

CHAPTER 6

학습 관련 기술들

6. 주제 – 신경망 학습의 핵심 개념

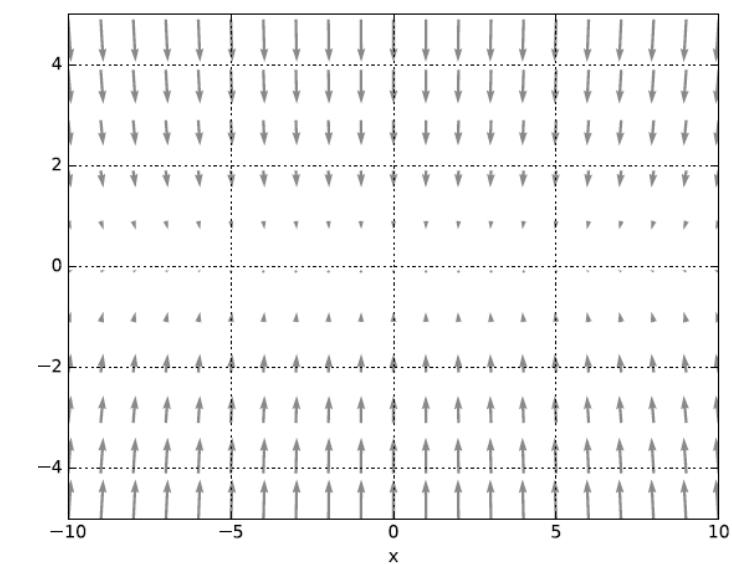
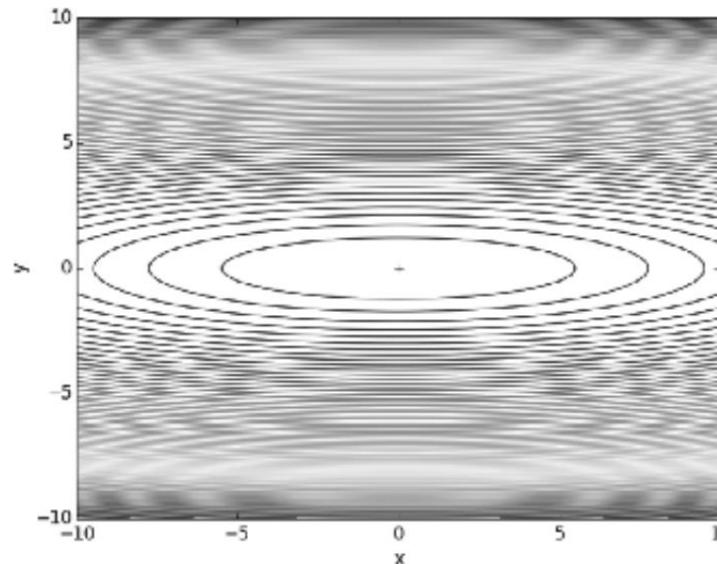
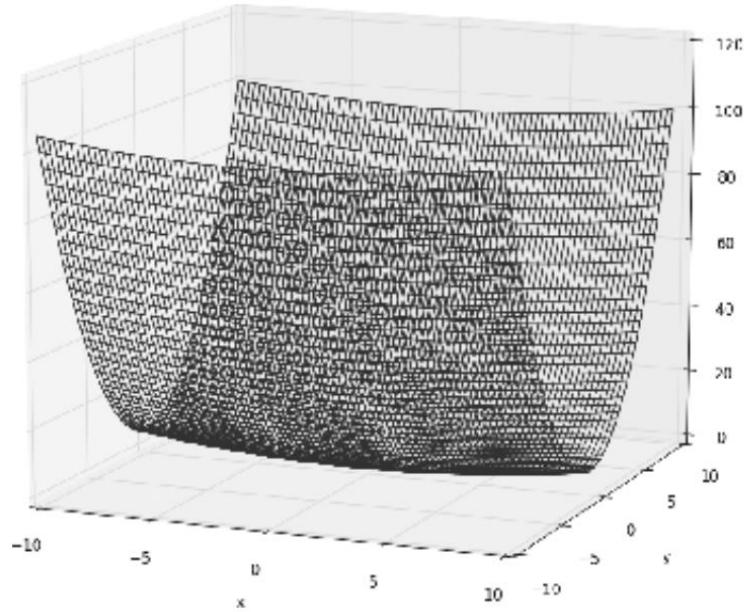
- 매개 변수 갱신
- 가중치 초기값
- 배치 정규화
- 바른학습을 위해
- 적절한 하이퍼 파라미터 값 찾기.

매개변수 갱신

- 문제: 매개 변수의 수와 범위가 넓어 최적화 하기 어렵다.
- 해결책 : 확률적 경사 강하법 (매개변수의 기울기를 이용해 최적화)
- 그 외 방법은? Momentum, AdaGrad, Adam 등.

확률적 경사 강하법의 단점

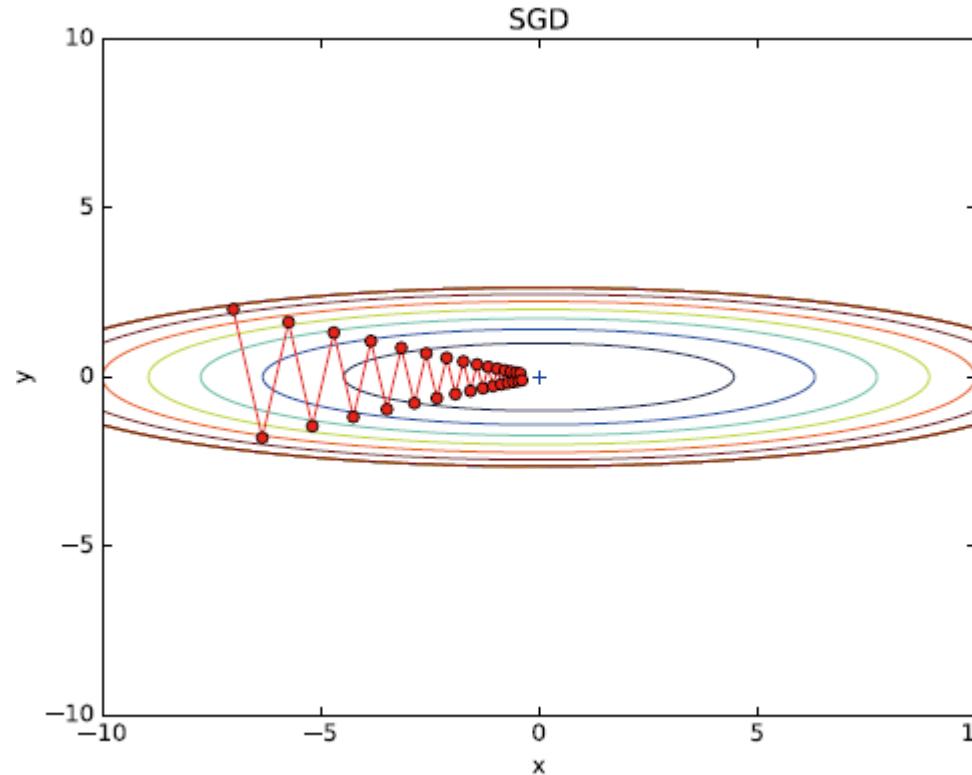
$f(x,y) = \frac{1}{20}x^2 + y^2$ 의 그래프와 등고선 그리고 기울기.



- $(x,y) = (0,0)$ 이 최소값이지만, 기울기 대부분이 $(0,0)$ 을 가르키지 않는다.

확률적 경사 강하법의 단점

- 확률적 경사 경하법을 적용하면 아래와 같이 수행된다.



결국 비등방성 함수 형태에서는 탐색 경로가 비효율적.
지그재그로 탐색하는 원인은 기울어진 방향이 최소값을 가르키지 않아서..

Momentum

- Momentum은 운동량을 뜻하는 단어. 공이 그릇의 바닥을 구르는 듯한 움직임을 보여준다.

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

[SGD]

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}$$

[Momentum]

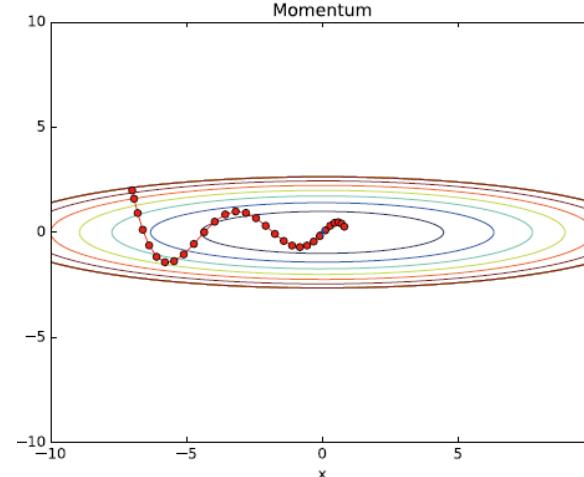
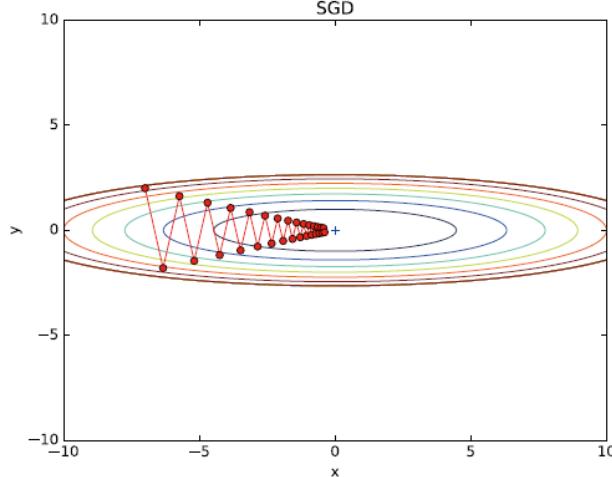
```
def __init__(self, lr=0.01, momentum=0.9):
    self.lr = lr
    self.momentum = momentum
    self.v = None

def update(self, params, grads):
    if self.v is None:
        self.v = {}
    for key, val in params.items():
        self.v[key] = np.zeros_like(val)

    for key in params.keys():
        self.v[key] = self.momentum * self.v[key] - self.lr * grads[key]
        params[key] += self.v[key]
```

모멘텀에 의한 최적화 갱신 경로

- SGD와 비교하면 지그재그 정도가 덜한 것을 알 수 있다. X축의 힘의 작지만 방향은 변하지 않아 누적하여 일정하게 가속하기 때문..



```
def update(self, params, grads):
    for key in params.keys():
        params[key] -= self.lr * grads[key]
```

```
def update(self, params, grads):
    if self.v is None:
        self.v = {}
    for key, val in params.items():
        self.v[key] = np.zeros_like(val)

    for key in params.keys():
        self.v[key] = self.momentum*self.v[key] - self.lr*grads[key]
        params[key] += self.v[key]
```

AdaGrad

- 처음에는 크게 학습하다가 조금씩 작게 학습한다는 식의 학습률 조정 알고리즘이 AdaGrad.

$$\mathbf{h} \leftarrow \mathbf{h} + \frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}}$$

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial L}{\partial \mathbf{W}}$$

[AdaGrad]

```
"""AdaGrad"""

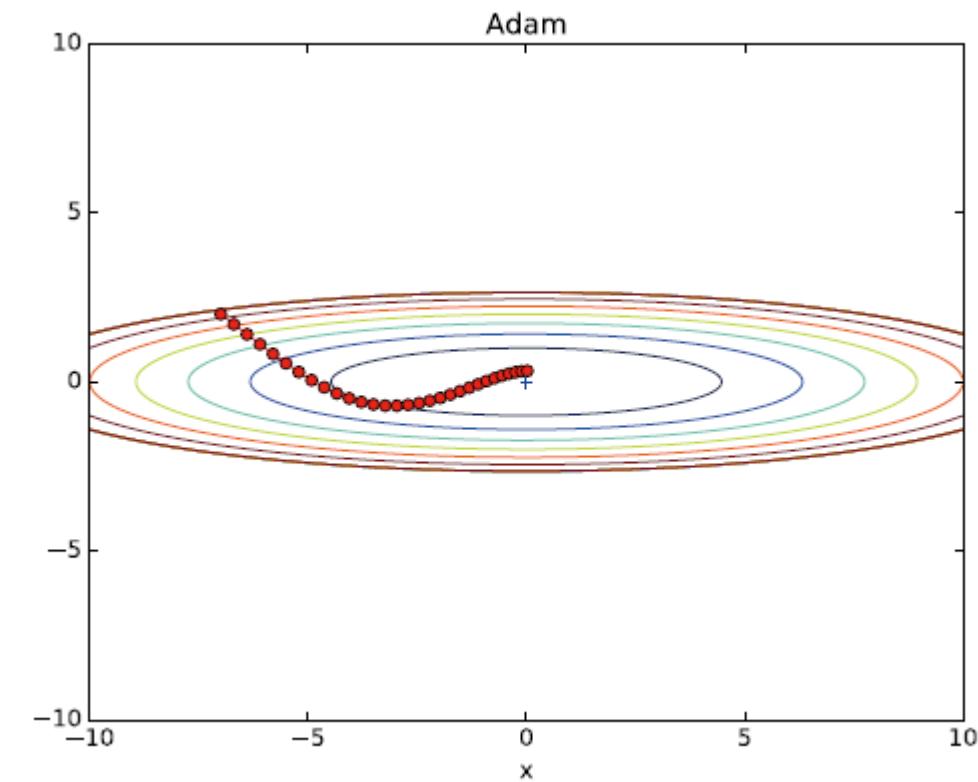
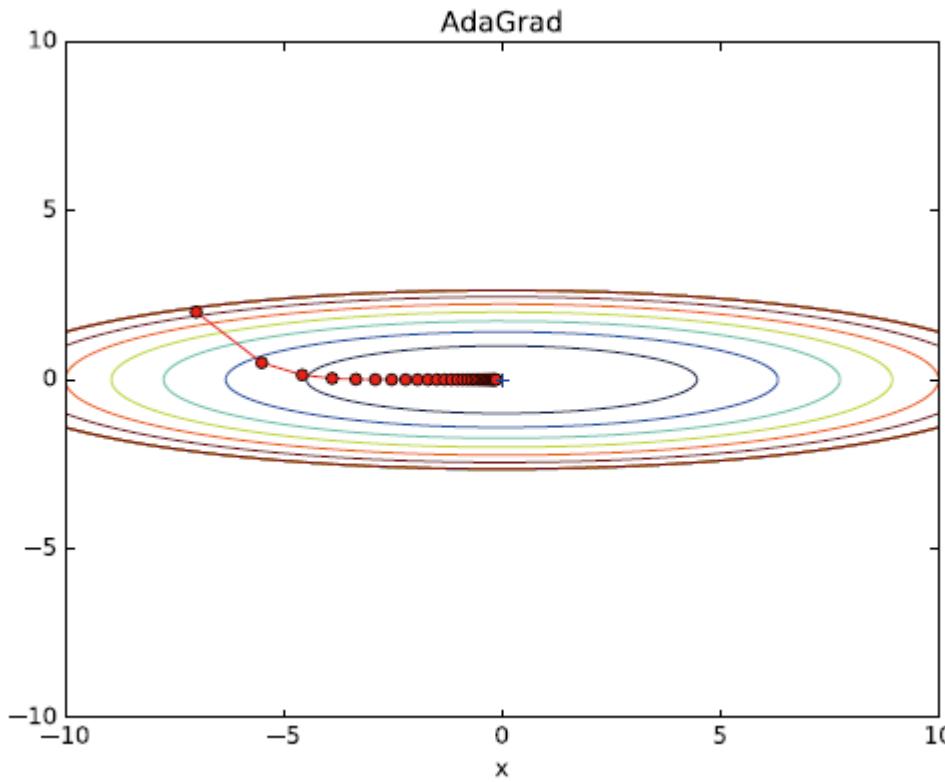
def __init__(self, lr=0.01):
    self.lr = lr
    self.h = None

def update(self, params, grads):
    if self.h is None:
        self.h = {}
    for key, val in params.items():
        self.h[key] = np.zeros_like(val)

    for key in params.keys():
        self.h[key] += grads[key] * grads[key]
        params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```

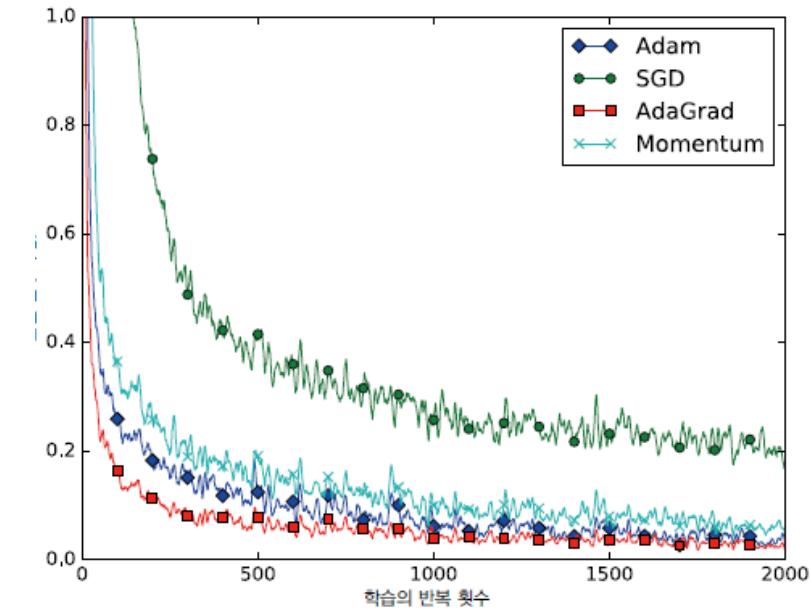
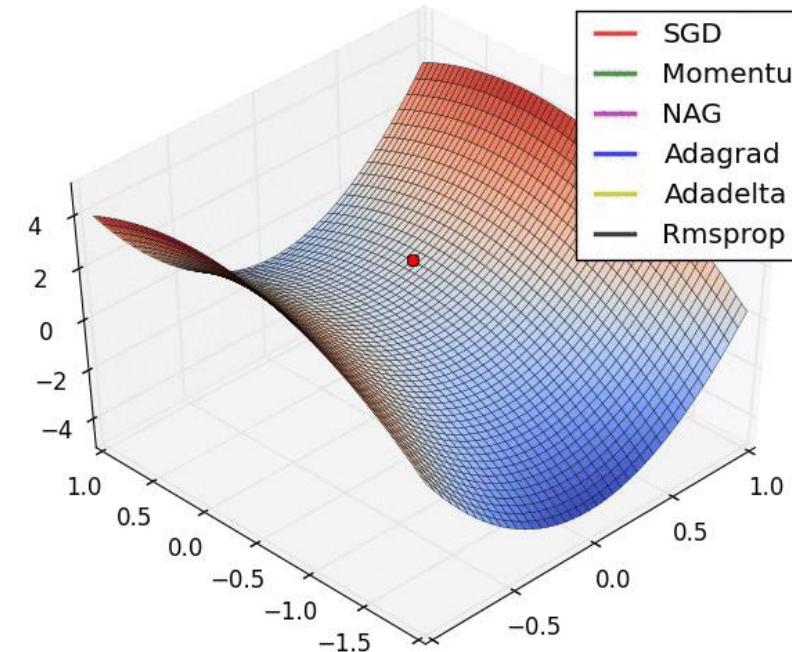
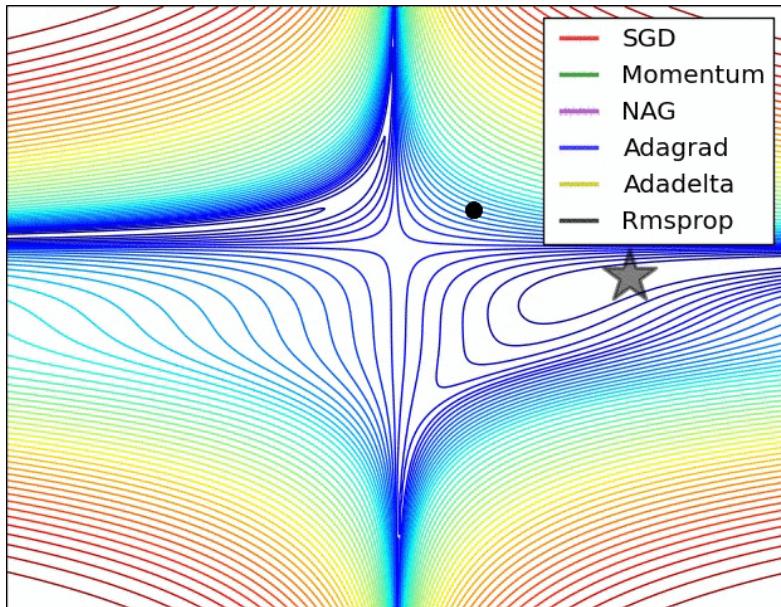
AdaGrad와 Adam에 의한 최적화 갱신 경로

- **Adam** = Momentum + AdaGrad



확률적 최적화(stochastic optimization) 방법 비교.

<http://aikorea.org/cs231n/neural-networks-3/>



가중치의 초기값.

- 가중치 초기값을 정하는 방법은 올바른 학습을 하는 데 매우 중요하다.
- 신경망에서는 가중치가 클수록 오버피팅이 일어날 가능성이 높으므로, 가중치 값을 작게 만들어 사용한다.
- 단 가중치의 초기값을 0으로 만들거나 같을 경우, 신경망의 각 층을 진행해도 가중치 값이 여전히 같은 값을 가지게 된다. 따라서 가중치가 고르게 되어버리는 상황을 막으려면 초기값을 작고, 무작위하게 설정해야 된다.

은닉층의 활성화값 분포

그림 6-10 기증치를 표준편차가 1인 정규분포로 초기화할 때의 각 층의 활성화값 분포

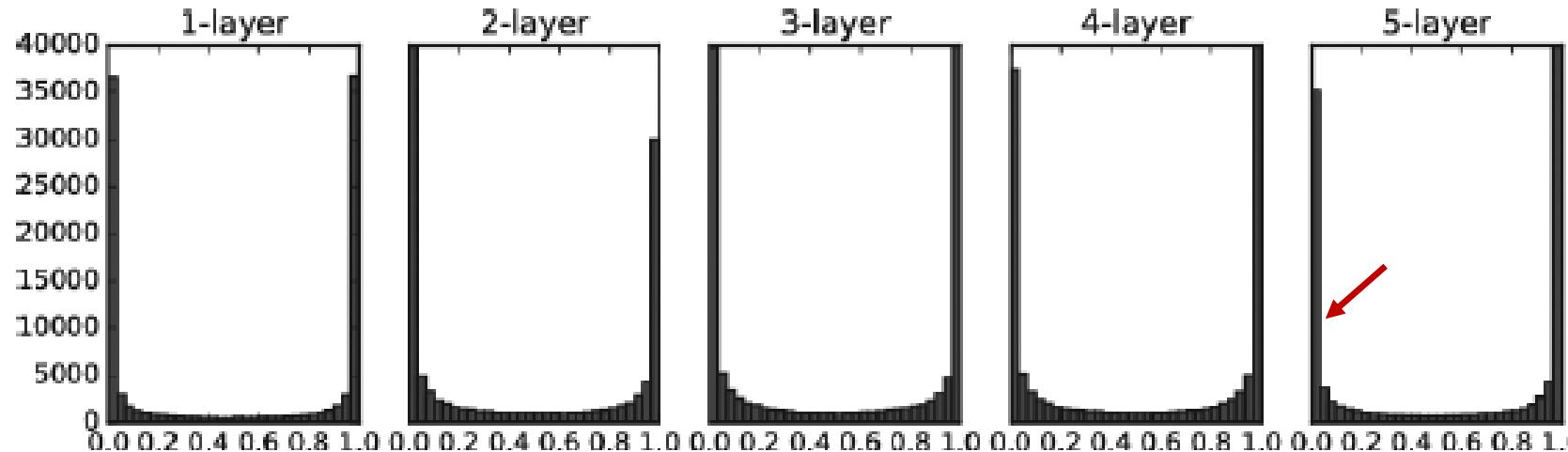
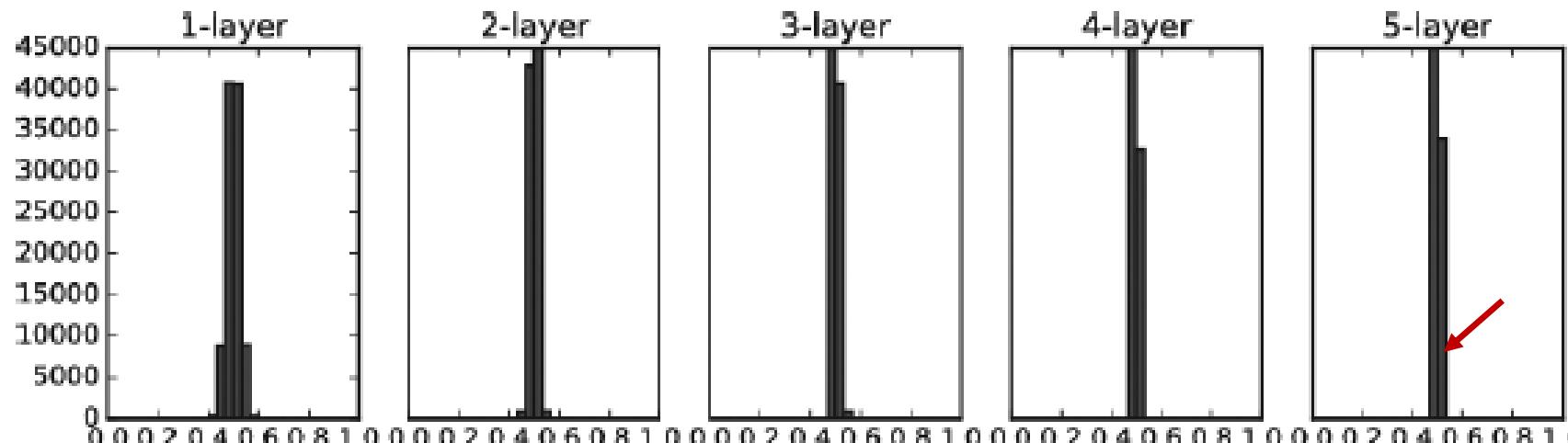


그림 6-11 기증치를 표준편차가 0.01인 정규분포로 초기화할 때의 각 층의 활성화값 분포



은닉층의 활성화값 분포

WARNING 각 층의 활성화값은 적당히 고루 **분포되어야 합니다**. 층과 층 사이에 적당하게 다양한 데이터가 흐르게 해야 신경망 학습이 효율적으로 이뤄지기 때문입니다. 반대로 치우친 데이터가 흐르면 기울기 소실이나 표현력 제한 문제에 빠져서 학습이 잘 이뤄지지 않는 경우가 생깁니다.

- 결국 초기값을 작고, 고르게 분포시켜야 하는 방법을 찾아 사용해야 한다.

Xavier 초기값, He's 초기값

- **Xavier initialization:** X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in International conference on artificial intelligence and statistics, 2010
- **He’s initialization:** K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” 2015

```
# Xavier initialization  
# Glorot et al. 2010
```

```
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)
```

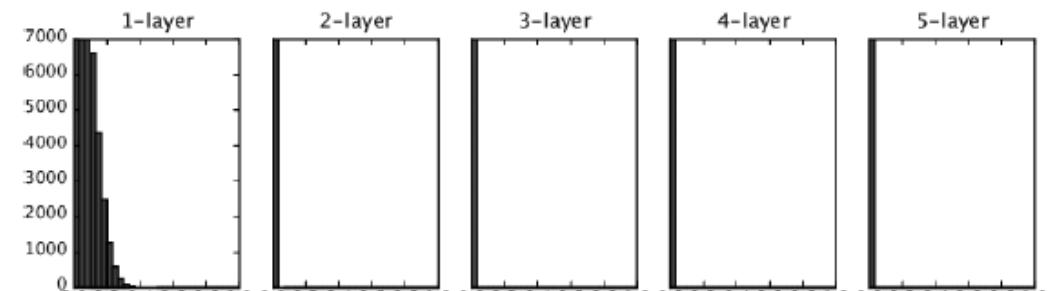
```
# He et al. 2015
```

```
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

Xavier 초기값, He's 초기값에 따른 활성화 값 분포 변화

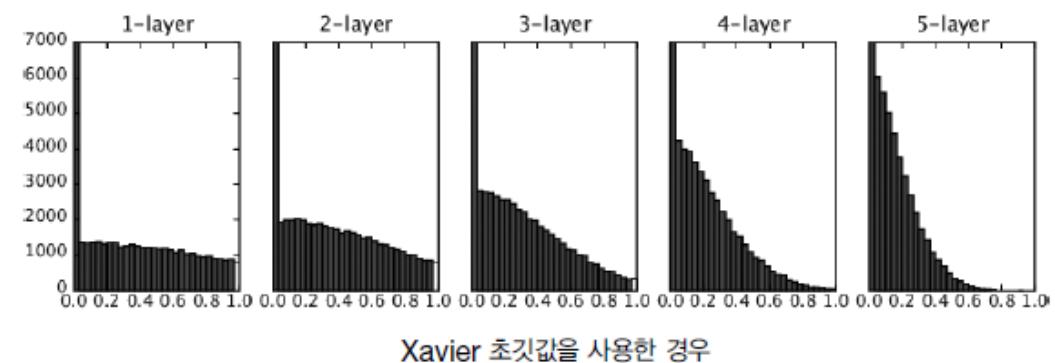
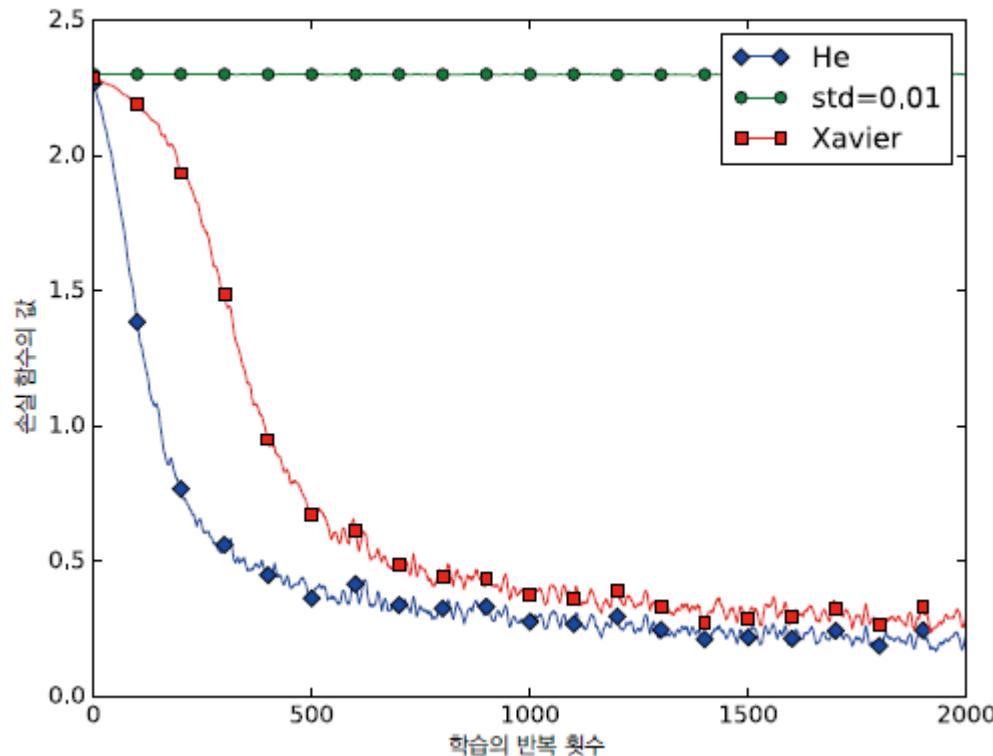
활성화 함수로 ReLU를 사용할 때는 He's 초기값을, Sigmoid나 Tanh 등의 S자 모양 곡선일 때는 Xavier 초기값을 사용한다.

그림 6-14 활성화 함수로 ReLU를 사용한 경우의 가중치 초기값에 따른 활성화값 분포 변화

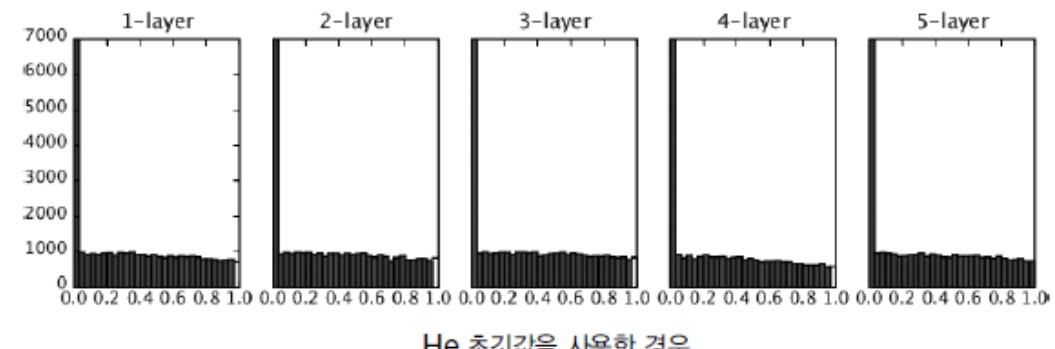


표준편차가 0.01인 정규분포를 가중치 초기값으로 사용한 경우

그림 6-15 MNIST 데이터셋으로 살펴본 '가중치의 초기값'에 따른 비교



Xavier 초기값을 사용한 경우



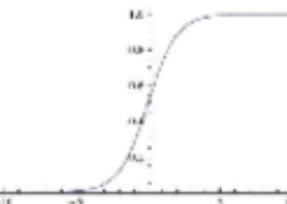
He 초기값을 사용한 경우

Xavier 초기값, He's 초기값에 따른 활성화 값 분포 변화

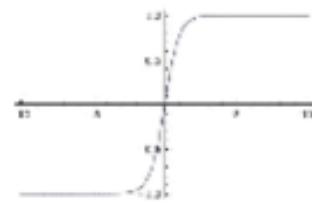
Activation Functions

Sigmoid

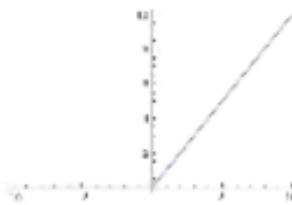
$$\sigma(x) = 1/(1 + e^{-x})$$



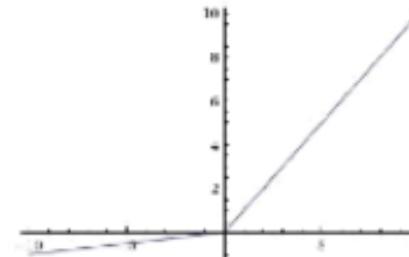
tanh tanh(x)



ReLU max(0,x)



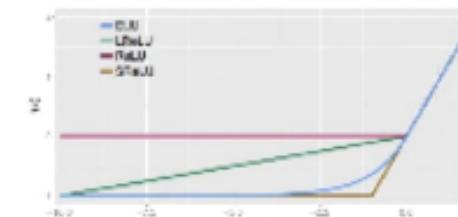
Leaky ReLU max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

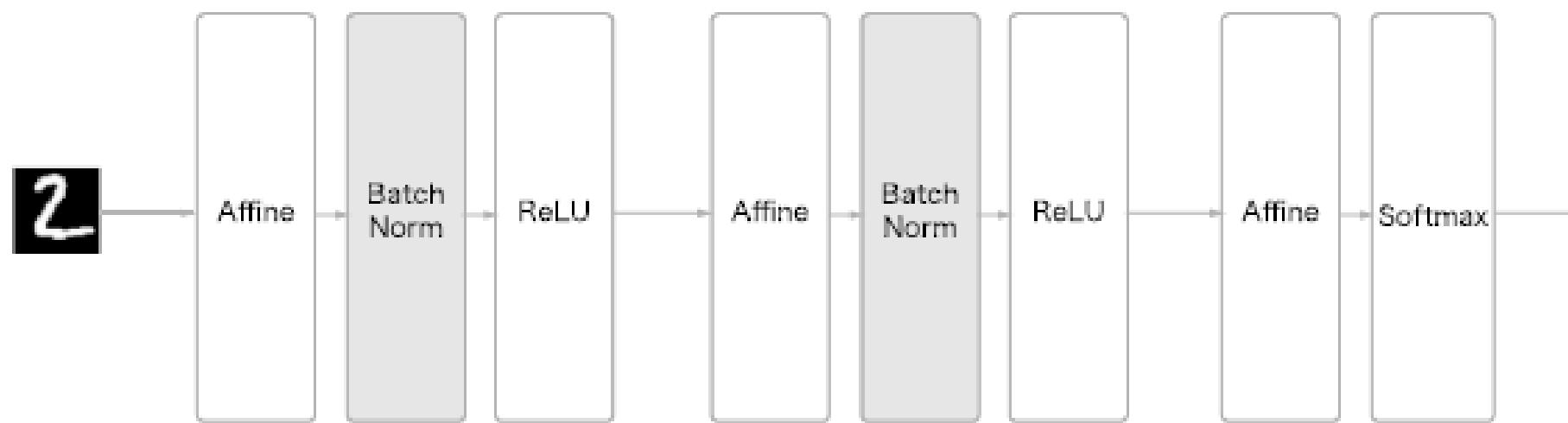


maxout	ReLU	VLReLU	tanh	Sigmoid
93.94	92.11	92.97	89.28	n/c
93.78	91.74	92.40	89.48	n/c
-	91.93	93.09	-	n/c
91.75	90.63	92.27	89.82	n/c
n/c†	90.91	92.43	89.54	n/c

배치 정규화

- 가중치의 초기값을 적절히 설정하면 각 층의 활성화값의 분포가 적당히 퍼지면서 학습이 원활하게 수행된다는 것을 알 수 있다.
- 그렇다면 각 층이 활성화를 적당히 퍼트리도록 강제하는 방법이 ‘배치 정규화’..

그림 6-16 배치 정규화를 사용한 신경망의 예



배치 정규화

- 배치 정규화는 데이터 분포가 평균이 0, 분산이 1이 되도록 정규화 시킨 후, 활성화 함수의 nonlinearity를 위해 γ (scale factor)와 β (shift factor) 더해주고 back-prop 과정에서 같이 train 시켜준다.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

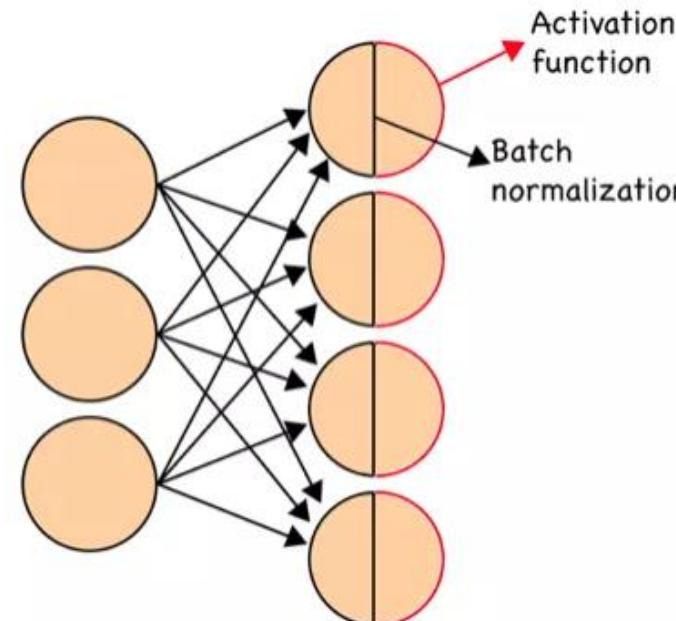
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$



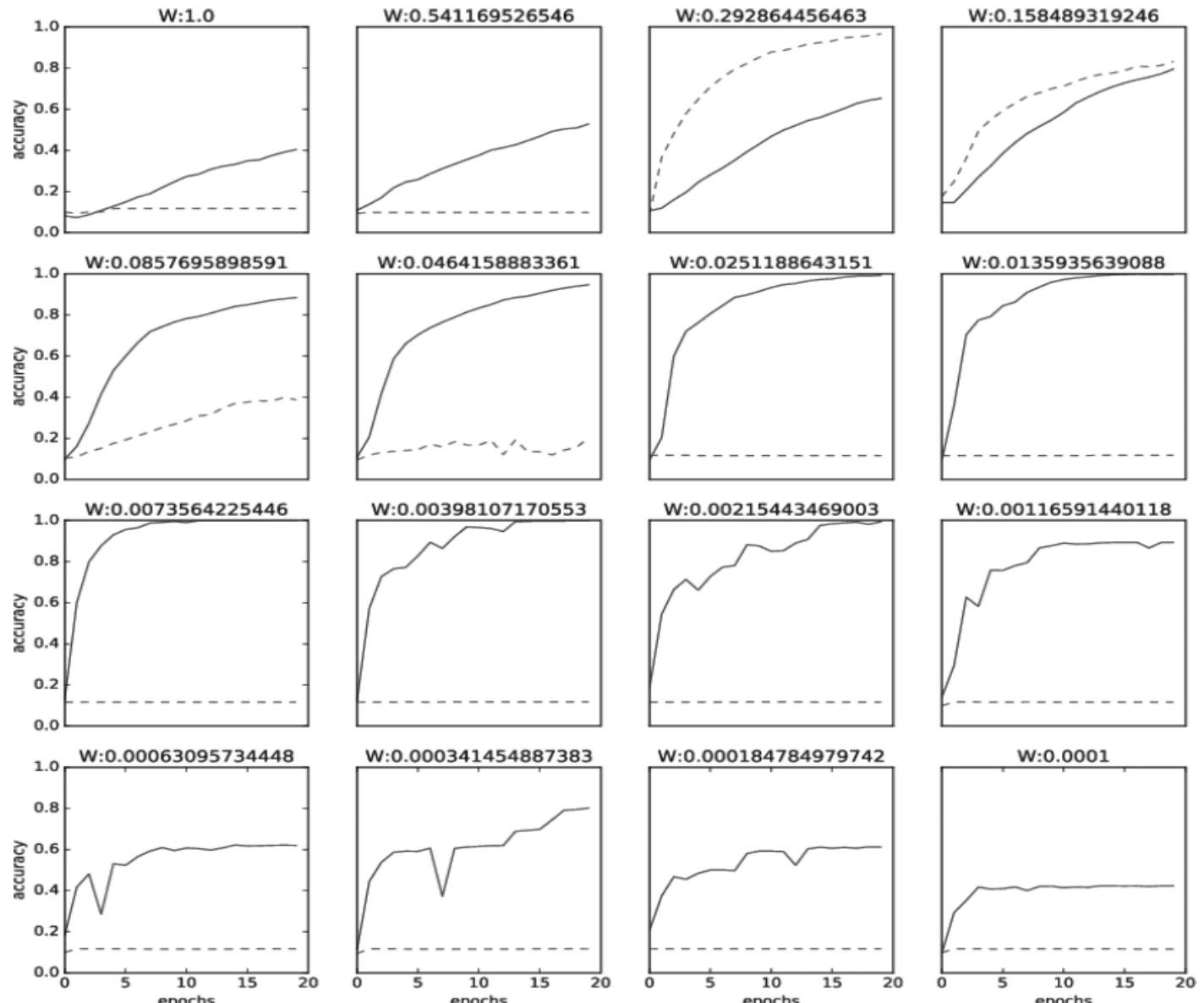
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$$

배치 정규화 효과

- 학습 속도 개선
- 초깃값에 크게 의존하지 않는다
- 오버피팅 억제(드롭아웃 등의 필요성 감소).

그림 6-19 실선이 배치 정규화를 사용한 경우. 점선이 사용하지 않은 경우 : 가중치 초기값의 표준편차는 각 그래프 위에 표기



바른 학습을 위해

- 훈련 데이터에는 포함되지 않는, 아직 보지 못한 데이터가 주어져도 바르게 식별해내는 모델이 바람직. 결국 오버피팅을 억제하는 기술 필요.
- 주로 매개변수가 많고 표현력이 높은 모델, 훈련데이터가 적을 때 발생.

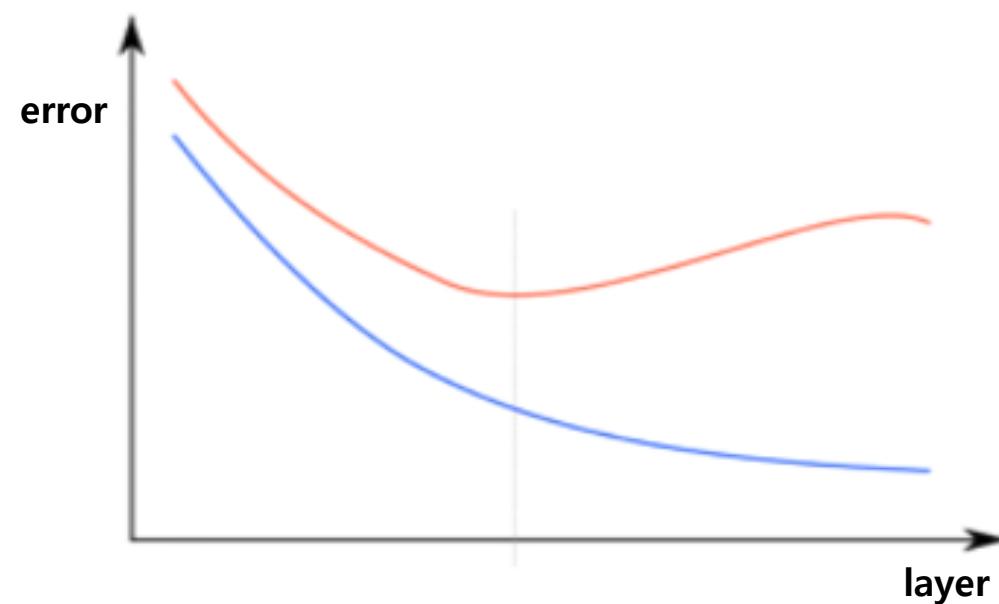
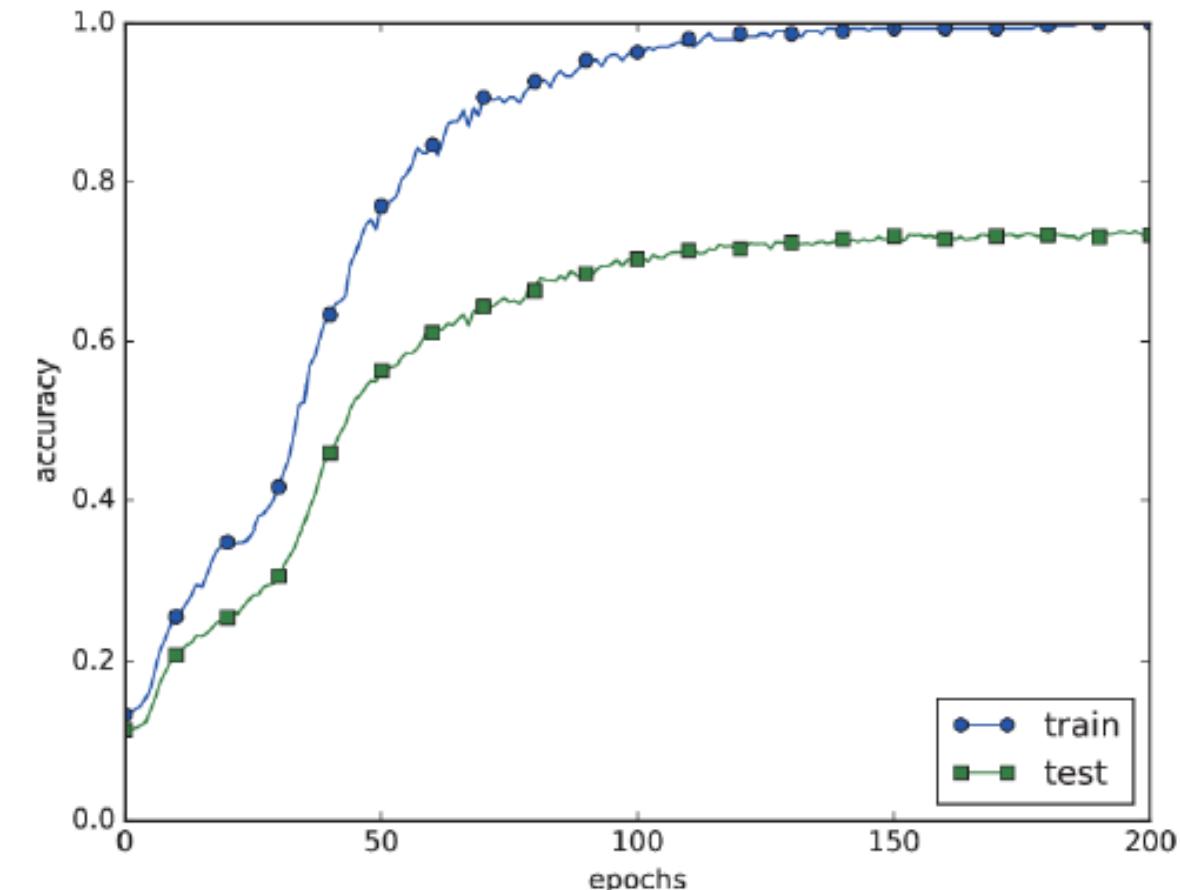


그림 6-20 훈련 데이터(train)와 시험 데이터(test)의 에폭별 정확도 추이



Solutions for overfitting

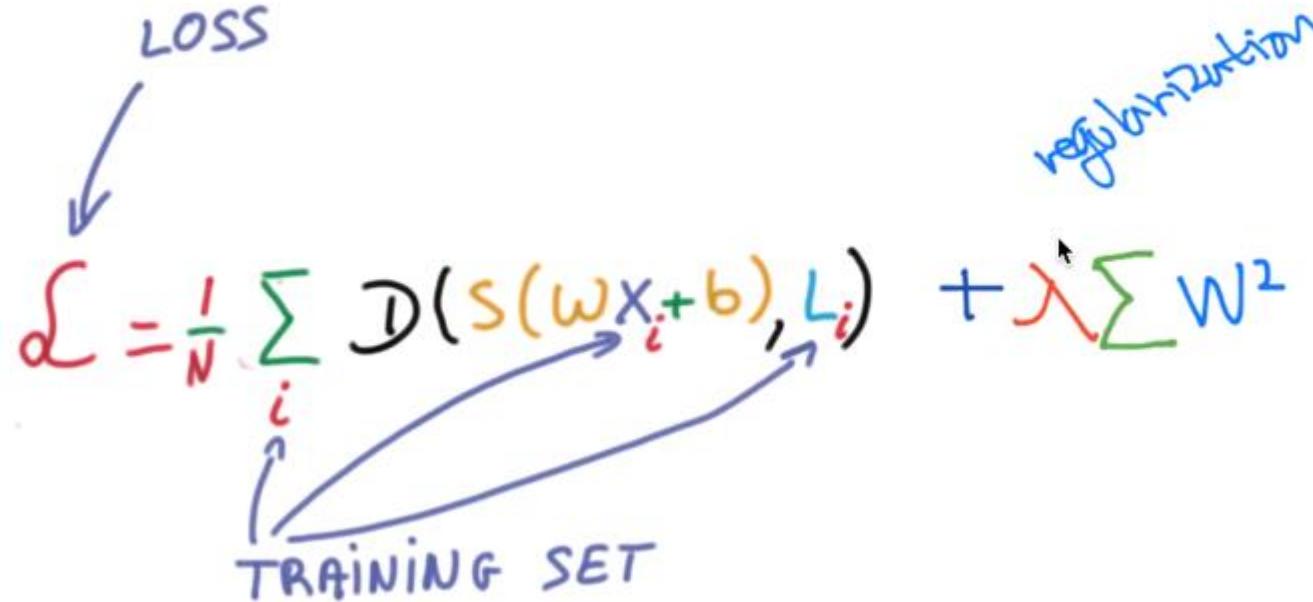
- More training data!
- Reduce the number of features
- **Regularization**
- Let's not have too big numbers in the weight

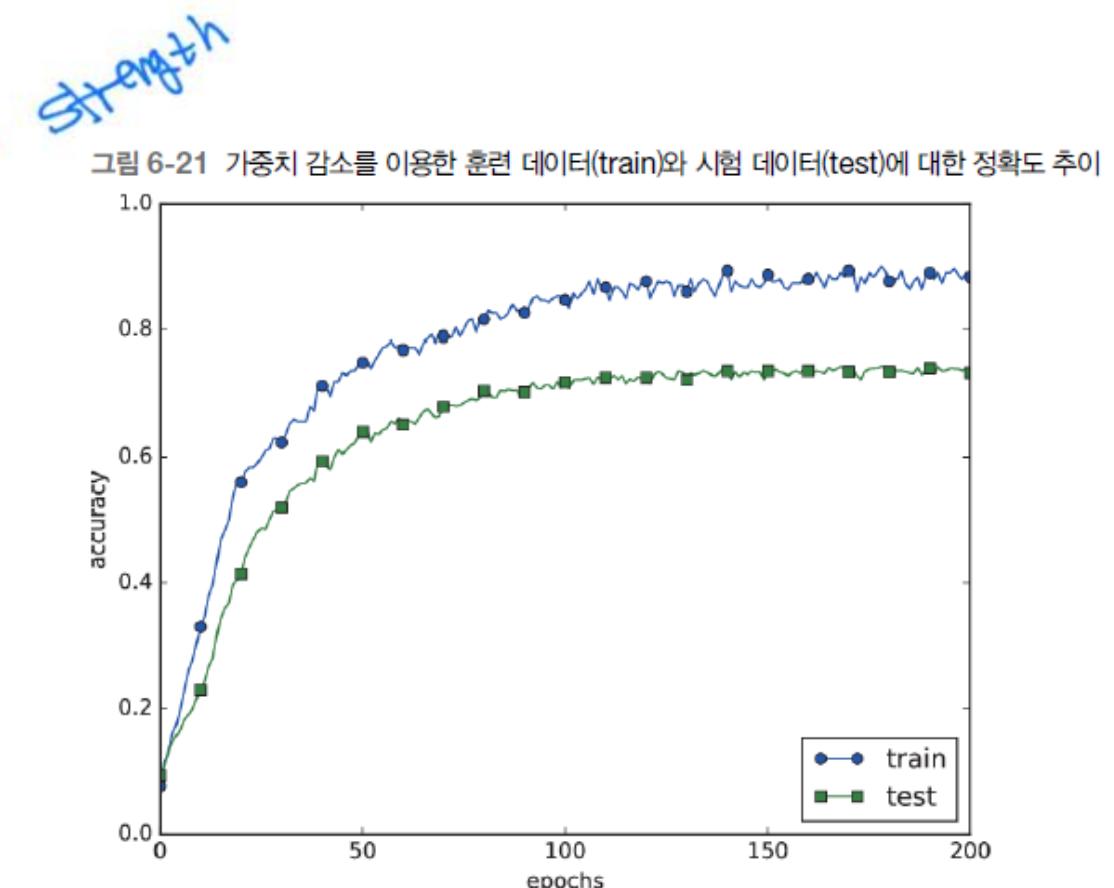
LOSS

$$\mathcal{L} = \frac{1}{N} \sum_i \mathcal{D}(s(\omega x_i + b), L_i) + \lambda \sum w^2$$

regularization

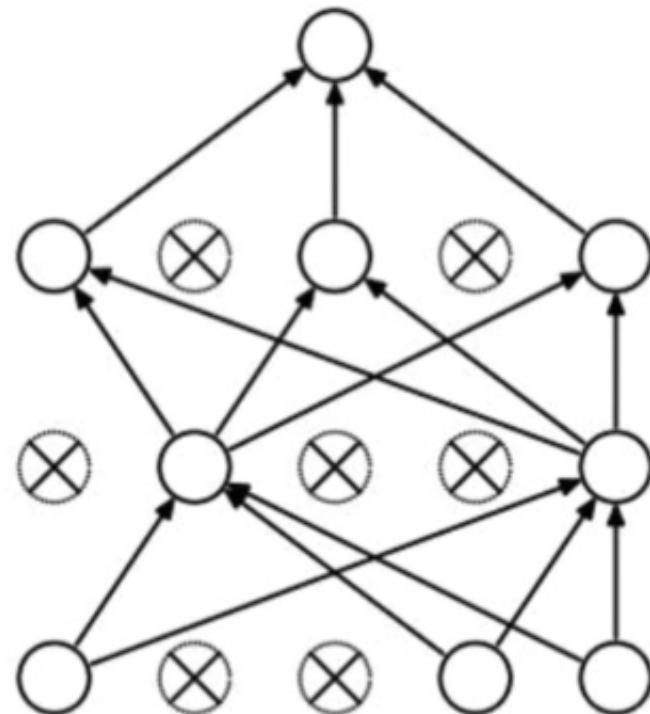
TRAINING SET





드롭아웃

- “Randomly set some neurons to zero in the forward pass”

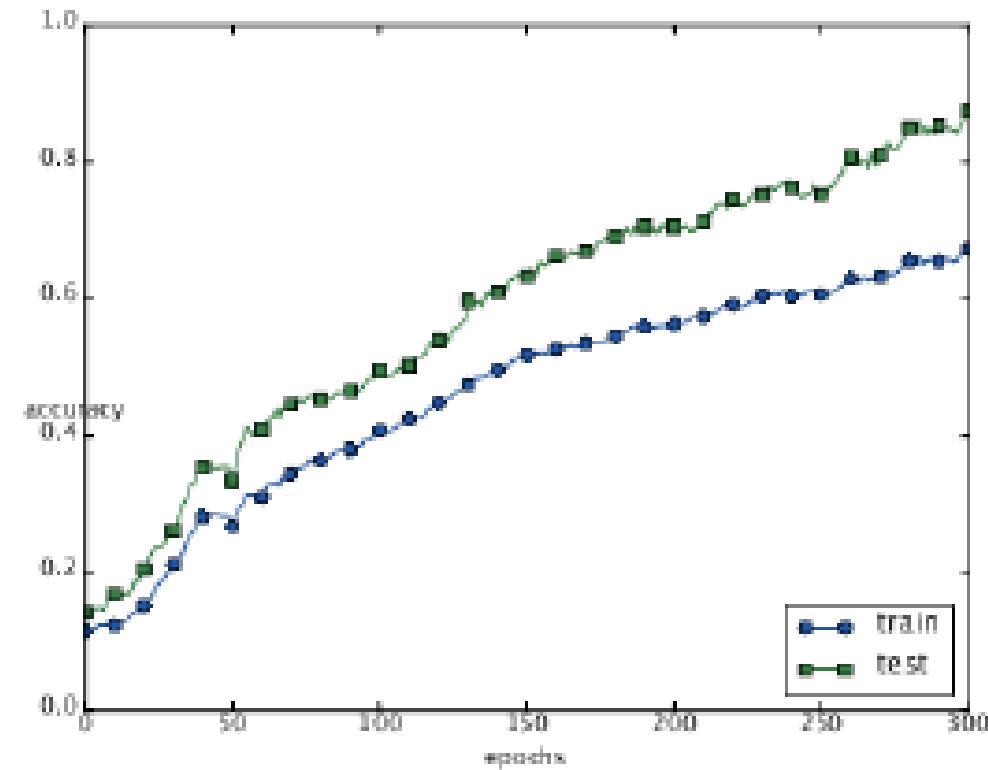
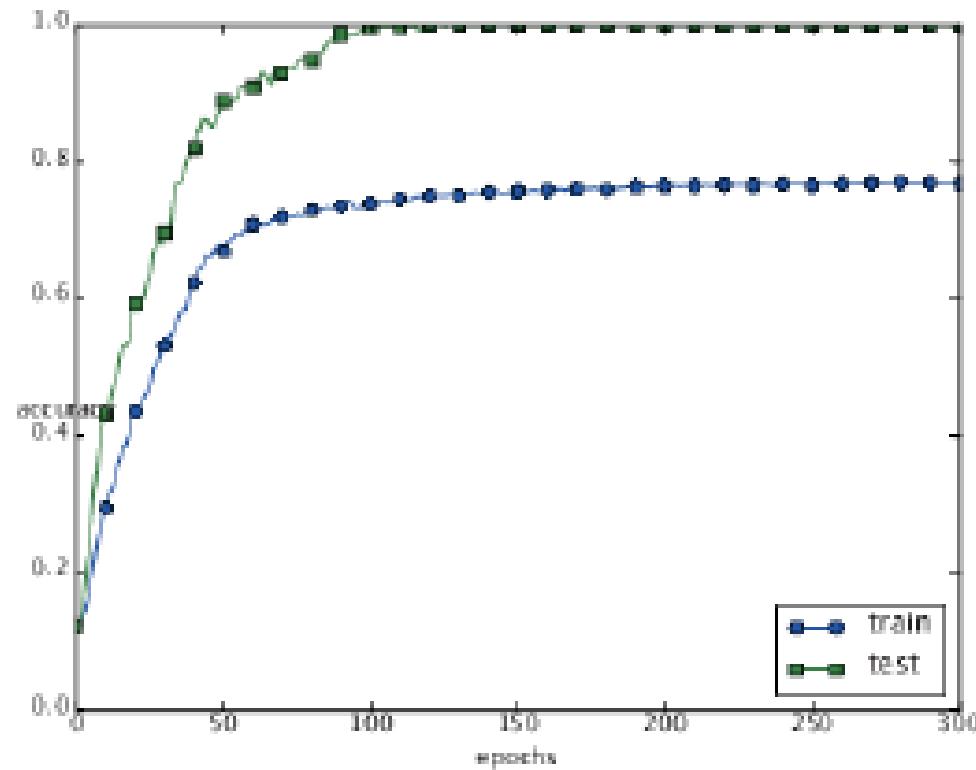


Forces the network to have a redundant representation.



드롭아웃

그림 6-23 왼쪽은 드롭아웃 없이, 오른쪽은 드롭아웃을 적용한 결과 ($\text{dropout_ratio} = 0.15$)



적절한 하이퍼 파라미터 값 찾기.

- 신경망의 하이퍼파라미터 (각 층의 뉴런 수, 배치 크기, 매개변수 갱신 시의 학습률과 가중치 감소률 등)
- 하이퍼 파라미터의 성능을 평가할 때는 검증데이터를 추가하여 사용한다.
단. 신경망에서만 적용.

NOTE 훈련 데이터는 매개변수(가중치와 편향)의 학습에 이용하고, 검증 데이터는 하이퍼파라미터의 성능을 평가하는 데 이용합니다. 시험 데이터는 범용 성능을 확인하기 위해서 마지막에 (이상적으로는 한 번만) 이용합니다.

- 훈련 데이터 : 매개변수 학습
- 검증 데이터 : 하이퍼파라미터 성능 평가
- 시험 데이터 : 신경망의 범용 성능 평가

적절한 하이퍼 파라미터 값 찾기.

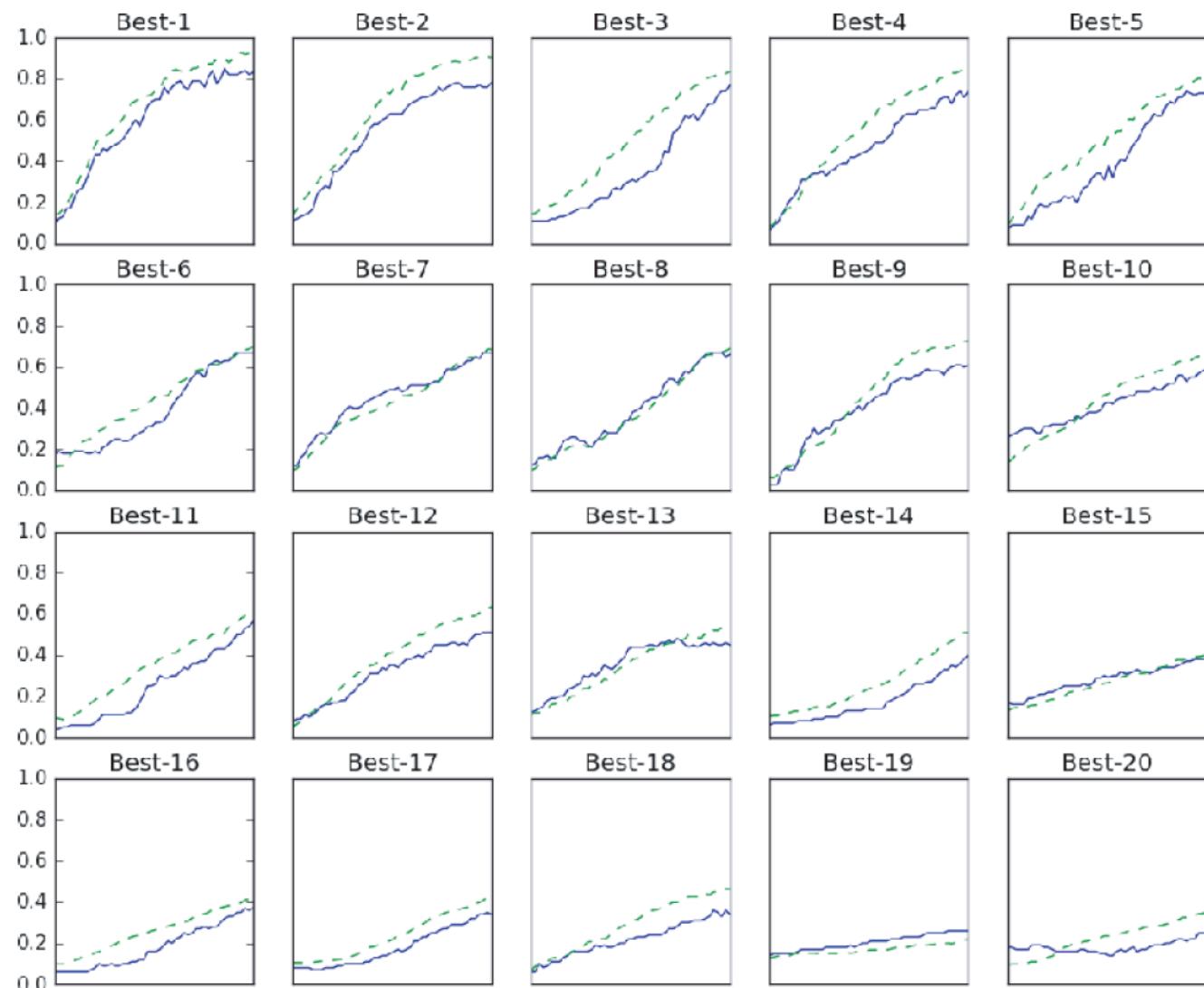
- 최적화할 때의 핵심은 하이퍼파라미터의 ‘최적 값’이 존재하는 범위를 조금씩 줄여간다는 것
 - 0단계
하이퍼파라미터 값의 범위를 설정한다.
 - 1단계
설정된 범위에서 하이퍼파라미터의 값을 무작위로 추출한다.
 - 2단계
1단계에서 샘플링한 하이퍼파라미터 값을 사용하여 학습하고, 검증 데이터로 정확도를 평가한다(단, 에폭은 작게 설정한다).
 - 3단계
1단계와 2단계를 특정 횟수(100회 등) 반복하며, 그 정확도의 결과를 보고 하이퍼파라미터의 범위를 좁힌다.

하이퍼파라미터 최적화 구현하기

```
weight_decay = 10 **  
np.random.uniform(-8, -4)  
lr = 10 ** np.random.uniform(-6, -2)
```

```
Best-1 (val acc:0.83) | lr:0.0092, weight decay:3.86e-07  
Best-2 (val acc:0.78) | lr:0.00956, weight decay:6.04e-07  
Best-3 (val acc:0.77) | lr:0.00571, weight decay:1.27e-06  
Best-4 (val acc:0.74) | lr:0.00626, weight decay:1.43e-05  
Best-5 (val acc:0.73) | lr:0.0052, weight decay:8.97e-06
```

그림 6-24 실선은 검증 데이터에 대한 정확도, 점선은 훈련 데이터에 대한 정확도



이번 장에서 배운 것

- 매개변수 갱신 방법에는 확률적 경사 하강법(SGD) 외에도 모멘텀, AdaGrad, Adam 등이 있다.
- 가중치 초기값을 정하는 방법은 올바른 학습을 하는 데 매우 중요하다.
- 가중치의 초기값으로는 'Xavier 초기값'과 'He 초기값'이 효과적이다.
- 배치 정규화를 이용하면 학습을 빠르게 진행할 수 있으며, 초기값에 영향을 덜 받게 된다.
- 오버피팅을 억제하는 정규화 기술로는 가중치 감소와 드롭아웃이 있다.
- 하이퍼파라미터 값 탐색은 최적 값이 존재할 법한 범위를 점차 좁히면서 하는 것이 효과적이다.