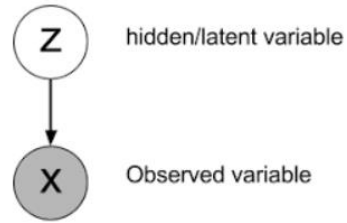
---

# The General EM Algorithm

- EM for latent

- X의 ML은 다음과 같다.

$$\max_{\theta} p(\mathbf{X}|\theta) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\theta).$$



- X의 marginal을 계산하기 어렵기 때문에

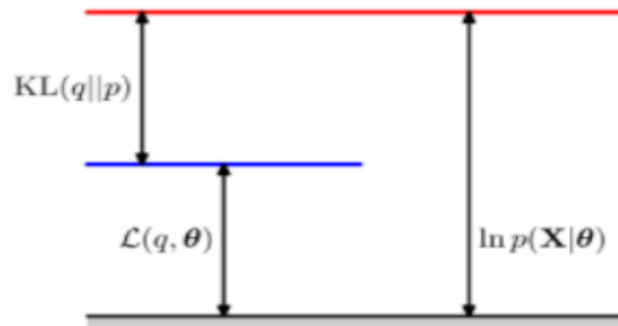
Joint  $p(\mathbf{X}, \mathbf{Z}|\theta)$  를 사용한다.

- Latent Z의 marginal을  $q(\mathbf{Z})$ 라 하면 log-likelihood를 다음과 같이 쓸 수 있다.

$$\ln p(\mathbf{X}|\theta) = L(q, \theta) + KL(q||p)$$

- KL divergence가 반드시 0보다 크거나 같기 때문에  $L(q, \theta)$  이 곧 log-likelihood의 lower bound가 된다.

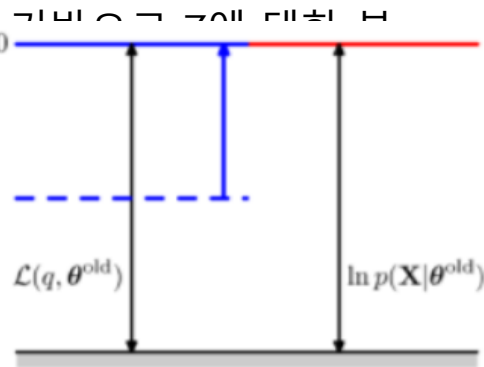
- 즉 EM for latent의 의미는 lower bound가 maximum이 되도록 하는  $\theta$ 와  $q(\mathbf{Z})$ 의 값을 찾고, 그에 해당하는 log-likelihood의 값을 찾는 것이다.
- 구체적으로는  $\theta$ 와  $q(\mathbf{Z})$ 를 jointly optimize하는 문제가 어려운 문제라면 둘 중 한 variable을 고정해두고 나머지를 update한 다음, 나머지 variable을 같은 방식으로 update하는 alternating method이다.



# The General EM Algorithm

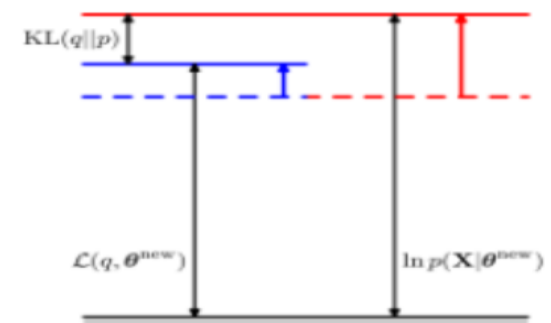
## • E-step

- $\theta_{old}$  값을 고정해두고,  $L(q, \theta)$ 의 값을 최대로 만드는  $q(Z)$ 의 값을 찾는 과정
- KL divergence는  $q(Z) = p(Z|X, \theta_{old})$  인 상황에서 0이 되기 때문에,  $q(Z)$ 에 posterior distribution  $p(Z|X, \theta_{old})$ 을 대입하는 것으로 해결할 수 있다.
- 따라서 E-step은 언제나 KL-divergence를 0으로 만들고, lower bound와 likelihood의 값을 일치시키는 과정이 된다.
- 즉 정리해보면 미리 정해둔  $\theta$ 를 기반으로  $q$ 에 대한 분포를 계산하는 과정이다.



## • M-step

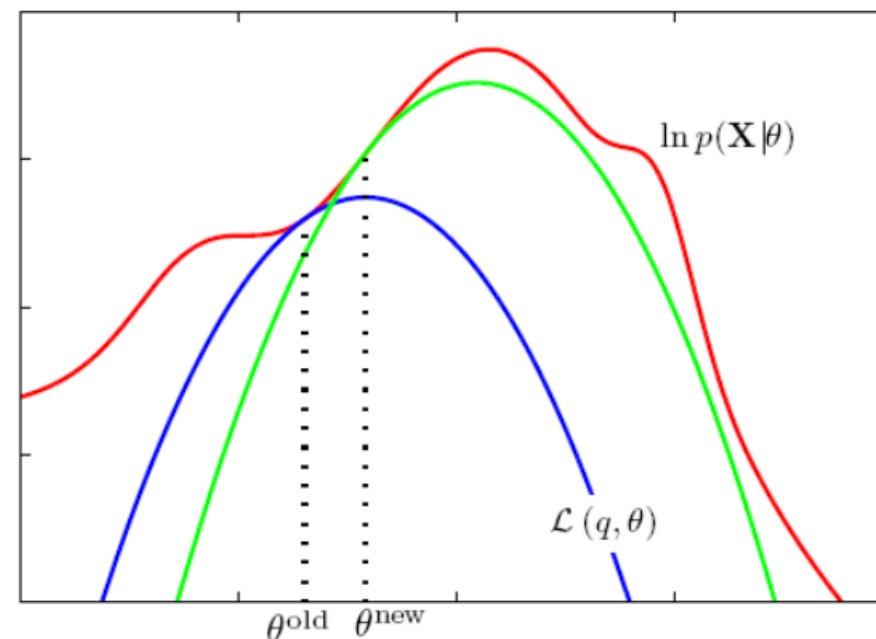
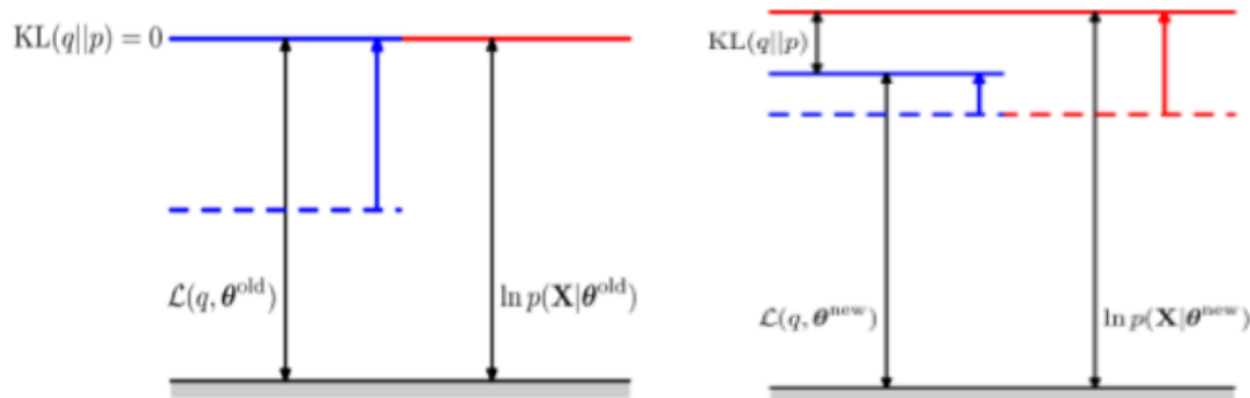
- M-step에서는 그 반대로,  $q(Z)$ 를 고정하고 log likelihood를 가장 크게 만드는  $\theta_{new}$ 를 찾는 optimization 문제를 푸는 단계이다.
- M-step에서는  $\theta$ 가 log likelihood에 직접 영향을 미치기 때문에 log likelihood 자체가 증가하게 된다.
- 즉 구한  $z$ 를 기반으로 다시 반복해서 데이터와 비교하는 과정이기 때문에  $\theta$ 가 업데이트가 되면서 log likelihood 자체가 증가하게 되는 것이다.



# The General EM Algorithm

- Summary

- 1) E-step :  $\theta_{old}$ 에서 log likelihood와 최대한 근사한 L을 얻는다. (파란색)
- 2) M-step : L을 최대화하는  $\theta_{new}$ 를 얻는다.
- 3) E-step :  $\theta_{new}$ 로부터 L을 새로 얻는다. (초록색)
- 4) 위의 E-M step을 수렴할 때까지 반복



# An example of EM Algorithm

- EM Algorithm 예시

- <COIN TOSS EXAMPLE>

- EM: state unknown 상태에서 probability 계산
  - 상황: 2개의 코인 A,B, 무작위로 코인을 선택하여 10번 toss
  - : toss 결과(앞/뒤) 만 알뿐, 어떤 코인을 던졌는지 모름
  - 목표 : 각 코인 A,B의 앞면 나올 확률을 추정해보자.

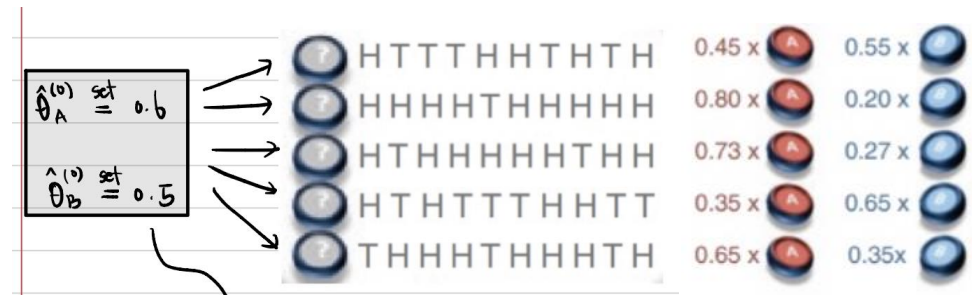
- 1) 초기화

- 각 코인 앞면 나올 확률을 임의로 설정

초기값	{	A의 앞면 : $\hat{\theta}_A^{(0)} \approx 0.6$
random 설정		B의 앞면 : $\hat{\theta}_B^{(0)} \approx 0.5$

- 2) E-step

- 결과를 바탕으로 사용된 코인이 A/B일 확률 계산



- 3) M-step

$$\hat{\theta}_A^{(1)} \approx \frac{21.3}{21.3 + 8.6} \approx 0.71$$

$$\hat{\theta}_B^{(1)} \approx \frac{11.7}{11.7 + 8.4} \approx 0.58$$

# 2

## Training Multilayer Perceptron Networks

---



# Training MLP

---

- EM Algorithm and Multiclass Classification

Assume multiclass classification with  $g$  groups,  $G_1, \dots, G_g$

**Problem:** Infer the unknown membership of an unclassified entity with feature vector of  $p$ -dimensions

Let  $(\mathbf{x}_1^T, \mathbf{y}_1^T)^T, \dots, (\mathbf{x}_n^T, \mathbf{y}_n^T)^T$  be the  $n$  examples available for training the neural network **and z be missing data or latent variable**

- EM uses **complete-data log likelihood** to estimate unknown parameters  $\Psi$

**1) E-Step:** Computes Q-function

$$\begin{aligned}\log L_c(\Psi; \mathbf{y}, \mathbf{z}, \mathbf{x}) &\propto \log \text{pr}(\mathbf{Y}, \mathbf{Z} | \mathbf{x}; \Psi) \\ &= \log \text{pr}(\mathbf{Y} | \mathbf{x}, \mathbf{z}; \Psi) \\ &\quad + \log \text{pr}(\mathbf{Z} | \mathbf{x}; \Psi)\end{aligned}$$

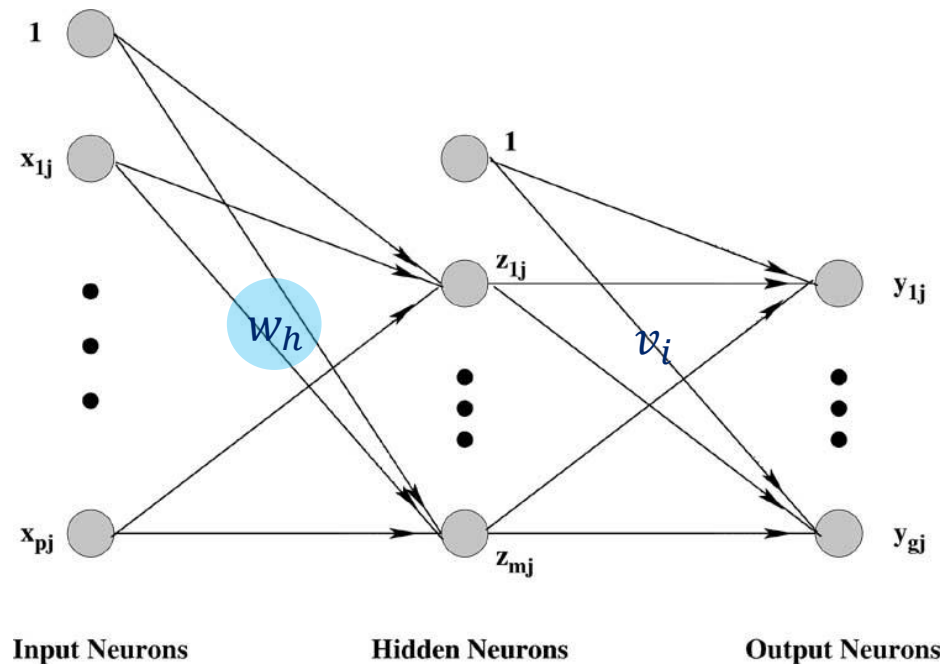
$$Q(\Psi; \Psi^{(k)}) = E_{\Psi^{(k)}} \{ \log L_c(\Psi; \mathbf{y}, \mathbf{z}, \mathbf{x}) | \mathbf{y}, \mathbf{x} \}$$

**2) M-Step**

:  $\Psi^{(k)}$  is updated by taking  $\Psi^{(k+1)}$  be the value of  $\Psi$  that maximizes Q-function

# Training MLP

MLP(Multi-Layer Perceptron) neural network with one hidden layer of  $m$  units



$z_{hj}$  be the realization of the zero-one random variable  $Z_{hj}$   
 $(h = 1, \dots, m ; j = 1, \dots, n)$

- Synaptic weight of the  $h$ th hidden unit:

$$\mathbf{w}_h = (w_{h0}, w_{h1}, \dots, w_{hp})^T$$

- Conditional distribution given  $\mathbf{x}_j$ :

$$pr(Z_{hj} = 1 \mid \mathbf{x}_j) = \frac{\exp(\mathbf{w}_h^T \mathbf{x}_j)}{1 + \exp(\mathbf{w}_h^T \mathbf{x}_j)}$$

sigmoid function

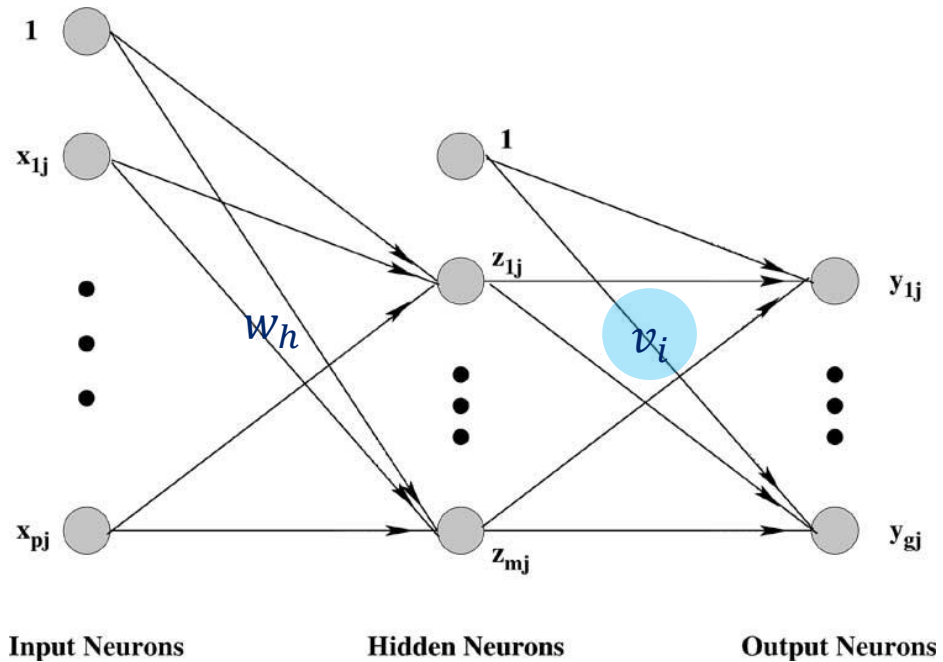
- The bias term ( $w_{h0}$ ) is included in  $\mathbf{w}_h$  by adding a constant input

$$x_{0j} = 1$$

$$\Rightarrow \text{input } \mathbf{x}_j = (x_{0j}, x_{1j}, \dots, x_{pj})^T$$

- Then,  $\mathbf{w}_h^T \mathbf{x}_j = \sum_{l=1}^p w_{hl} x_{lj} + w_{h0} = \sum_{l=0}^p w_{hl} x_{lj}$

# Training MLP



- Synaptic weight of the  $i$ th hidden unit:

$$\mathbf{v}_i = (v_{i0}, v_{i1}, \dots, v_{im})^T$$

for  $i = 1, \dots, g$

- Conditional distribution given  $\mathbf{x}_j, \mathbf{z}_j$ :

$$pr(Y_{ij} = 1 \mid \mathbf{x}_j, \mathbf{z}_j) = \frac{\exp(\mathbf{v}_i^T \mathbf{z}_j)}{\sum_{r=1}^g \exp(\mathbf{v}_r^T \mathbf{z}_j)}$$

softmax function

- The bias term ( $v_{i0}$ ) is included in  $\mathbf{v}_i$  by adding a constant hidden unit  $z_{0j} = 1$   
 $\Rightarrow$  hidden layer  $\mathbf{z}_j = (z_{0j}, z_{1j}, \dots, z_{mj})^T$
- Then,  $\mathbf{v}_i^T \mathbf{z}_j = \sum_{h=1}^m v_{ih} z_{hj} + v_{i0} = \sum_{h=0}^m v_{ih} z_{hj}$

# Training MLP

**Goal:** Find ML estimate for unknown parameters  $\Psi = (w_1^T, w_2^T, \dots, w_m^T, v_1^T, v_2^T, \dots, v_{g-1}^T)^T$  through **EM Steps** using complete-data log likelihood  $L_c(\Psi; \mathbf{y}, \mathbf{z}, \mathbf{x})$

- Likelihood Function ( $Z_{hj} \sim \text{Bernoulli}$ )

$$pr(\mathbf{Z}|\mathbf{x}; \Psi) = \prod_{j=1}^n \prod_{h=1}^m u_{hj}^{z_{hj}} (1 - u_{hj})^{(1-z_{hj})}$$

where

$$u_{hj} = pr(Z_{hj} = 1 | \mathbf{x}_j) = \frac{\exp(\sum_{l=0}^p w_{hl} x_{lj})}{1 + \exp(\sum_{l=0}^p w_{hl} x_{lj})}$$

$$(\mathbf{w}_h^T \mathbf{x}_j = \sum_{l=0}^p w_{hl} x_{lj} \text{ 를 } \frac{\exp(\mathbf{w}_h^T \mathbf{x}_j)}{1 + \exp(\mathbf{w}_h^T \mathbf{x}_j)} \text{에 대입})$$

- Likelihood Function ( $\mathbf{Y}_j \sim \text{Multinomial}$ )

$$pr(\mathbf{Y}|\mathbf{x}, \mathbf{z}; \Psi) = \prod_{j=1}^n \prod_{i=1}^g o_{ij}^{y_{ij}}$$

where

$$o_{ij} = pr(Y_{ij} = 1 | \mathbf{x}_j, \mathbf{z}_j) = \frac{\exp(\sum_{h=0}^m v_{ih} z_{hj})}{1 + \sum_{r=1}^{g-1} \exp(\sum_{h=0}^m v_{rh} z_{hj})}$$

$$o_{gj} = pr(Y_{ij} = 1 | \mathbf{x}_j, \mathbf{z}_j) = \frac{1}{1 + \sum_{r=1}^{g-1} \exp(\sum_{h=0}^m v_{rh} z_{hj})}$$

$$(\mathbf{v}_i^T \mathbf{z}_j = \sum_{h=0}^m v_{ih} z_{hj} \text{ 를 } \frac{\exp(\mathbf{v}_i^T \mathbf{z}_j)}{\sum_{r=1}^g \exp(\mathbf{v}_r^T \mathbf{z}_j)} \text{에 대입})$$

# Training MLP

---

**Goal:** Find ML estimate for unknown parameters  $\Psi = (w_1^T, w_2^T, \dots, w_m^T, v_1^T, v_2^T, \dots, v_{g-1}^T)^T$  through **EM Steps** using complete-data log likelihood  $L_c(\Psi; \mathbf{y}, \mathbf{z}, \mathbf{x})$

- Recall

$$\log L_c(\Psi; \mathbf{y}, \mathbf{z}, \mathbf{x}) \propto \log \text{pr}(\mathbf{Y}, \mathbf{Z} | \mathbf{x}; \Psi) = \log \text{pr}(\mathbf{Y} | \mathbf{x}, \mathbf{z}; \Psi) + \log \text{pr}(\mathbf{Z} | \mathbf{x}; \Psi)$$

- Then, the Complete-data log likelihood for  $\Psi$  is

$$\sum_{j=1}^n \left\{ \underbrace{\sum_{h=1}^m [z_{hj} \log \frac{u_{hj}}{1 - u_{hj}} + \log(1 - u_{hj})]}_{\text{Linear in } \mathbf{z}} + \underbrace{\sum_{i=1}^g y_{ij} \log o_{ij}}_{\text{Nonlinear}} \right\}$$

- We will calculate the expectation of the complete-data log likelihood  $\log L_c(\Psi; \mathbf{y}, \mathbf{z}, \mathbf{x})$  conditional on the current estimate  $\Psi^{(k)}$  and the observed input and output vectors

# Training MLP

---

- E-step

: Compute the Q-function

$$\begin{aligned} Q(\Psi; \Psi^{(k)}) &= E_{\Psi^{(k)}} \{ \log L_c(\Psi; \mathbf{y}, \mathbf{z}, \mathbf{x}) | \mathbf{y}, \mathbf{x} \} \\ &= \sum_{j=1}^n \sum_{h=1}^m \left[ E_{\Psi^{(k)}}(Z_{hj} | \mathbf{y}, \mathbf{x}) \right. \\ &\quad \left. \times \log \frac{u_{hj}}{1 - u_{hj}} + \log(1 - u_{hj}) \right] \\ &\quad + \sum_{j=1}^n \sum_{i=1}^g y_{ij} E_{\Psi^{(k)}}(o_{ij} | \mathbf{y}, \mathbf{x}) \\ &= Q_w + Q_v \end{aligned} \tag{11}$$

- Complete-data log likelihood에 대해 Expectation을 취하면 **(marginalize out all possible Z)** 다음과 같이 Q-function이 유도된다.
- Q-function은 가중치  $w, v$ 에 대한 식으로 각각 분해된다.
- 따라서 M-step에서  $Q_w, Q_v$ 를 각각 최대화 하는 과정을 통해  $w$ 와  $v$ 를 update할 수 있다.

# Training MLP

---

- M-step
- Set the differentiation of  $Q_w$  with respect to  $w$  as 0.
- Then we take  $\mathbf{w}_h^{(k+1)} = \operatorname{argmax} Q_w$
- Set the differentiation of  $Q_v$  with respect to  $v$  as 0.
- Then we take  $\mathbf{v}_i^{(k+1)} = \operatorname{argmax} Q_v$

$$\sum_{j=1}^n [E_{\Psi^{(k)}}(Z_{hj}|\mathbf{y}, \mathbf{x}) - u_{hj}] \mathbf{x}_j = \mathbf{0} \quad (h = 1, \dots, m) \quad (12)$$

where

$$E_{\Psi^{(k)}}(Z_{hj}|\mathbf{y}, \mathbf{x}) = \frac{\sum_{\mathbf{z}_j: z_{hj}=1} \operatorname{pr}_{\Psi^{(k)}}(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)}{\sum_{\mathbf{z}_j} \operatorname{pr}_{\Psi^{(k)}}(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)} \quad (13)$$

and where

$$\operatorname{pr}_{\Psi^{(k)}}(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j) = \prod_{h=1}^m u_{hj}^{z_{hj}} (1 - u_{hj})^{(1-z_{hj})} \prod_{i=1}^g o_{ij}^{y_{ij}}. \quad (14)$$

$$\sum_{j=1}^n \left[ y_{ij} E_{\Psi^{(k)}}(Z_{hj}|\mathbf{y}, \mathbf{x}) - \frac{\sum_{\mathbf{z}_j: z_{hj}=1} o_{ij} \operatorname{pr}_{\Psi^{(k)}}(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)}{\sum_{\mathbf{z}_j} \operatorname{pr}_{\Psi^{(k)}}(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)} \right] = 0. \quad (15)$$

- **M-step for gradient descent:** Since we cannot obtain our new parameters as a closed form.

# 3

## Training MLP : with Python

---



# Training MLP : with Python

## 1. 데이터 준비 및 전처리

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

# Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# One-hot encoding
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform(y.reshape(-1, 1))

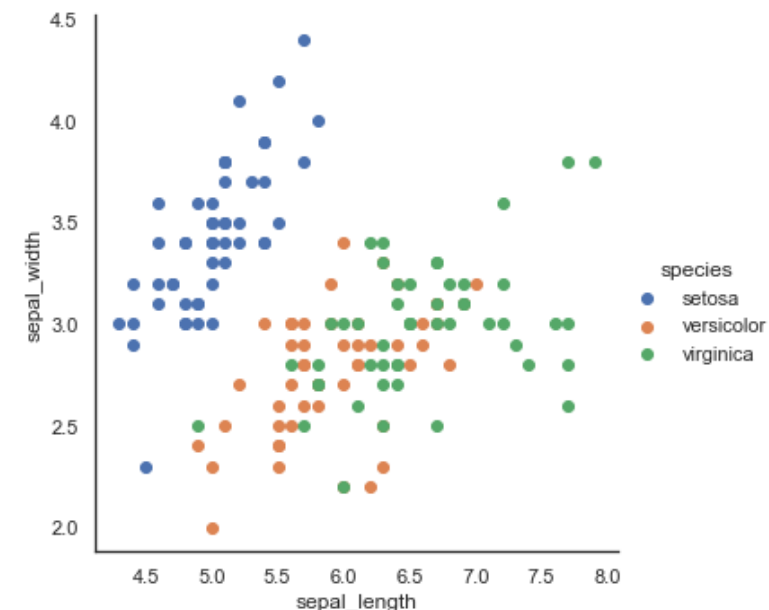
# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)#, random_state=42)
```

Iris 데이터셋 로드

One - hot encoding

학습 & 테스트 데이터  
분할

print(X)	print(y)
[5.1 3.5 1.4 0.2]	[1. 0. 0.]
[4.9 3. 1.4 0.2]	[1. 0. 0.]
[4.7 3.2 1.3 0.2]	[1. 0. 0.]
[4.6 3.1 1.5 0.2]	[1. 0. 0.]
[5. 3.6 1.4 0.2]	[1. 0. 0.]
[5.4 3.9 1.7 0.4]	[1. 0. 0.]
[4.6 3.4 1.4 0.3]	[1. 0. 0.]
[5. 3.4 1.5 0.2]	[1. 0. 0.]
[4.4 2.9 1.4 0.2]	[1. 0. 0.]
[4.9 3.1 1.5 0.1]	[1. 0. 0.]
[5.4 3.7 1.5 0.2]	[1. 0. 0.]
[4.8 3.4 1.6 0.2]	[1. 0. 0.]
[4.8 3. 1.4 0.1]	[0. 1. 0.]
[4.3 3. 1.1 0.1]	[0. 1. 0.]
[5.8 4. 1.2 0.2]	[0. 1. 0.]
[5.7 4.4 1.5 0.4]	[0. 1. 0.]
[5.4 3.9 1.3 0.4]	[0. 1. 0.]
[5.1 3.5 1.4 0.3]	[0. 1. 0.]
[5.7 3.8 1.7 0.3]	[0. 1. 0.]
[5.1 3.8 1.5 0.3]	[0. 1. 0.]
[5.4 3.4 1.7 0.2]	[0. 1. 0.]
[5.1 3.7 1.5 0.4]	[0. 1. 0.]



# Training MLP : with Python

## 2. 보조 함수 및 활성화 함수 정의

zlst(m) : 가능한 은닉층의 활성화 조합을 생성하는 함수

```
def zlst(m):
    zlst=np.zeros((2**m,m))
    for i in range(2**m):
        z=format(i, 'b').zfill(m)
        z=np.array(list(z))
        zlst[i,:]=z
    return zlst
```

```
import numpy as np
import random

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def softmax(z):
    exp_z = np.exp(z)
    return exp_z / np.sum(exp_z)
```

활성화 함수인 sigmoid와 softmax 함수 구현

# Training MLP : with Python

## 3. 신경망 클래스 정의 (by using EM

### Algorithm)

```
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, lr): #p,m,g
        self.p = input_size #p
        self.m = hidden_size #m
        self.g = output_size #g
        self.lr = lr
        self.zlst=zlst(self.m)

        # weight initialize & shape construction
        self.W = np.random.randn(self.p, self.m) #p*m
        self.V = np.random.randn(self.m, self.g) #m*g

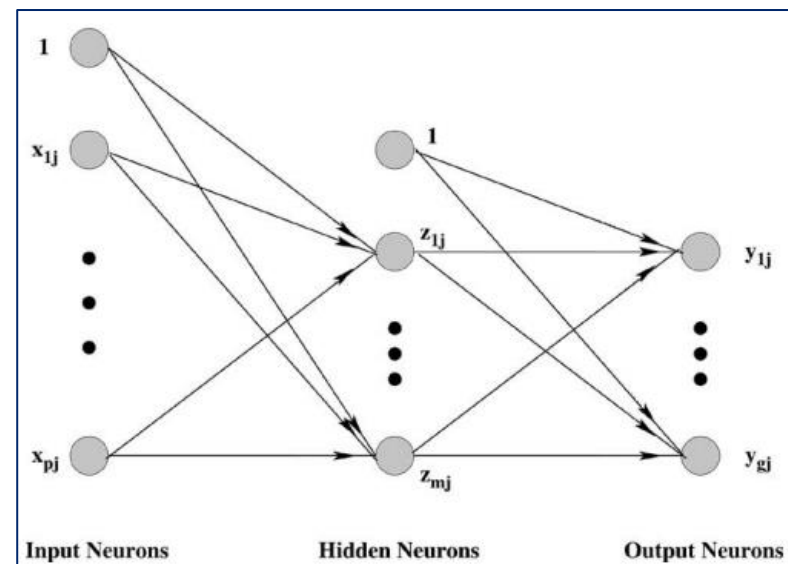
    def forward(self, X): #forward propagation

        self.A1 = self.W.T @ X #m*1
        self.U = sigmoid(self.A1) #m*1

        self.A2 = self.V.T @ self.U #g*1
        self.O = softmax(self.A2) #g*1
        return self.O
```

모델의 가중치 초기화

Forward Propagation 구현



$$\text{pr}(Z_{hj} = 1 | \mathbf{x}_j) = \frac{\exp(\mathbf{w}_h^T \mathbf{x}_j)}{1 + \exp(\mathbf{w}_h^T \mathbf{x}_j)} \quad (5)$$

$$\text{pr}(Y_{ij} = 1 | \mathbf{x}_j, \mathbf{z}_j) = \frac{\exp(\mathbf{v}_i^T \mathbf{z}_j)}{\sum_{r=1}^g \exp(\mathbf{v}_r^T \mathbf{z}_j)} \quad (6)$$

# Training MLP : with Python

## 3. 신경망 클래스 정의 – E step 구현(1)

```
def E_step_W(self, j, X, y):
    self.forward(X[j,:])

    sumz=0
    for z in self.zlst:
        pr_xyz=1
        for h in range(self.m):
            pr_xyz*=self.U[h]**z[h]*(1-self.U[h])**((1-z[h]))
        for i in range(self.g):
            pr_xyz*=softmax(self.V.T @ z)[i]**y[j,i]
        sumz+=pr_xyz

    sumzh=0
    for z in self.zlsth:
        pr_xyz=1
        for h in range(self.m):
            pr_xyz*=self.U[h]**z[h]*(1-self.U[h])**((1-z[h]))
        for i in range(self.g):
            pr_xyz*=softmax(self.V.T @ z)[i]**y[j,i]
        sumzh+=pr_xyz

    return sumz, sumzh
```

Expectation(summation) algorithm for M-step to update w

$$\begin{aligned} Q(\Psi; \Psi^{(k)}) &= E_{\Psi^{(k)}} \{ \log L_c(\Psi; \mathbf{y}, \mathbf{z}, \mathbf{x}) | \mathbf{y}, \mathbf{x} \} \\ &= \sum_{j=1}^n \sum_{h=1}^m \left[ E_{\Psi^{(k)}}(Z_{hj} | \mathbf{y}, \mathbf{x}) \right. \\ &\quad \left. \times \log \frac{u_{hj}}{1 - u_{hj}} + \log(1 - u_{hj}) \right] \\ &\quad + \sum_{j=1}^n \sum_{i=1}^g y_{ij} E_{\Psi^{(k)}}(o_{ij} | \mathbf{y}, \mathbf{x}) \\ &= Q_w + Q_v \end{aligned} \quad (11)$$

$$\sum_{j=1}^n [E_{\Psi^{(k)}}(Z_{hj} | \mathbf{y}, \mathbf{x}) - u_{hj}] \mathbf{x}_j = 0 \quad (h = 1, \dots, m) \quad (12)$$

# Training MLP : with Python

## 3. 신경망 클래스 정의 – E step 구현(2)

```
def E_step_V(self, j, X, y, i):
    self.forward(X[j,:])

    sumz=0
    for z in self.zlst:
        pr_xyz=1
        for h in range(self.m):
            pr_xyz*=self.U[h]**z[h]*(1-self.U[h])**((1-z[h]))
        for i in range(self.g):
            pr_xyz*=softmax(self.V.T @ z)[i]**y[j,i]
        sumz+=pr_xyz

    sumzy=0
    for z in self.zlsth:
        pr_xyz=1
        for h_ in range(self.m):
            pr_xyz*=self.U[h_]**z[h_]*(1-self.U[h_])**((1-z[h_]))
        for i_ in range(self.g):
            pr_xyz*=softmax(self.V.T @ z)[i_]**y[j,i_]
        pr_xyz*=y[j,i]-softmax(self.V.T @ z)[i]
        sumzy+=pr_xyz

    return sumz, sumzy
```

Expectation(summation) algorithm for M-step to update v

$$\begin{aligned} Q(\Psi; \Psi^{(k)}) &= E_{\Psi^{(k)}} \{ \log L_c(\Psi; \mathbf{y}, \mathbf{z}, \mathbf{x}) | \mathbf{y}, \mathbf{x} \} \\ &= \sum_{j=1}^n \sum_{h=1}^m \left[ E_{\Psi^{(k)}}(Z_{hj} | \mathbf{y}, \mathbf{x}) \right. \\ &\quad \left. \times \log \frac{u_{hj}}{1 - u_{hj}} + \log(1 - u_{hj}) \right] \\ &\quad + \sum_{j=1}^n \sum_{i=1}^g y_{ij} E_{\Psi^{(k)}}(o_{ij} | \mathbf{y}, \mathbf{x}) \\ &= Q_w + Q_v \end{aligned} \quad (11)$$

$$\sum_{j=1}^n \left[ y_{ij} E_{\Psi^{(k)}}(Z_{hj} | \mathbf{y}, \mathbf{x}) - \frac{\sum_{\mathbf{z}_j: z_{hj}=1} o_{ij} \Pr_{\Psi^{(k)}}(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)}{\sum_{\mathbf{z}_j} \Pr_{\Psi^{(k)}}(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)} \right] = 0. \quad (15)$$

# Training MLP : with Python

## 3. 신경망 클래스 정의 – M step 구현

```
def M_step(self, X, y): #EM algorithm

    grad_W = np.zeros((self.p, self.m)) #p*m
    grad_V = np.zeros((self.m, self.g)) #m*g

    for h in range(self.m):
        grad_Wh=0
        self.zlsth=self.zlst[self.zlst[:,h]==1]
        for j in range(len(X)):
            sumz, sumzh=self.E_step_W(j,X,y)
            grad_Wh += sumzh/sumz-self.U[h]*X[j,:]

        grad_W[:,h]=grad_Wh

    #####

    for h in range(self.m):
        self.zlsth=self.zlst[self.zlst[:,h]==1]
        for i in range(self.g):
            for j in range(len(X)):
                sumz, sumzy = self.E_step_V(j, X, y, i)
                grad_V[h,i] += sumzy/sumz

    #####

    # update weight & bias
    self.W += grad_W * self.lr
    self.V += grad_V * self.lr
```

$$\sum_{j=1}^n [E_{\Psi^{(k)}}(Z_{hj}|\mathbf{y}, \mathbf{x}) - u_{hj}] \mathbf{x}_j = 0 \quad (h = 1, \dots, m) \quad (12)$$

$$\sum_{j=1}^n \left[ y_{ij} E_{\Psi^{(k)}}(Z_{hj}|\mathbf{y}, \mathbf{x}) - \frac{\sum_{\mathbf{z}_j: z_{hj}=1} o_{ij} \text{Pr}_{\Psi^{(k)}}(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)}{\sum_{\mathbf{z}_j} \text{Pr}_{\Psi^{(k)}}(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)} \right] = 0. \quad (15)$$

# Training MLP : with Python

## 4. 모델 학습 및 평가

```
def Train(self, X, y, epochs):
    for epoch in range(epochs):

        self.M_step(X, y)

        losses=list()
        for n in range(len(X)):
            y_pred = self.forward(X[n,:])
            loss = -np.sum(y[n,:]*np.log(y_pred)) #Cross Entropy Loss
            losses.append(loss)
        avgloss=np.mean(losses)

        if (epoch+1) % 1 == 0:
            print(f'Epoch {epoch+1}, Loss: {avgloss}')

def Test(self, X):
    testoutput=[]
    for n in range(len(X)):
        y_pred = self.forward(X[n,:])
        testoutput.append(np.argmax(y_pred))
    return testoutput
```

*Train* : 주어진 epochs만큼 모델 학습 및 손실 값 출력

*Test* : 테스트 데이터를 사용하여 예측 수행

# Optimization via EM vs Backpropagation

## 5. 실험 결과1: Cross Entropy Loss 관측

### EM

*#setting hyperparameters*

epochs=50

lr=0.005

NN=NeuralNetwork(4,7,3,lr)

NN.Train(X\_train,y\_train,epochs)

Epoch 0, Loss: 1.1427729846642725  
Epoch 5, Loss: 0.8477192072820486  
Epoch 10, Loss: 0.8088932560503146  
Epoch 15, Loss: 0.8013409928099052  
Epoch 20, Loss: 0.7953260351338284  
Epoch 25, Loss: 0.7862787851641184  
Epoch 30, Loss: 0.6765491949913677  
Epoch 35, Loss: 0.6733092653604303  
Epoch 40, Loss: 0.670328548725725  
Epoch 45, Loss: 0.6676518890360496  
Epoch 50, Loss: 0.6652140857718162

### Backpropagation

*#setting hyperparameters*

learning\_rate=0.005

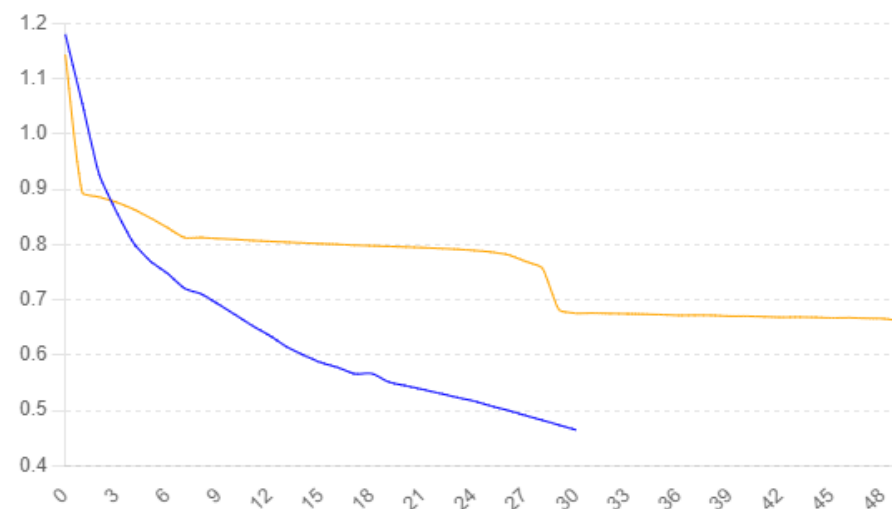
epochs=30

NN=NeuralNetwork(4,7,3, learning\_rate)

NN.Train(X\_train,y\_train,epochs)

Epoch 0, Loss: 1.1799459307487936  
Epoch 5, Loss: 0.7697343605798745  
Epoch 10, Loss: 0.6724774620576618  
Epoch 15, Loss: 0.5865647960528668  
Epoch 20, Loss: 0.5445126310389149  
Epoch 25, Loss: 0.5078813746853241  
Epoch 30, Loss: 0.46463978978466247

Y Loss X Epochs Backpropagation Loss EM Algorithm Loss





# Optimization via EM vs Backpropagation

## 5. 실험 결과2: 정확도 & test data 10개 sample

### EM

```
testoutput = NN.Test(X_test)
y_test_labels = np.argmax(y_test, axis=1)

#accuracy
accuracy = np.mean(testoutput == y_test_labels)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Accuracy: 93.33%

```
for i in range(10):
    pred=NN.forward(X_test[i,:])
    target=y_test[i,:]
    print(pred,target,np.argmax(pred)==np.argmax(target))
```

```
[0.936249  0.04692205 0.01682896] [1. 0. 0.] True
[0.27988311 0.32700521 0.39311167] [0. 0. 1.] True
[0.28207146 0.33332921 0.38459933] [0. 0. 1.] True
[0.95591958 0.03276875 0.01131167] [1. 0. 0.] True
[0.27057917 0.31213924 0.4172816 ] [0. 0. 1.] True
[0.33065804 0.38445328 0.28488868] [0. 1. 0.] True
[0.95479909 0.03355867 0.01164225] [1. 0. 0.] True
[0.27467353 0.31926968 0.4060568 ] [0. 0. 1.] True
[0.94697415 0.03924066 0.01378519] [1. 0. 0.] True
[0.31662273 0.37827904 0.30509823] [0. 1. 0.] True
```

### Backpropagation

```
testoutput = NN.Test(X_test)
y_test_labels = np.argmax(y_test, axis=1)

#accuracy
accuracy = np.mean(testoutput == y_test_labels)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Accuracy: 96.67%

```
for i in range(10):
    pred=NN.forward(X_test[i,:])
    target=y_test[i,:]
    print(pred,target,np.argmax(pred)==np.argmax(target))
```

```
[0.79087513 0.17280196 0.03632291] [1. 0. 0.] True
[0.0147225  0.34725775 0.63801976] [0. 0. 1.] True
[0.01470303 0.33480121 0.65049575] [0. 0. 1.] True
[0.83338793 0.13828392 0.02832815] [1. 0. 0.] True
[0.00902271 0.33238056 0.65859673] [0. 0. 1.] True
[0.17068413 0.52608564 0.30323023] [0. 1. 0.] True
[0.8299676  0.14205715 0.02797525] [1. 0. 0.] True
[0.01058652 0.32585278 0.6635607 ] [0. 0. 1.] True
[0.81611671 0.15257764 0.03130565] [1. 0. 0.] True
[0.13539729 0.5567226  0.30788011] [0. 1. 0.] True
```

# 4

## Discussion & Conclusion

---

# Discussion

---

## 1. 코드 구현 과정에서 개선할 점

### 1. E-step의 효율성

: E-step에서 가능한 모든 은닉층의 조합을 반복하여 확률을 계산하는 과정은 계산 비용이 매우 높음 (실제로 코드 실행까지 걸리는 시간이 길다)

→ 샘플링 또는 근사 방법을 사용하여 계산 비용을 줄이는 방법을 채택해볼 수 있음.

### 2. Hyperparameter 튜닝

: 모델 학습 과정에서 학습률(lr)과 은닉층 유닛의 수(m) 등의 hyperparameter를 튜닝하여 모델의 성능을 향상시키는 방법을 생각해볼 수 있음.

### 3. 입력 데이터의 정규화(Normalization)

: 입력 데이터를 정규화하여 모든 입력 특성이 동일 범위를 갖게 되면 특정 값이 다른 값보다 지나치게 큰 영향을 주는 것을 방지할 수 있고, M-step의 최적화 알고리즘이 더욱 빠르게 수렴하게끔 도와주며, 오버플로우 또는 언더플로우 문제를 줄여줄 수 있음.

# Discussion

## 2. Backpropagation 방법론 신경망 학습과의 비교

### Backpropagation

일반적으로 GD 방법이 더 빠르게 수렴하며,  
복잡한 손실함수 공간에서도 잘 작동함

반복적인 손실 함수의 경사를 계산, 일반적으로 효율  
적

손실 함수가 단순히 예측값과 실제값 간의 차이를 나  
타내기 때문에, 데이터의 분포가 무엇인지와는 무관하  
게 작동함

VS

### Expectation – Maximization

E-step과 M-step이 교차하며 수렴 속도가 느려질 수  
있고, 특히 고차원 데이터에서 결과값의 수렴이 어려  
울 수 있음

가능한 모든 은닉층 유닛의 조합을 반복 계산,  
높은 계산 비용이 필요하며, 큰 데이터셋에서 비효율  
적

잠재 변수의 추정을 기반으로 하기에,  
가정된 분포에 대한 해석이 부족하면 성능 저하 우려

# Conclusion

---

EM을 shallow Neural Network 이상으로 Deep Learning에 적용하기에는 적합하지 않다

## 1. 시간복잡도

: shallow network에서 hidden layer의 node수에 따른  $O(2^m * )$ 의 시간복잡도 발생, 2개 이상의 hidden layer에 대해서는 요구되는 연산량이 매우 증폭될 것으로 예상됨

## 2. 수렴 속도

: 실험 결과를 통해 확인한 수렴 속도는 backpropagation method보다 현저히 느림. 앞서 언급한 EM algorithm의 상당한 시간복잡도를 고려한다면 더욱 비효율적

## 3. 데이터의 분포 가정

: Deep Learning에서는 모델 파라미터와 데이터 간의 관계에 대한 비선형성이 더욱 강화되므로 latent variable  $z$ 에 대한 분포를 적절하게 가정하기 어려움

---

감사합니다