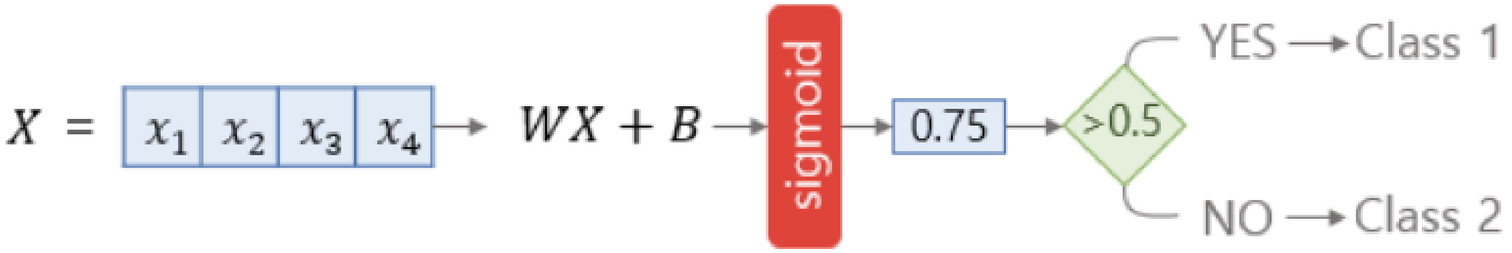
1조 SAI인머스켓

[PyTorch] Lab-05 Logistic Regression
[PyTorch] Lab-06 Softmax Classification
[PyTorch] Lab-07-1 Tips

로지스틱 회귀(Logistic Regression)

이진 분류를 풀기 위한 대표적인 알고리즘



가설 : $H(X) = sigmoid(WX + B)$

로지스틱 회귀에서는 선형 회귀처럼 직선 함수로 분류하는 것이 아닌 S자 모양의 함수를 이용해 분류

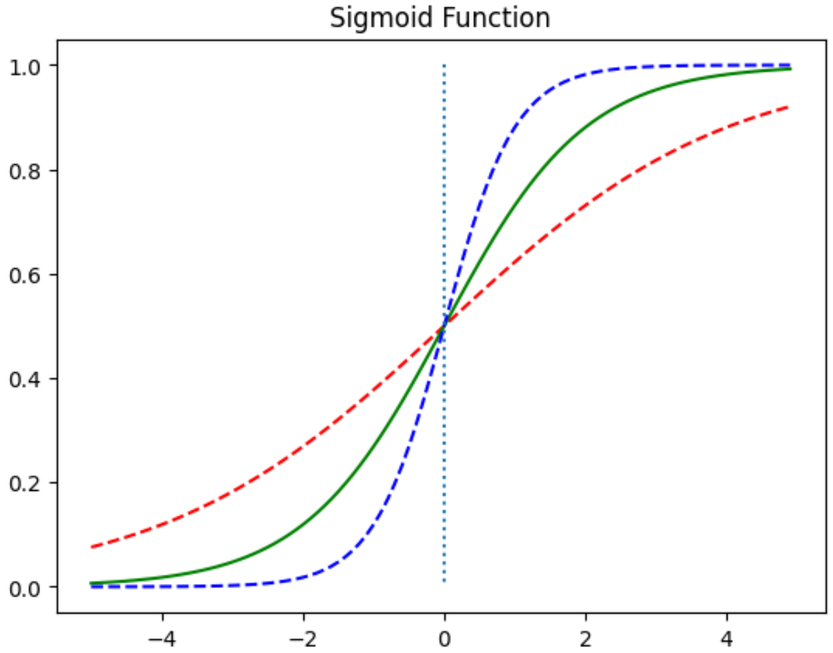
S자 모양의 함수 = 시그모이드(Sigmoid) 함수

시그모이드 함수(Sigmoid function)

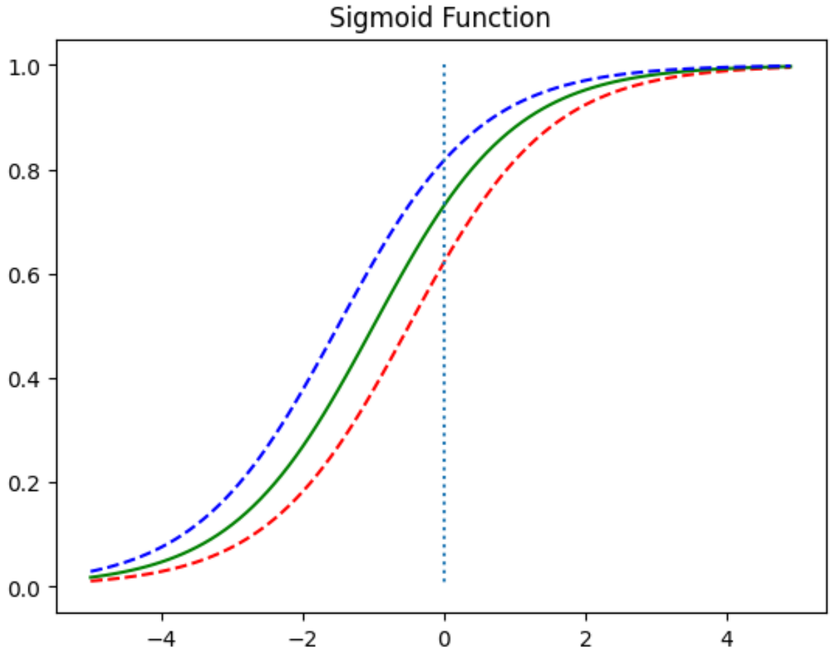
$$H(x) = sigmoid(Wx + b) = \frac{1}{1 + e^{-(Wx+b)}} = \sigma(Wx + b)$$

시그모이드 함수의 출력값은
0과 1사이의 값을 가지는데, 이 특성을 이용해
0.5이상은 1(True), 0.5이하는 0(False)로
분류를 진행한다

W(기울기)와 b(y절편)이 그래프에 영향을 준다



W의 값이 커지면 그래프의 경사가 커지고,
W의 값이 작아지면 그래프의 경사가 작아진다



b의 값에 따라서 그래프가 좌, 우로 이동한다

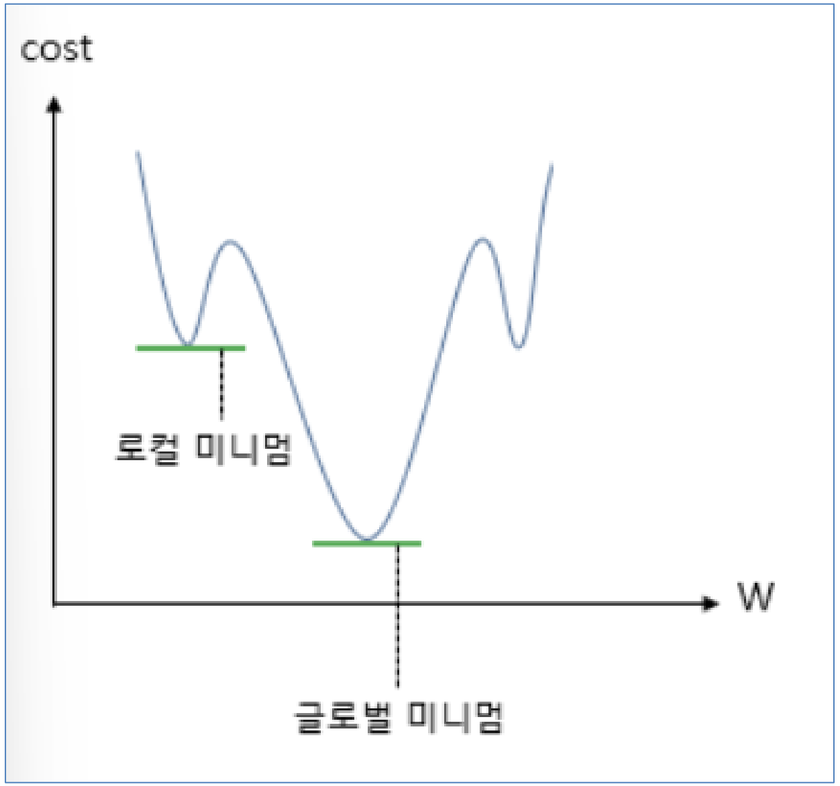
비용 함수(Cost function)

$$cost(W,b) = \frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} - H(x^{(i)})\right]^2$$

만약 로지스틱 회귀에
선형 회귀에서 사용한 MSE(평균제곱오차)를 사용하면

전체에서 오차가 최소값이 되는 구간을 찾지 못하고

특정 구역에서 오차가 최솟값이 되는 구간을 찾아버리는
문제가 생김



비용 함수(Cost function)

$$cost(W) = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}logH(x^{(i)}) + (1-y^{(i)})log(1-H(x^{(i)}))]$$

로지스틱 회귀에서는

이러한 문제를 해결하기 위해 위와 같은 수식의 비용함수를 사용하는데
이를 Binary Cross Entropy라고 한다

이 비용 함수에 대해서 경사 하강법을 수행해 최적의 가중치 W를 찾는다

파이토치로 로지스틱 회귀 구현

비용함수나 모델 초기화 방법의 차이로
다양하게 구현할 수 있다

강의에서는 4가지 방법을 다룸

Low-level Binary Cross Entropy Loss

```
# 모델 초기화
W = torch.zeros((2, 1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)

# Cost 계산
hypothesis = torch.sigmoid(x_train.matmul(W) + b)
cost = -(y_train * torch.log(hypothesis) +
    (1 - y_train) * torch.log(1 -hypothesis)).mean()
```

F.binary_cross_entropy

```
# 모델 초기화
W = torch.zeros((2, 1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)

# Cost 계산
hypothesis = torch.sigmoid(x_train.matmul(W) + b)
cost = F.binary_cross_entropy(hypothesis, y_train)
```

nn.Module(nn.Linear, nn.Sigmoid)

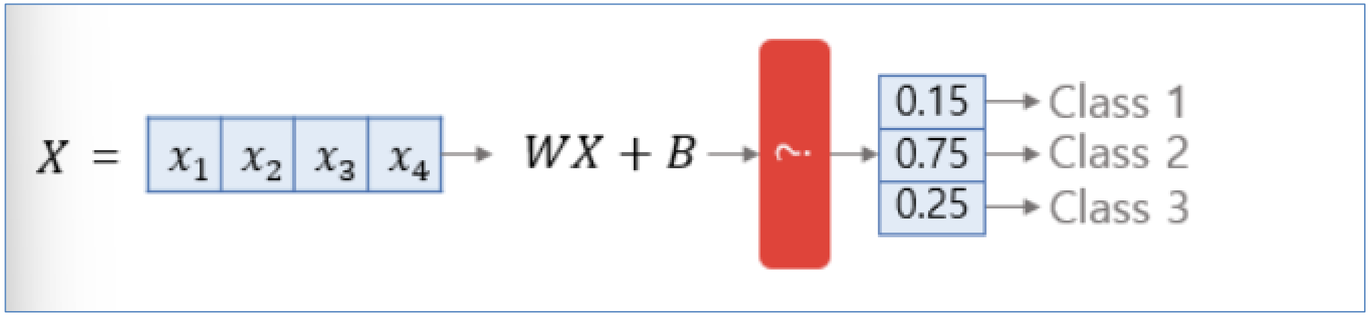
```
# 모델 초기화
model = nn.Sequential(
   nn.Linear(2, 1), # input_dim = 2, output_dim = 1
   nn.Sigmoid() # 출력은 시그모이드 함수를 거친다
)

# H(x) 계산
hypothesis = model(x_train)
# cost 계산
cost = F.binary_cross_entropy(hypothesis, y_train)
```

nn.Module(클래스로 구현)

```
# 모델 초기화
class BinaryClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(2, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        return self.sigmoid(self.linear(x))

model = BinaryClassifier()

# H(x) 계산
hypothesis = model(x_train)
# cost 계산
cost = F.binary_cross_entropy(hypothesis, y_train)
```

소프트맥스 회귀(Softmax Regression)

이진 분류가 아닌, 3개 이상의 답 중 하나를 고르는 다중 분류를 풀기 위한 대표적인 알고리즘



가설 : $H(X) = softmax(WX + B)$

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad for\ i = 1, 2, \ldots k$$

i번째 클래스가 정답일 확률

$$softmax(z) = [\frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} \ \frac{e^{z_2}}{\sum_{j=1}^{3} e^{z_j}} \ \frac{e^{z_3}}{\sum_{j=1}^{3} e^{z_j}}] = [p_1, p_2, p_3] = \hat{y} = \text{예측값}$$

예를 들어 클래스가 3개이면 이와 같이 p1, p2, p3를 구함

원-핫 인코딩(One-hot encoding), 원-핫 벡터(one-hot vector)

원-핫 인코딩은 선택해야 하는 선택지의 개수만큼의 차원을 가지면서
각 선택지의 인덱스에 해당하는 원소에는 1, 나머지 원소는 0의 값을 가지도록 하는 표현 방법

강아지 = [1, 0, 0]

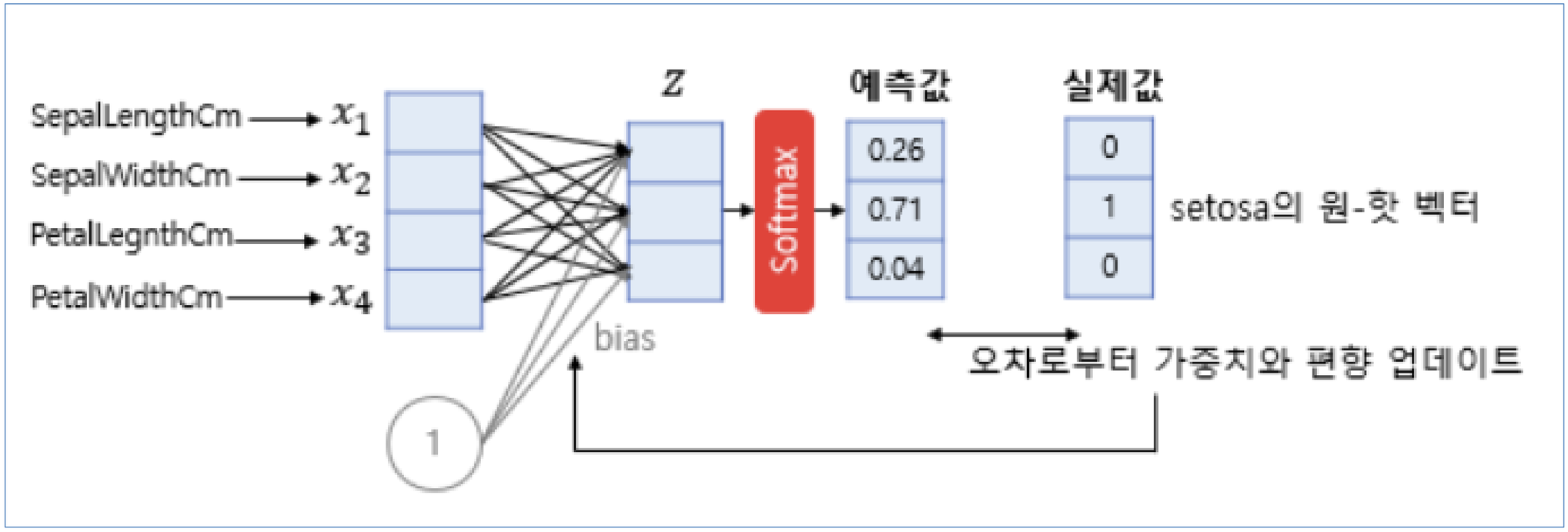
고양이 = [0, 1, 0]

냉장고 = [0, 0, 1]

원-핫 벡터는 원-핫 인코딩으로 표현된 벡터
원-핫 벡터로 표현해야만 다중 클래스 분류 문제를 풀 수 있는 것은 아니지만,
정수 인코딩과 달리 원-핫 인코딩은 모든 클래스 간의 관계를 균등하게 분배해서,
클래스의 성질을 잘 표현한다는 장점이 있다(무작위성).

원-핫 인코딩(One-hot encoding), 원-핫 벡터(one-hot vector)

이러한 원-핫 벡터를 이용해 오차로부터 가중치와 편향을 업데이트 한다



비용 함수(Cost function)

소프트맥스 회귀에서는 로지스틱 회귀에서와 동일한 비용 함수인
크로스 엔트로피 함수를 사용

$$cost(W) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_j^{(i)}\ log(p_j^{(i)})$$

이진 분류일 때는 k 값을 2로 설정하면 되는데

이는 로지스틱 회귀에서 사용한 Binary Cross Entropy 와 동일해
결국에는 동일한 함수를 사용했다는 것을 알 수 있다

파이토치로 소프트맥스 회귀 구현

비용함수나 모델 초기화 방법의 차이로
다양하게 구현할 수 있다

강의에서는 3가지 방법을 다룸

Low-level Cross Entropy Loss

```
# Cost 계산

hypothesis = F.softmax(x_train.matmul(W) + b,dim=1) y_one_hot =

torch.zeros_like(hypothesis)

y_one_hot.scatter_(1, y_train.unsqueeze(1), 1)

cost = (y_one_hot * -

torch.log(F.softmax(hypothesis,dim=1))).sum(dim=1).mean()
```

F._cross_entropy

```
# Cost 계산

    z = x_train.matmul(W) + b

    cost = F.cross_entropy(z, y_train)
```

nn.Module

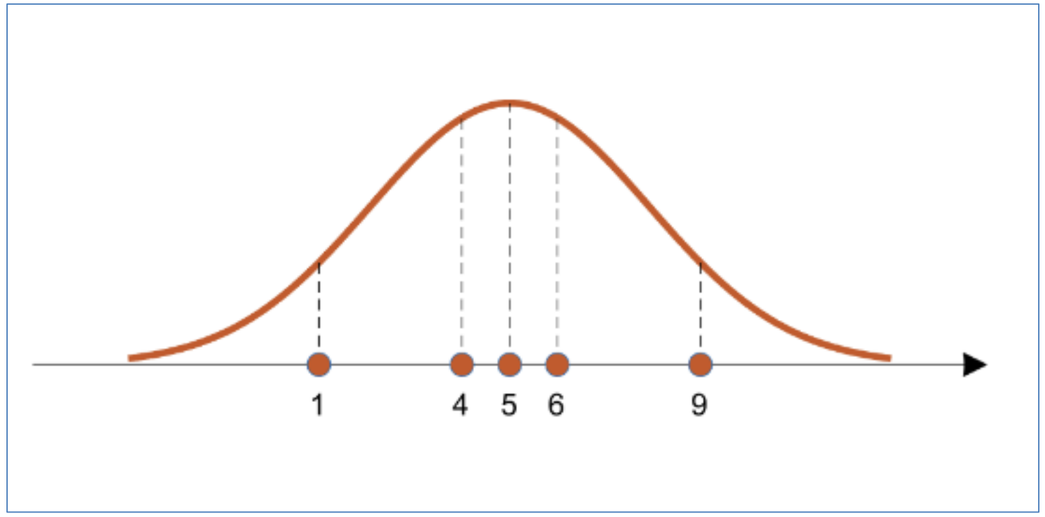
```
# 모델 초기화

class SoftmaxClassifierModel(nn.Module):

    def __init__(self):

        super().__init__()

        self.linear = nn.Linear(4, 3)


    def forward(self, x):

        return self.linear(x)


model = SoftmaxClassifierModel()


# H(x) 계산

prediction = model(x_train)


# cost 계산

cost = F.cross_entropy(prediction, y_train)
```

Tips - 최대우도법, Maximum Likelihood Estimation (MLE)

데이터와 제일 잘 맞는 모델과 추정치를 계산하는 방법

파라미터 추정 및 모델 선택 등 다양한 분야에서 중요한 역할을 한다

MLE는 다양한 통계 모델 및 머신러닝 알고리즘에서 사용된다
(선형 회귀, 로지스틱 회귀, 가우시안 혼합 모델, 신경망 등)



$$P(x|\theta) = \prod_{k=1}^{n} P(x_k|\theta)$$

likelihood function

$$L(\theta|x) = \log P(x|\theta) = \sum_{i=1}^{n} \log P(x_i|\theta)$$

log-likelihood function

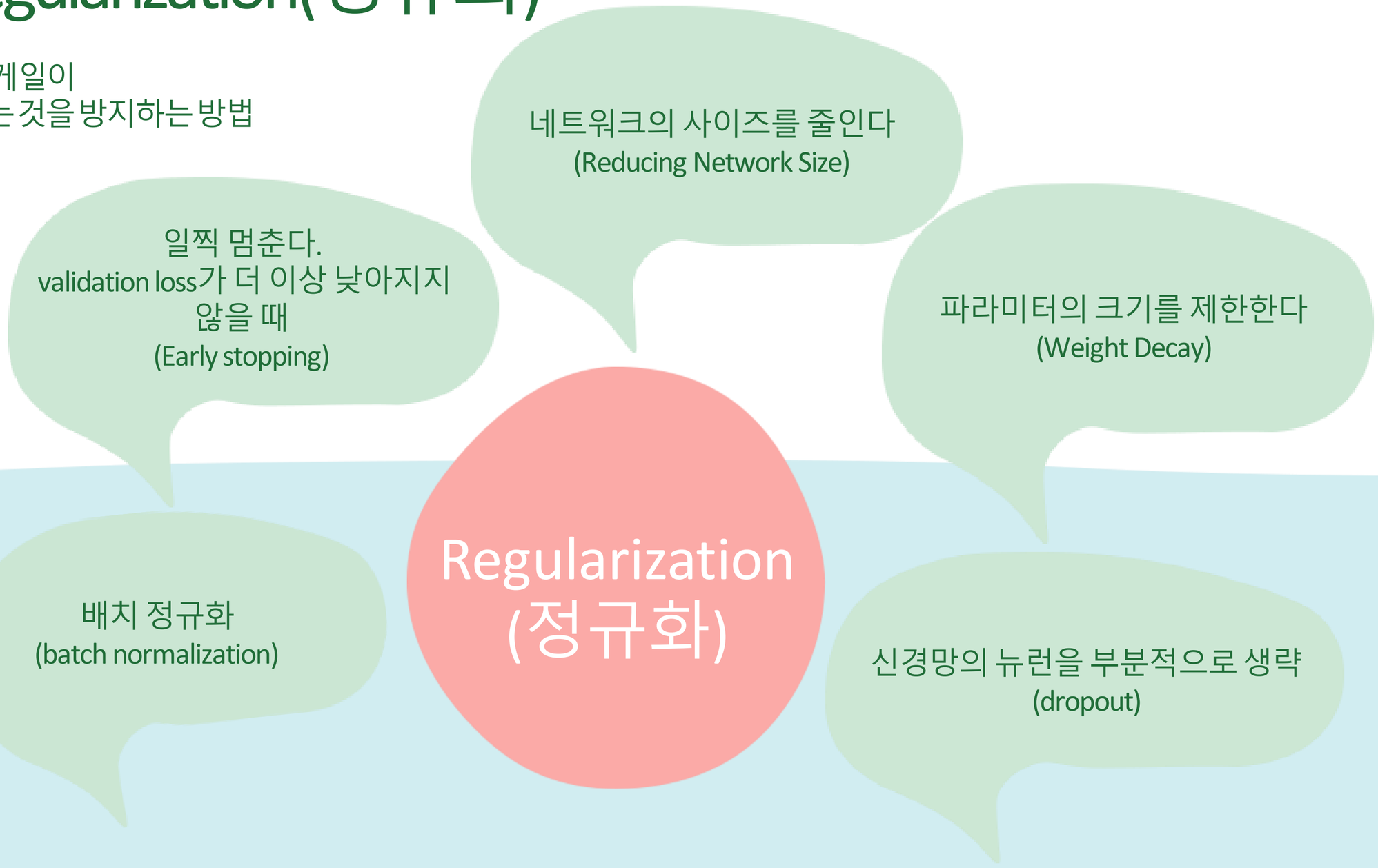
Tips – Overfitting(과적합)

학습이 지나치게 잘 되어 과도하게 적합하는 문제
학습이 잘 되는 것이 무조건 좋은 것이 아님
test set에 대한 결과가 좋지 않을 수 있다

해결 방법은 크게 3가지

Tips - Regularization(정규화)

특징 값들의 스케일이
심하게 차이나는 것을 방지하는 방법



감사합니다