2021 Spring

Artificial Intelligence & Deep Learning

Prof. Minsuk Koo

Department of Computer Science & Engineering
Incheon National University

5.2.5 활성 함수

- 활성값 z를 계산하고 활성함수 τ를 적용하는 과정

$$z = \mathbf{w}^{\mathrm{T}}\tilde{\mathbf{x}} + b$$
$$y = \tau(z) \qquad (5.15)$$

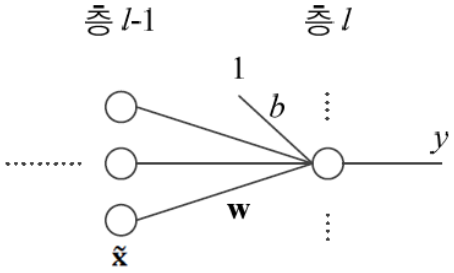


그림 5-14 신경망 노드의 연산

- 시대별 활성함수

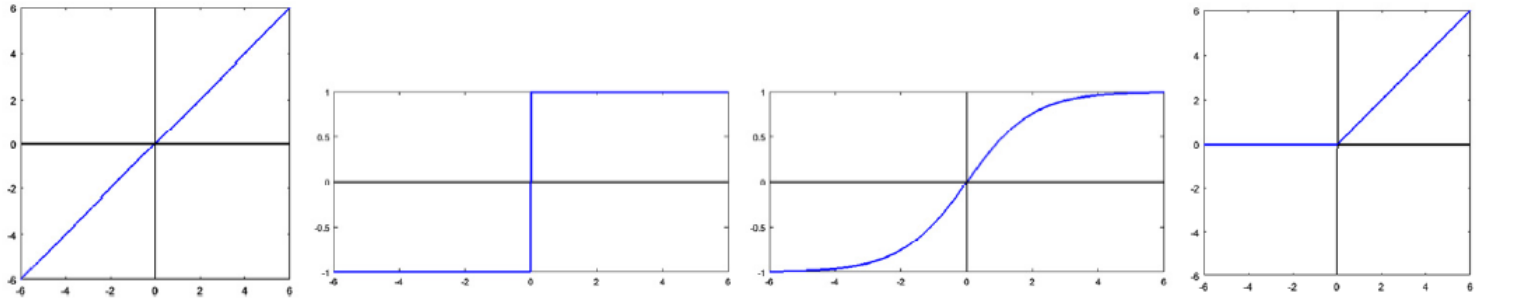


그림 5-15 활성함수 τ

  - tanh는 활성값이 커지면 포화 상태가 되고 그레이디언트는 0에 가까워짐 → 매개변수 갱신(학습)이 매우 느린 요인

5.2.5 활성 함수

- ReLU(Rectified Linear Unit) 활성함수
  - 포화 문제 해소

$$z = \mathbf{w}^{\mathrm{T}}\tilde{\mathbf{x}} + b$$
$$y = \mathrm{ReLU}(z) = \max(0, z) \qquad (5.16)$$

- ReLU의 변형
  - Leaky ReLU (보통 $\alpha = 0.01$을 사용)
  - PReLU (α를 학습으로 알아냄)

$$\mathrm{leakyReLU}(z) = \begin{cases} z, & z \geq 0 \\ az, & z < 0 \end{cases} \qquad (5.17)$$

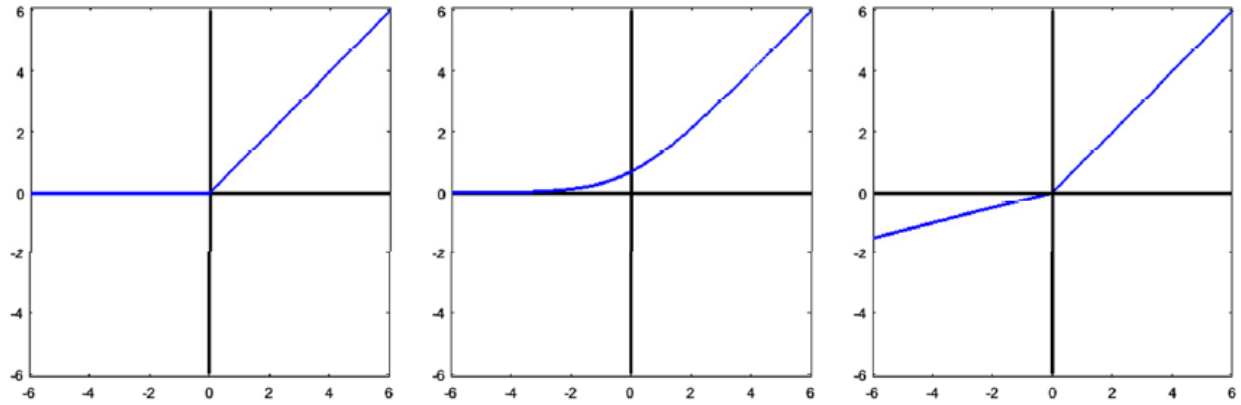


그림 5-16 ReLU의 변형

5.2.6 배치 정규화

- 공변량 시프트covariate shift 현상
  - 학습이 진행되면서 층1의 매개변수가 바뀜에 따라 $\tilde{x}^{(1)}$이 따라 바뀜 → 층2 입장에서 보면 자신에게 입력되는 데이터의 분포가 수시로 바뀌는 셈
  - 층2, 층3, ...으로 깊어짐에 따라 더욱 심각
  - 학습을 방해하는 요인으로 작용

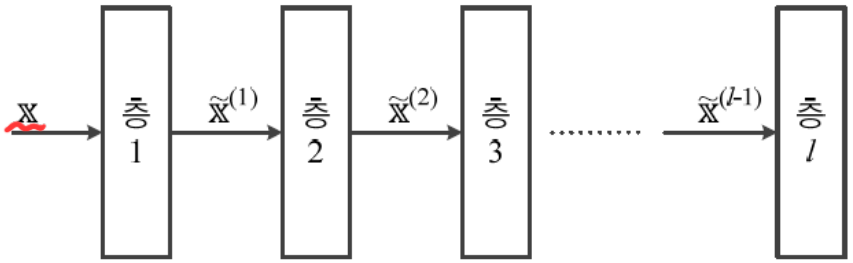
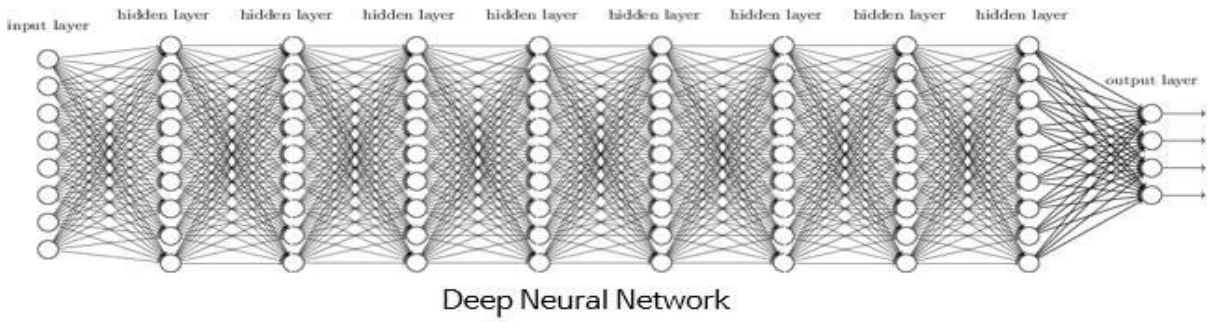
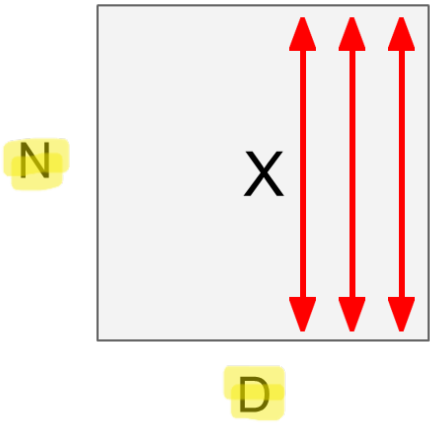


그림 5-17 공변량 시프트 현상

5.2.6 배치 정규화





1. compute the empirical mean and variance independently for each di mension.

$z = \frac{x-\mu}{\sigma}$

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

5.2.6 배치 정규화

배치 정규화

- 공변량 시프트 현상을 누그러뜨리기 위해 식 (5.9)의 정규화를 모든 층에 적용하는 기법

$$x_i^{new} = \frac{x_i^{old} - \mu_i}{\sigma_i} \qquad (5.9)$$

- 정규화를 적용하는 곳이 중요
  - 식 (5.15)의 연산 과정 중 식 (5.9)를 어디에 적용하나?

$$z = \mathbf{w}^T\tilde{\mathbf{x}} + b$$
$$y = \tau(z) \qquad (5.15)$$

  - 입력 $\tilde{\mathbf{x}}$ 또는 중간 결과 z 중 어느 것에 적용? → z에 적용하는 것이 유리
- 훈련집합 전체 또는 미니배치 중 어느 것에 적용?
  - 미니배치에 적용하는 것이 유리 각 층마다!!

5.2.6 배치 정규화

- 정규화 변환을 수행하는 코드
  - 미니배치 $\mathbb{X}_B = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m\}$에 식 (5.15)를 적용하여 $\tilde{\mathbb{X}}_B = \{z_1, z_2, \cdots, z_m\}$를 얻은 후, $\tilde{\mathbb{X}}_B$를 가지고 코드 1을 수행
  - 노드마다 독립적으로 코드 1을 수행
  - γ와 β는 노드마다 고유한 매개변수로서 학습으로 알아냄

코드 1:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} z_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu_B)^2$$

$$\tilde{z}_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}, \qquad i = 1,2,\cdots,m$$

$$z_i' = \gamma\tilde{z}_i + \beta, \qquad i = 1,2,\cdots,m$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

5.2.6 배치 정규화

- 최적화를 마친 후 추가적인 후처리 작업 필요
  - 각 노드는 전체 훈련집합을 가지고 독립적으로 코드2를 수행

코드 2:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} z_i$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (z_i - \mu)^2$$

노드에 μ, σ^2, γ, β를 저장한다. // 예측 단계에서 식 (5.18)로 변환을 수행하기 위함

train

: mini batch 에 평균 분산 저장

=)test 할 때 그 값들을 평균 시킴 (moving average)

- 예측 단계 (inference)
  - 각 노드는 독립적으로 식 (5.18)을 적용(코드 1의 마지막 두 라인을 수행하는 셈)

$$z' = \frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}} z + \left(\beta - \frac{\gamma\mu}{\sqrt{\sigma^2 + \varepsilon}}\right) \quad (5.18)$$

test 할 때도

mini batch 가 아니라

train 에서 나왔던 전체

μ 와 σ 를 적용 (고정된값 적용)

5.2.6 배치 정규화

- CNN에서는
  - 노드 단위가 아니라 특징 맵 단위로 코드 1과 코드 2를 적용
  - Bias b 삭제 - β가 대신함
  - 각 채널 기준으로 γ와 β를 생성

    ex) mini batch –m, channel size –n, feature map size – p x q 일 때,

    n x m x p x q 개의 값에 대해 평균과 분산을 구함 -> 각 채널에 총 n개의 γ와 β 생성

- 배치 정규화의 긍정적 효과를 측정한 실험사례[Ioffe2015]
  - 가중치 초기화에 덜 민감함
  - 학습률을 크게 하여 수렴 속도 향상 가능
  - 시그모이드를 활성함수로 사용하는 깊은 신경망도 학습이 이루어짐
- 배치 정규화는 규제 효과를 제공
  - 드롭아웃이라는 규제 기법을 적용하지 않아도 높은 성능

5.2.6 배치 정규화 – Train & eval mode

```
...
total_batch = len(data_loader)
model.train()    # set the model to train mode (dropout=True)
for epoch in range(training_epochs):
...

...
# Test model and check accuracy
with torch.no_grad():
   model.eval()    # set the model to evaluation mode (dropout=False)
...
```

model.train() & model.eval()

- Sets the module in training/evaluation mode.
- This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

https://pytorch.org/docs/stable/nn.html?highlight=eval#torch.nn.Module.eval

5.2.6 배치 정규화 – Train & eval mode

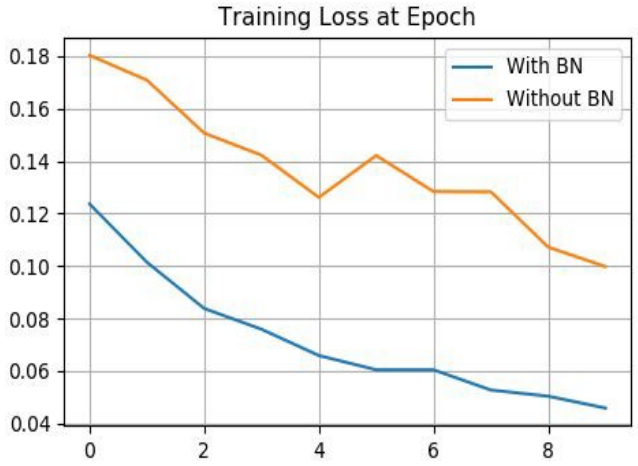
```
for epoch in range(training_epochs):  bn
   _model.train()    # set the model to
   train mode

   for X, Y in train_loader:
       # reshape input image into [batch_si
       ze by 784] # label is not one-hot en
       coded
       X = X.view(-1, 28 * 28).
       to(device) Y = Y.to(devi
       ce)

       bn_optimizer.zero
       _grad()  bn_predi
       ction = bn_model(
       X)
       bn_loss = criterion(bn_pred
       iction, Y) bn_loss.backward
       ()  bn_optimizer.step()

       nn_optimizer.zero
       _grad()  nn_predi
       ction = nn_model(
       X)
       nn_loss = criterion(nn_pred
       iction, Y) nn_loss.backward
       ()  nn_optimizer.step()
```

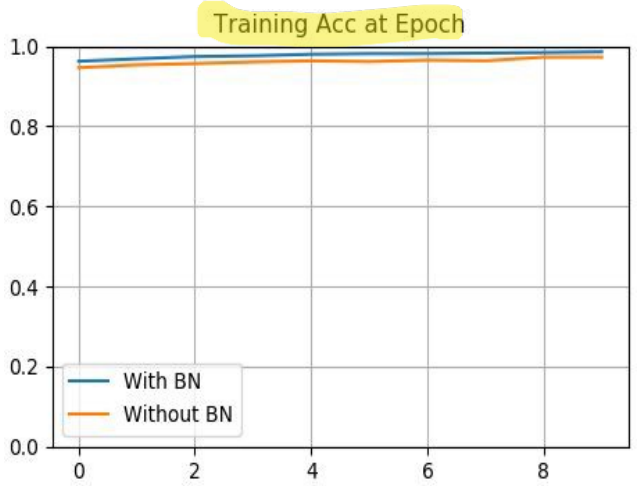
```
with torch.no_grad(): bn_model.eval()
   # set the model to evaluation mode

   # Test the model using train sets
    bn_loss, nn_loss, bn_acc, nn_acc = 0, 0, 0, 0
   for i, (X, Y) in enumerate(train_loader):
     X = X.view(-1, 28 * 28).to(device)
     Y = Y.to(device)

     bn_prediction = bn_model(X)
     bn_correct_prediction = torch.argmax(bn_prediction,
   1) == Y
     bn_loss += criterion(bn_prediction, Y)
     bn_acc += bn_correct_prediction.float().mean()

     nn_prediction = nn_model(X)
     nn_correct_prediction = torch.argmax(nn_prediction,
   1) == Y
     nn_loss += criterion(nn_prediction, Y)
     nn_acc += nn_correct_prediction.float().mean()
```

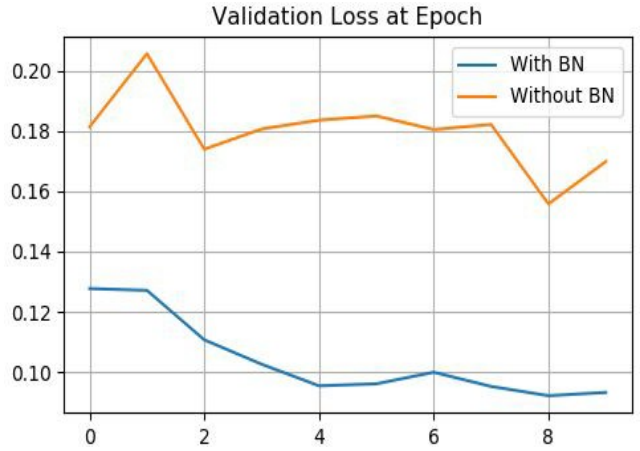
5.2.6 배치 정규화 – Layer configuration

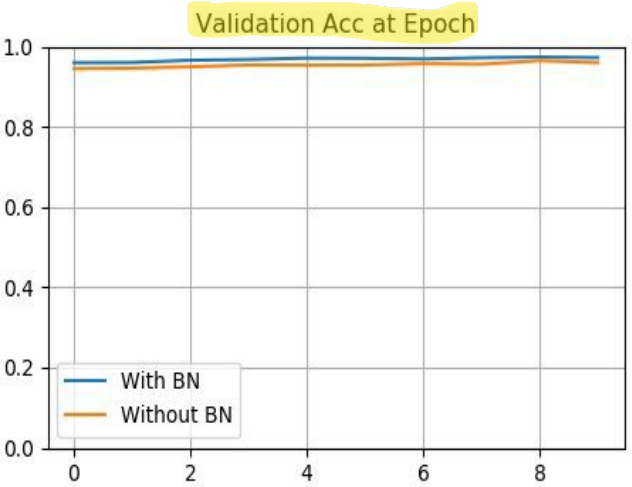
```
# nn layers
linear1 = torch.nn.Linear(784, 32, bias=True)
linear2 = torch.nn.Linear(32, 32, bias=True)
linear3 = torch.nn.Linear(32, 10, bias=True)
relu = torch.nn.ReLU()
bn1 = torch.nn.BatchNorm1d(32) bn2
= torch.nn.BatchNorm1d(32)

nn_linear1 = torch.nn.Linear(784, 32, bias=True)
nn_linear2 = torch.nn.Linear(32, 32, bias=True)
nn_linear3 = torch.nn.Linear(32, 10, bias=True)

# model
bn_model = torch.nn.Sequential(linear1, bn1, relu,
                        linear2, bn2, relu, linear
                        3).to(device)
nn_model = torch.nn.Sequential(nn_linear1, relu,
                         nn_linear2, relu,  nn_linear
                         3).to(device)
```

5.2.6 배치 정규화 – Results









5.3 규제의 필요성과 원리

- 5.3.1 과잉적합에 빠지는 이유와 과잉적합을 피하는 전략
- 5.3.2 규제의 정의

- 규제가 중요하기 때문에 1장에서 미리 소개한 내용
  - 1.5절의 과소적합과 과잉적합, 바이어스와 분산([그림 1-13], [그림 1-14])
  - 1.6절의 데이터 확대와 가중치 감쇠

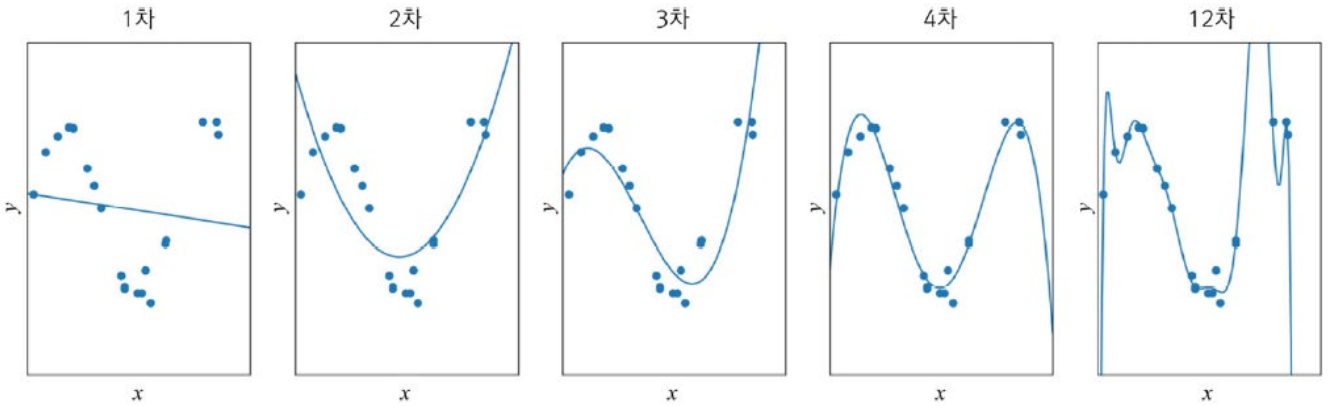


그림 1-13 과소적합과 과잉적합 현상

5.3.1 과잉적합에 빠지는 이유와 과잉적합을 피하는 전략

- 학습 모델의 용량에 따른 일반화 능력

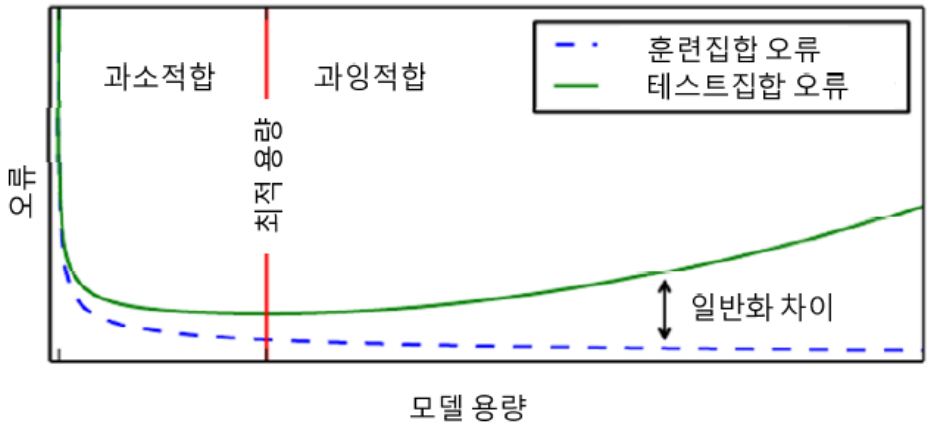


그림 5-18 학습 모델의 용량과 일반화 능력의 관계

- 대부분 가지고 있는 데이터에 비해 훨씬 큰 용량의 모델을 사용
  - 예) VGGNet은 분류층에 1억 2천 1백만 개의 매개변수
  - 훈련집합을 단순히 '암기'하는 과잉적합에 주의를 기울여야 함
- 현대 기계 학습의 전략
  - 충분히 큰 용량의 모델을 설계한 다음, 학습 과정에서 여러 규제 기법을 적용

5.3.2 규제의 정의

- 규제는 오래 전부터 수학과 통계학에서 연구해온 주제
  - 모델 용량에 비해 데이터가 부족한 경우의 불량 문제를ill-posed problem 푸는 데 사용
  - 적절한 가정을 투입하여 문제를 품 → 입력과 출력 사이의 매핑은 매끄럽다는 사전 지식

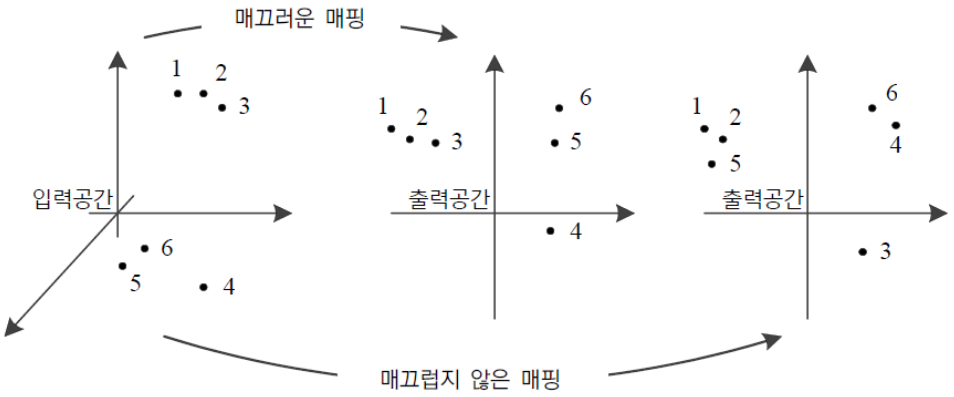


그림 5-20 사전 지식으로서 매끄러움의 특성

  - 티호노프의 규제 기법는 매끄러움 가정에 기반을 둔 식 (5.19)를 사용

$$\underbrace{J_{regularized}(\Theta)}_{\text{규제를 적용한 목적함수}} = \underbrace{J(\Theta)}_{\text{목적함수}} + \underbrace{\lambda R(\Theta)}_{\text{규제 항}} \qquad (5.19)$$

5.3.2 규제의 정의

- 현대 기계 학습도 매끄러움 가정을 널리 사용함
  - 5.4.1절의 가중치 감쇠 기법
    - 모델의 구조적 용량을 충분히 크게 하고, '수치적 용량'을 제한하는 규제 기법
  - 6장의 비지도 학습 등

- 『Deep Learning』 책의 정의

"...any modification we make to a learning algorithm that is intended to reduce its generalization error ... 일반화 오류를 줄이려는 의도를 가지고 학습 알고리즘을 수정하는 방법 모두"