Neural Network Learning Concepts

Computational Linguistics @ Seoul National University

DL from Scratch

By Hyopil Shin

Core Concepts for NN

- Optimization
  - Stochastic Gradient Descent
  - Momentum
  - AdaGrad
  - Adam
- Weight Initial Value
  - Weight decay
- Batch Nomalization
- Overfitting
  - Weight decay
  - Drop out

Parameter Optimization

- Optimization – 최적의 매개변수를 찾는 것
  - Statistical Gradient Descent

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

```
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr

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

```
network = TwoLayerNet(...)
optimizer = SGD()

for i in range(10000):
    ...
    x_batch, t_batch = get_mini_batch(...) # 미니배치
    grads = network.gradient(x_batch, t_batch)
    params = network.params
    optimizer.update(params, grads)
    ...
```

Parameter Optimization- SGD

- SGD의 단점

$$f(x,y) = \frac{1}{20}x^2 + y^2$$

그림 6-1 $f(x,y) = \frac{1}{20}x^2 + y^2$의 그래프(왼쪽)와 그 등고선(오른쪽)

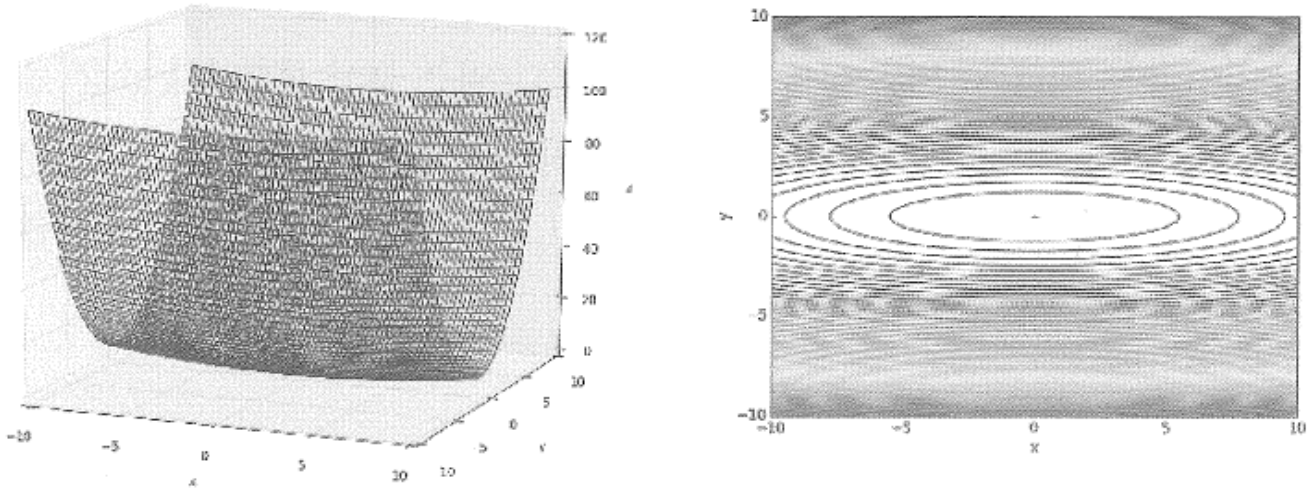
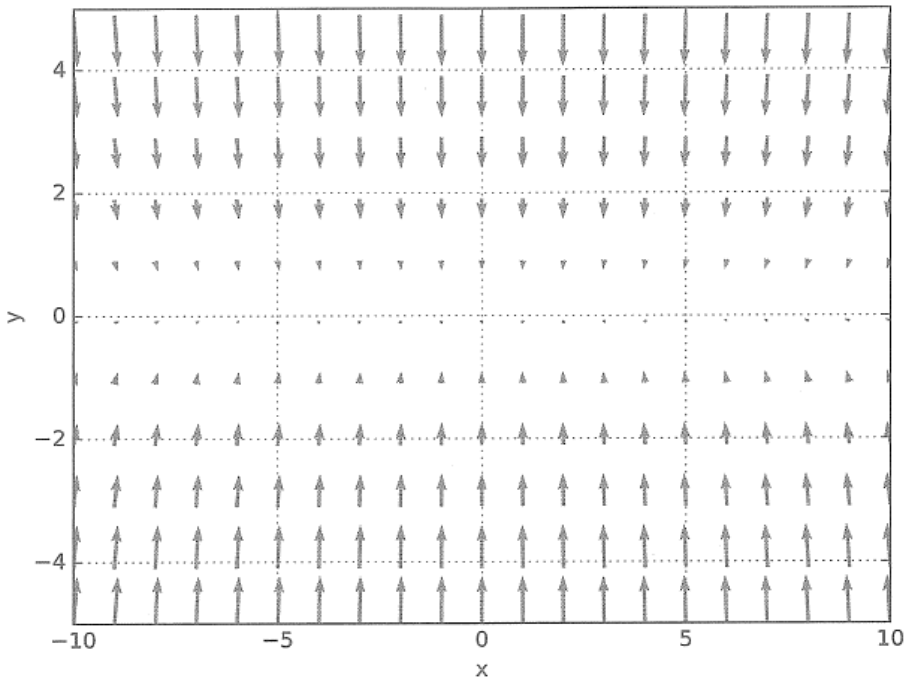


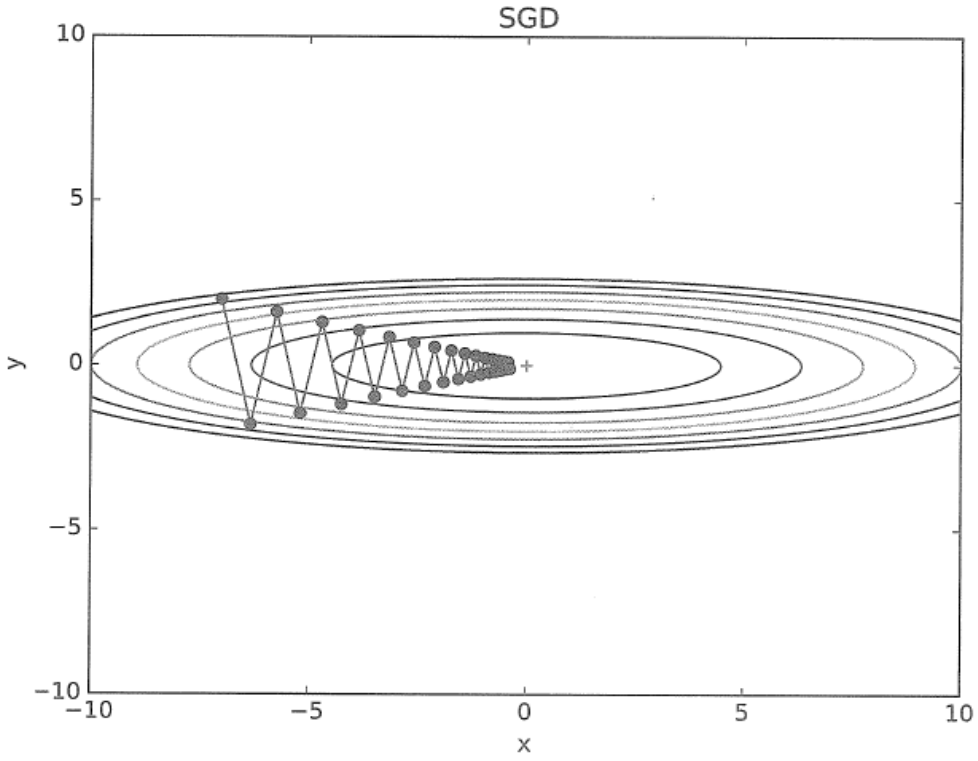
그림 6-2 $f(x,y) = \frac{1}{20}x^2 + y^2$의 기울기



Parameter Optimization- SGD

- 초기값 (x,y) = (-7.0, 2.0)
- 비등방성(anisotropy)함수 (방향에 따라 기울기가 달라지는 함수)에서는 탐색 경로가 비효율적
- 기울어진 방향이 본래의 최소값과 다른 방향을 가리키는 경우
- SGD의 단점을 개선해 주는 여러 다른 방법
  - Momentum, AdaGrad, Adam

그림 6-3 SGD에 의한 최적화 갱신 경로 : 최솟값인 (0, 0)까지 지그재그로 이동하니 비효율적이다.



Parameter Optimization- Momentum

- Momentum (운동량)
- optimizer.py

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

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

그림 6-5 모멘텀에 의한 최적화 갱신 경로

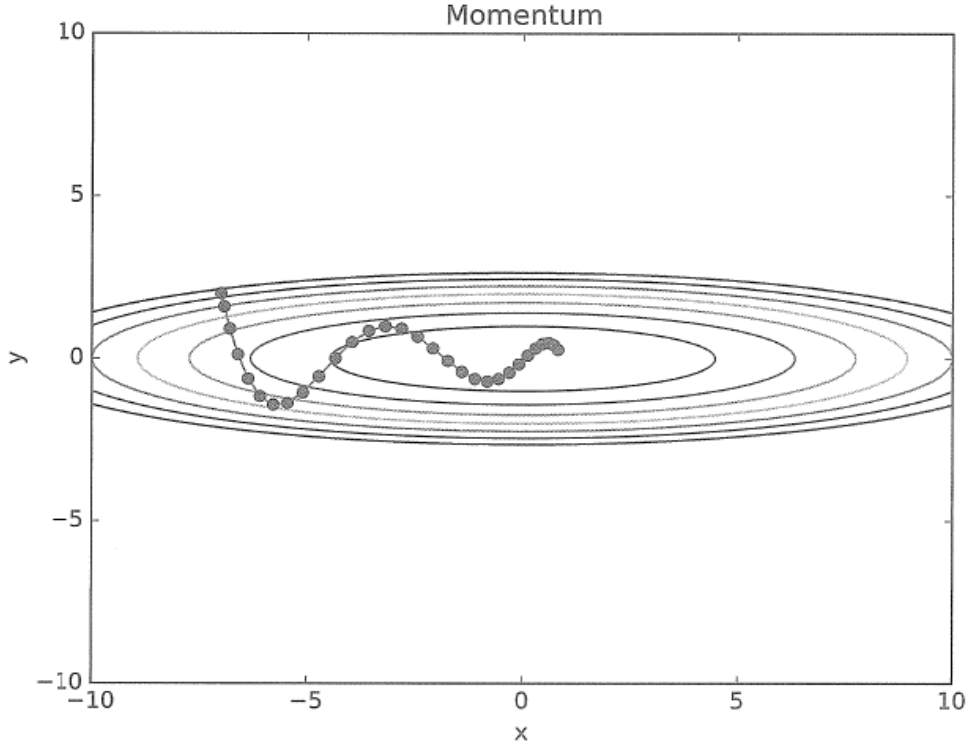


그림 6-4 모멘텀의 이미지 : 공이 그릇의 곡면(기울기)을 따라 구르듯 움직인다.

Parameter Optimization - AdaGrad

- Learning rate decay(학습률 감소)
  - 학습을 진행하면서 학습률을 점차 줄여가는 방법
  - 처음에는 크게 학습하다가 조금씩 작게 학습
- AdaGrad – 개별 매개변수에 적응적으로 학습률을 진행하면서 조정
  - 매개변수를 조정할 때 $\frac{1}{\sqrt{\mathrm{h}}}$ 을 곱해 학습률 조정
  - 매개변수 원소 중에서 많이 움직인(크게 갱신된) 원소는 학습률이 낮아짐
  - 학습률 감소가 매개변수의 원소마다 다르게 적용됨

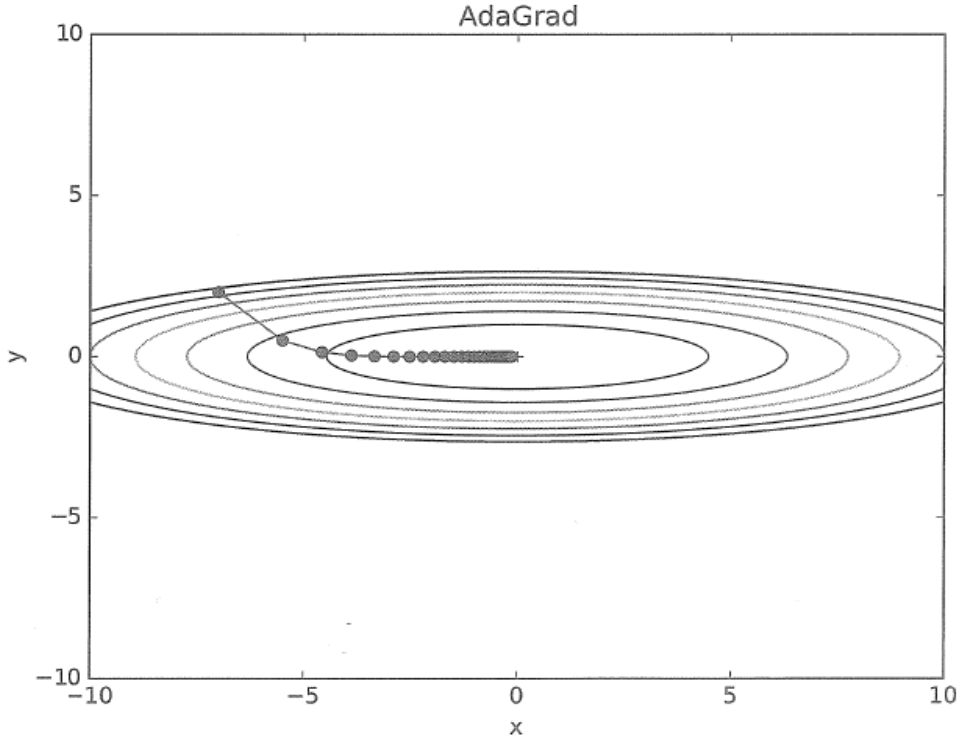
$$\mathrm{h} \leftarrow \mathrm{h} + \frac{\partial L}{\partial \mathrm{W}} \odot \frac{\partial L}{\partial \mathrm{W}}$$

$$\mathrm{W} \leftarrow \mathrm{W} + \eta \frac{1}{\sqrt{\mathrm{h}}} \frac{\partial L}{\partial \mathrm{W}}$$

Parameter Optimization -AdaGrad

- optimizer.py
- 그림 5-6에 의하면 최소값을 향해 효율적으로 움직임
- y축 방향은 기울기가 커서 처음에는 크게 움직이지만, 그 큰 움직임에 비례해 갱신 정도도 큰 폭으로 작아지도록 조정됨

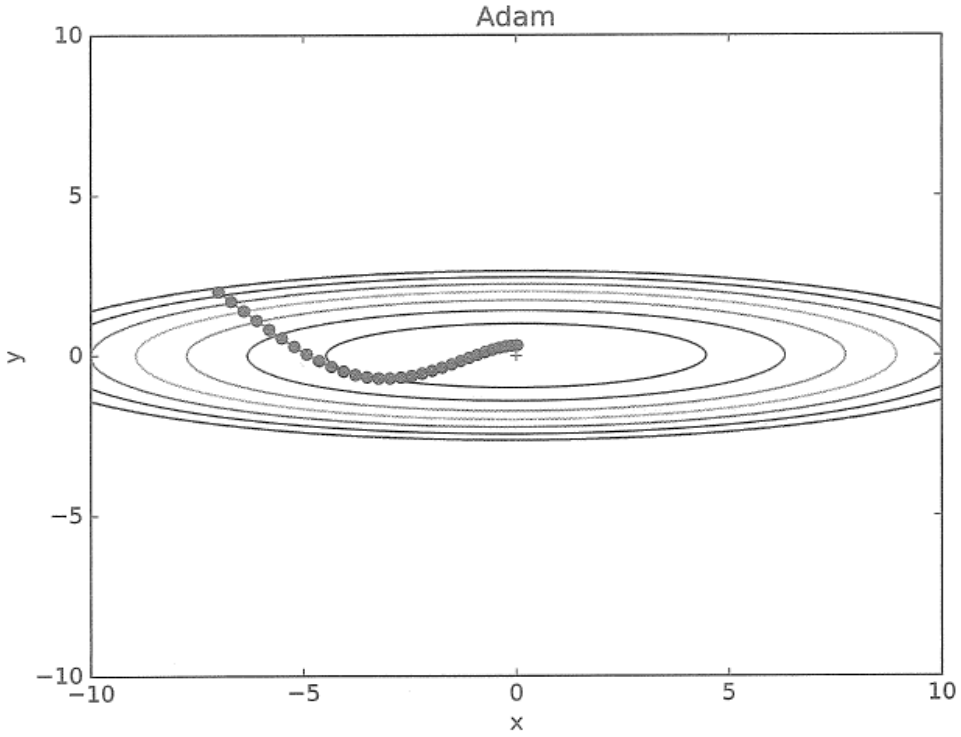
그림 6-6 AdaGrad에 의한 최적화 갱신 경로



Parameter Optimization -Adam

- Adam – momentum + AdaGrad
  - 하이퍼파라미터의 '편향보정'이 진행됨
  - 그림 6-7과 같이 Adam 갱신 과정도 그릇에서 공이 구르는 듯 움직임
  - Momentum과 비슷한 패턴이나, Momentum보다 공의 좌우 흔들림이 적다. 이는 학습 갱신 강도를 적응적으로 조정해서 얻는 혜택

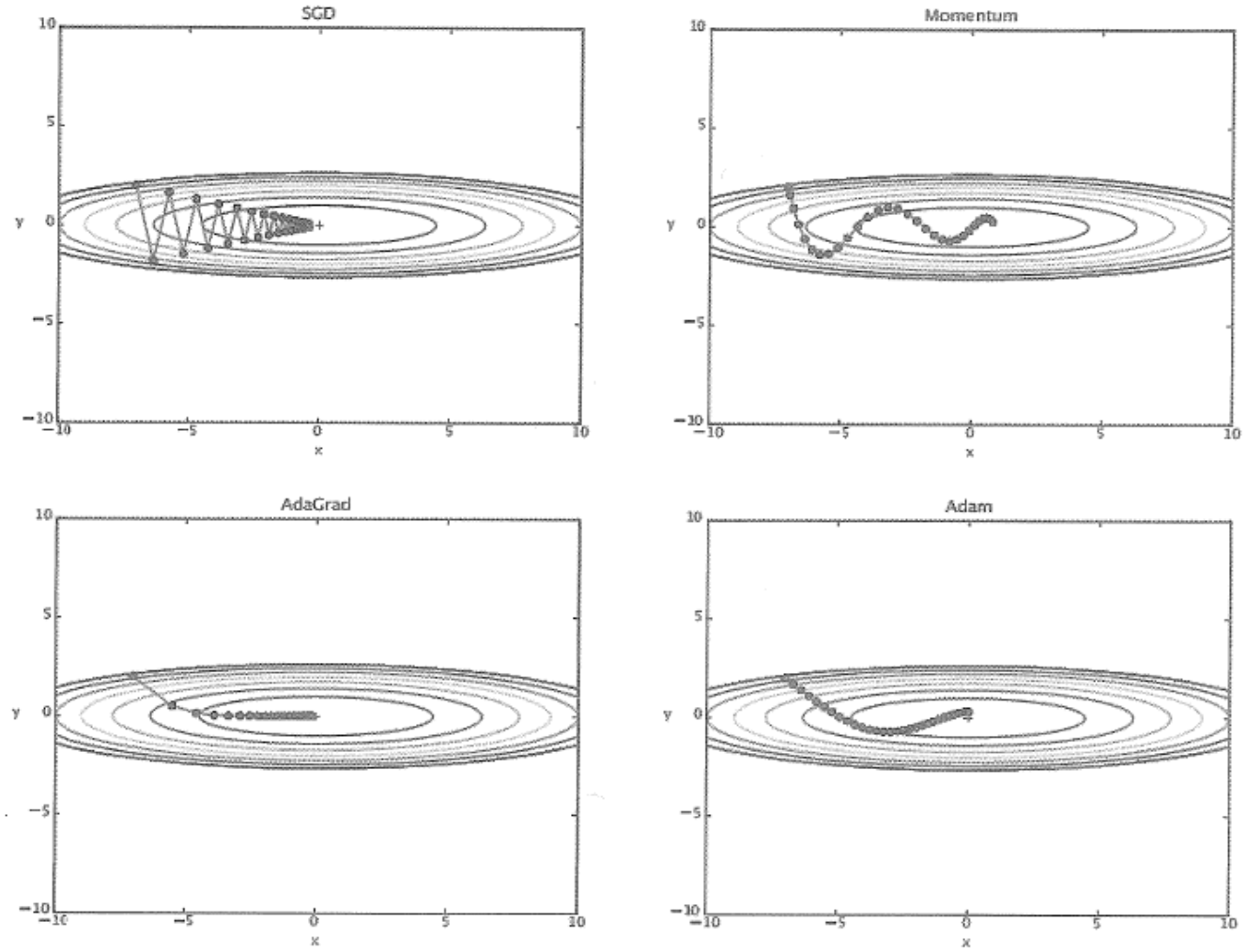
그림 6-7 Adam에 의한 최적화 갱신 경로



Parameter Optimization

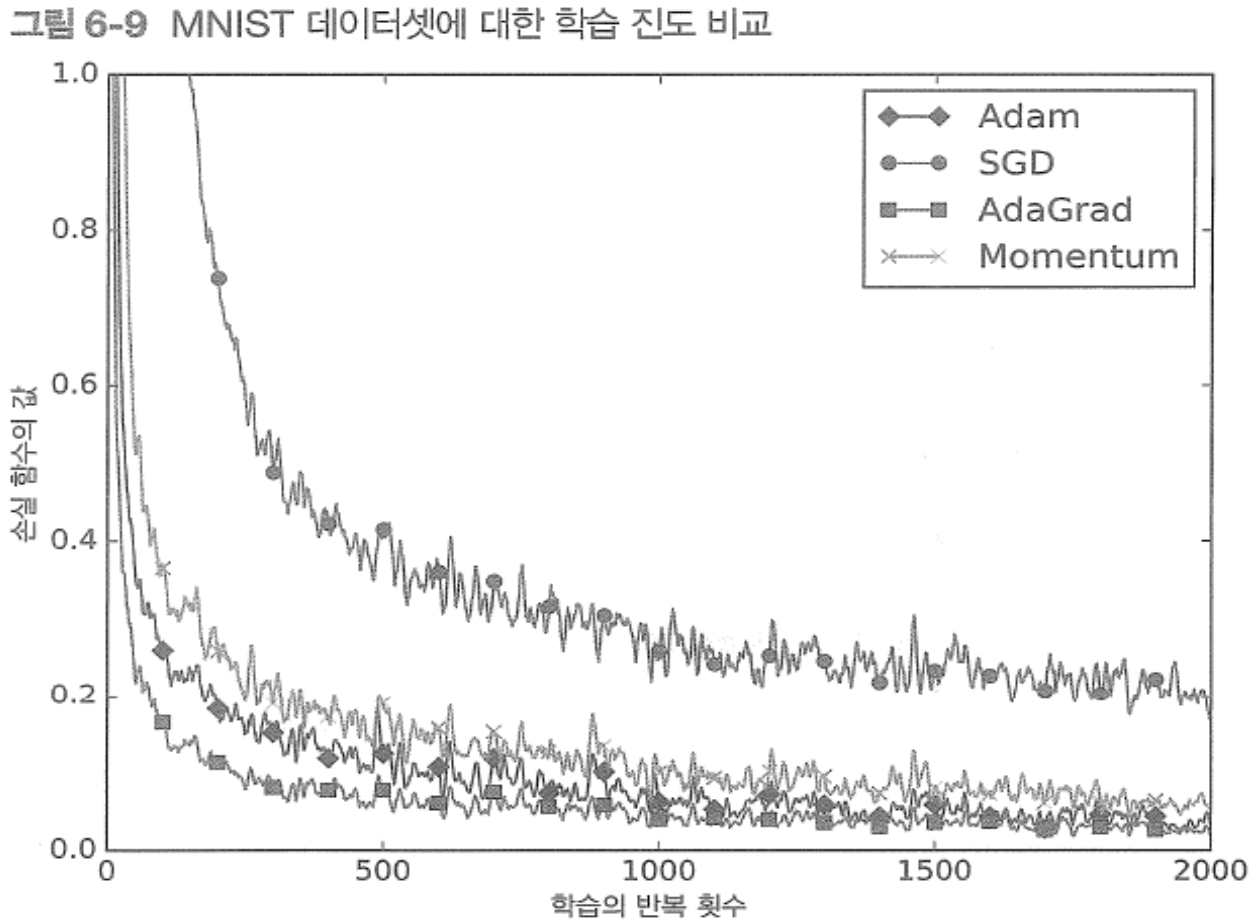
- optimizer_compare_naïve.py

그림 6-8 최적화 기법 비교 : SGD, 모멘텀, AdaGrad, Adam



Parameter Optimization

- MNIST 데이터 셋으로 본 갱신방법 비교
- optimizer_compare_mnist.py



Weight Init Value

- 가중치의 초기값 설정이 신경망 학습의 성패를 좌우
- 가중치 매개변수의 값이 작아지도록 학습하는 방법이 필요
- 가중치 값을 작게 하여 overfitting이 일어나지 않도록
- 가중치 초기값을 0으로 하면(균일한 값으로 설정하면)?
  - Backpropagation에서 모든 가중치의 값이 똑같이 갱신됨
  - 가중치들은 같은 초기값에서 시작하고 갱신을 거쳐도 여전히 같은 값을 유지
  - 가중치가 고르게 되는 상황을 피하려면 초기값을 무작위로 설정

Activation value Distribution in Hidden Layers

- 가중치의 초기값에 따라 hidden layer의 활성화 값이 어떻게 변화하는지
- 06-weight_init_activation histogram.py
- Gradient vanishing problem

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

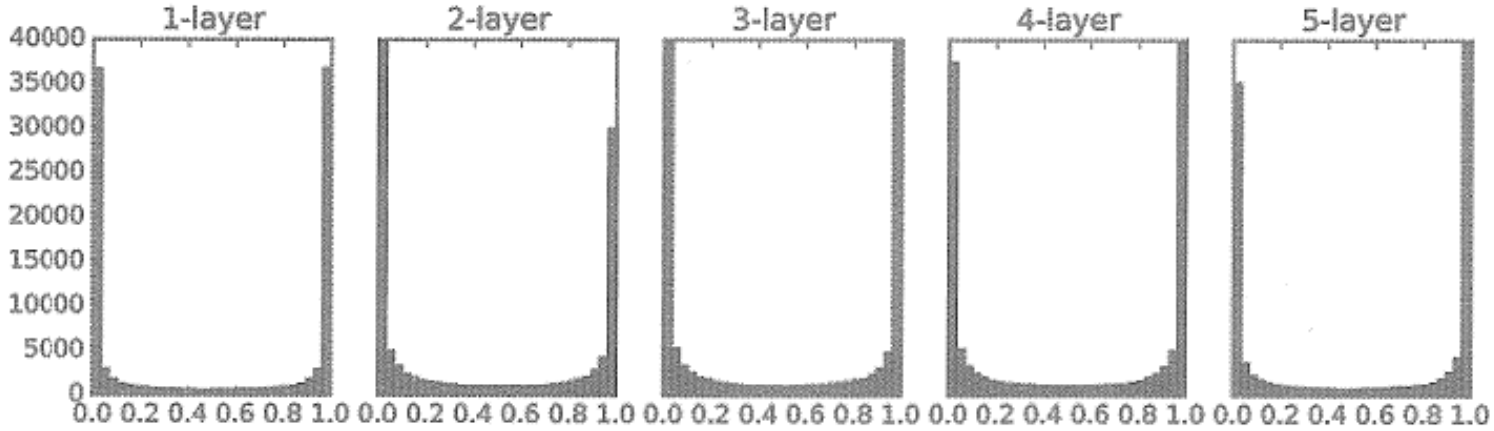
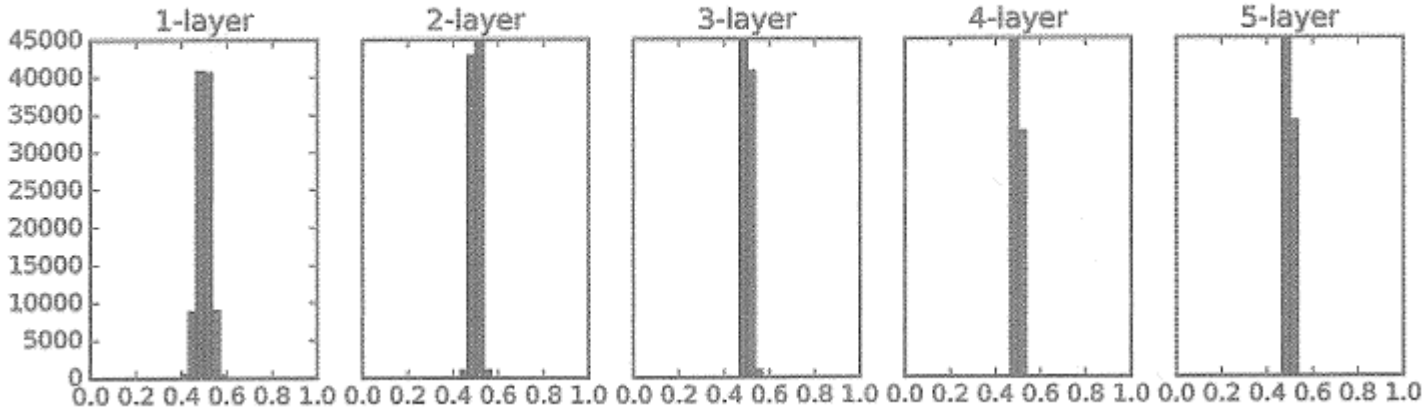


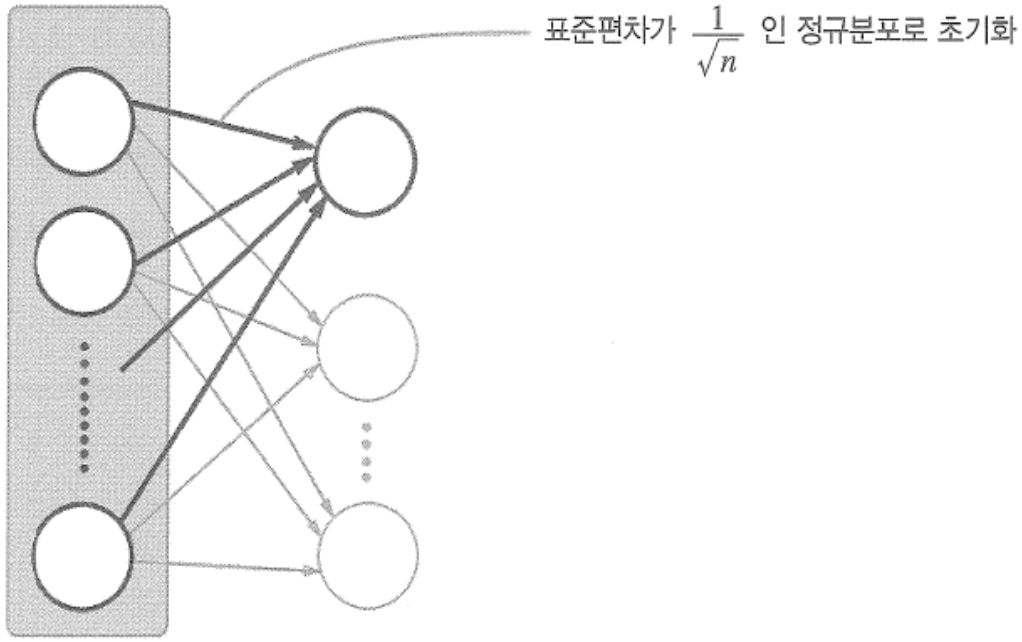
그림 6-11 가중치를 표준편차가 0.01인 정규분포로 초기화할 때의 각 층의 활성화값 분포



Xavier 초기값

- 앞 계층 n개 노드의 표준편차가 $\frac{1}{\sqrt{n}}$ 인 분포를 사용
- Xavier 초기값을 사용하면 앞 층에 노드가 많을 수록 대상 노드의 초기값으로 설정하는 가중치가 좁게 펴짐

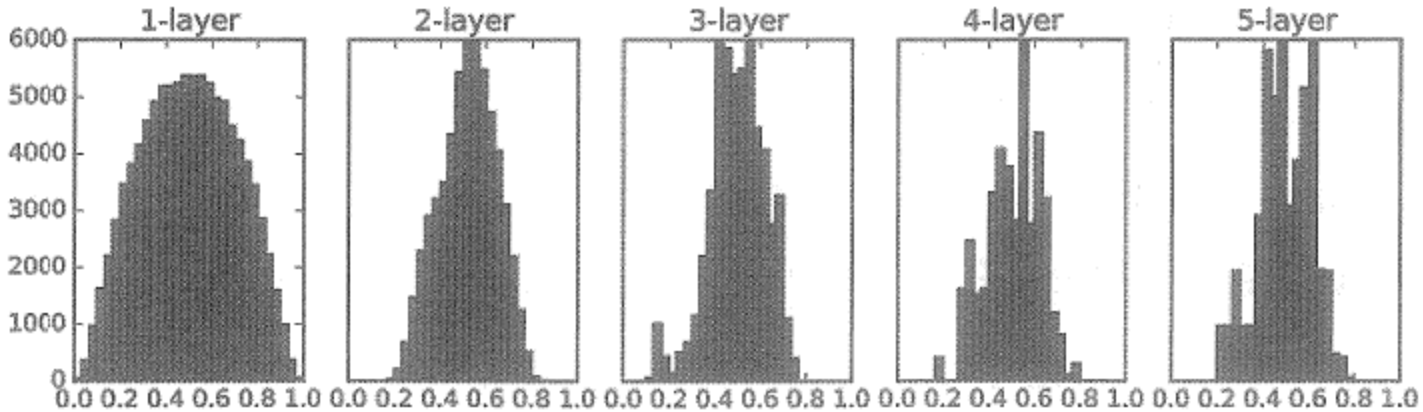
그림 6-12 Xavier 초깃값 : 초깃값의 표준편차가 $\frac{1}{\sqrt{n}}$ 이 되도록 설정 (n은 앞 층의 노드 수)



Xavier 초기값

```
node_num = 100 # 앞 층의 노드 수
w = np.random.randn(node_num, node_num) / np.sqrt(node_num)
```

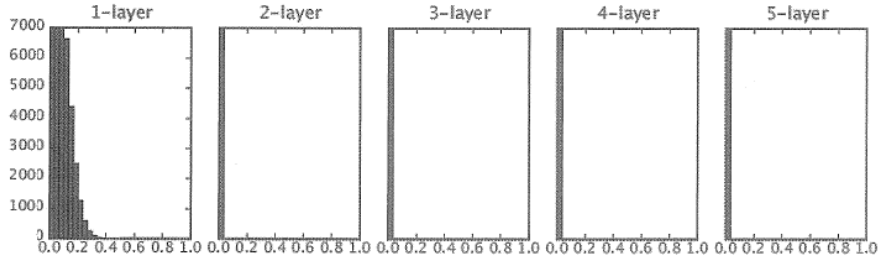
그림 6-13 가중치의 초깃값으로 'Xavier 초깃값'을 이용할 때의 각 층의 활성화값 분포



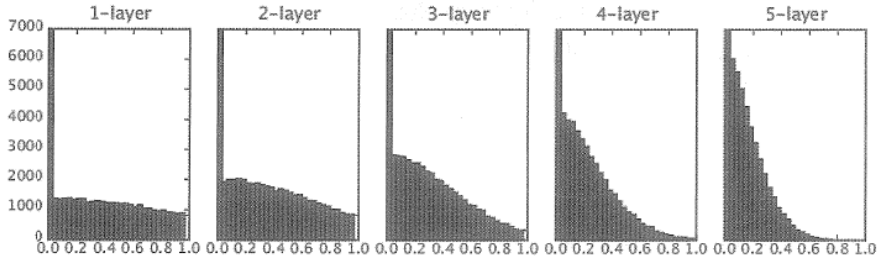
ReLU를 사용할 때의 가중치 초기값

- Xavier 초기값은 활성화 함수가 선형인 것을 전제
- ReLU의 경우는 이에 특화된 초기값 사용
  - He초기값(Kaiming He)
  - 표준편차가 $\sqrt{\frac{2}{n}}$ 인 정규분포를 사용
  - ReLU는 음의 영역이 0이라서 더 넓게 분포시키기 위해 2배의 계수가 필요

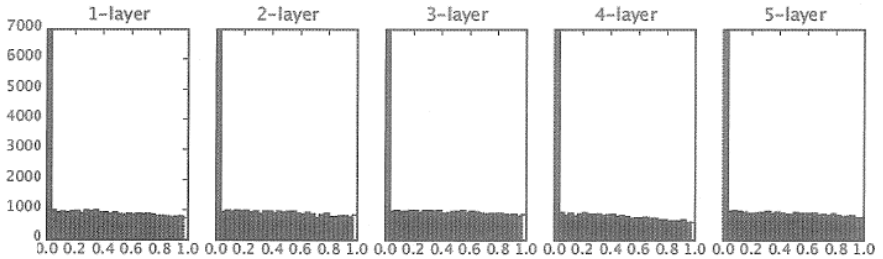
그림 6-14 활성화 함수로 ReLU를 사용한 경우의 가중치 초깃값에 따른 활성화값 분포 변화



표준편차가 0.01인 정규분포를 가중치 초깃값으로 사용한 경우



Xavier 초깃값을 사용한 경우

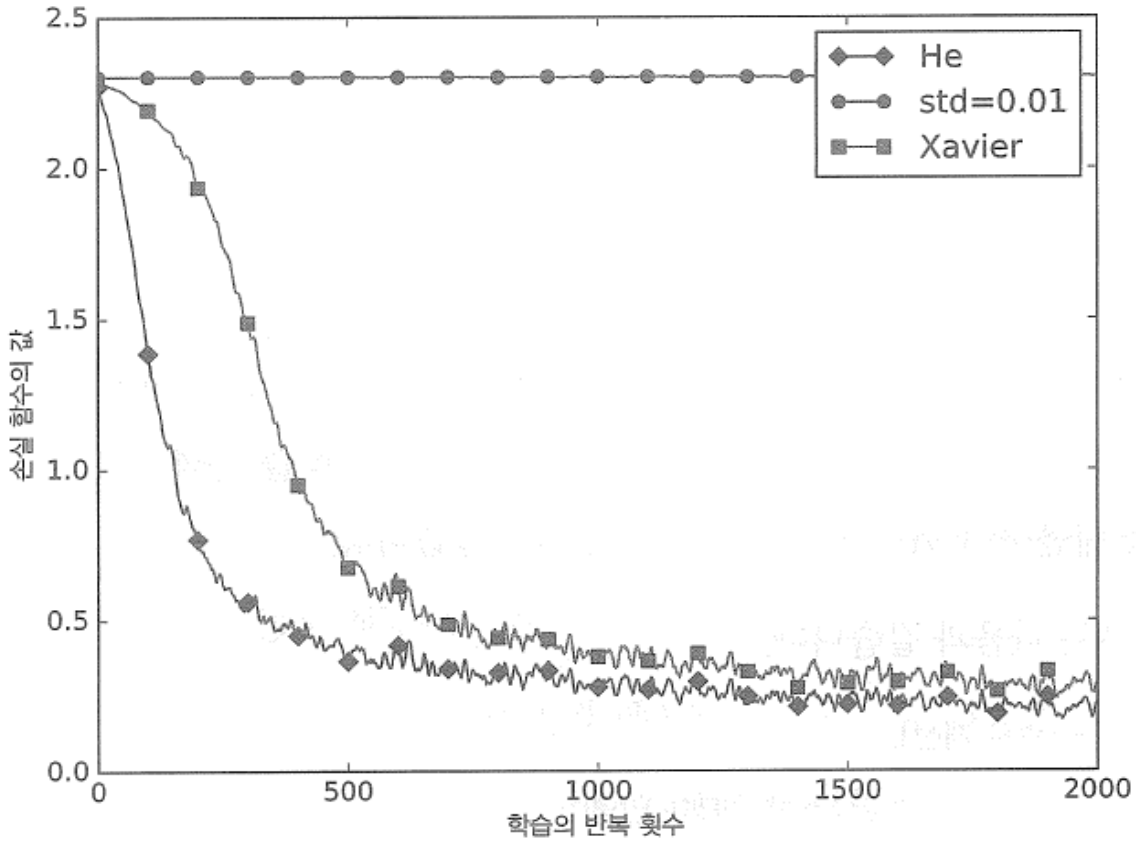


He 초깃값을 사용한 경우

MNIST데이터 셋으로 본 가중치 초기값 비교

- 06-weight_init_compare.py

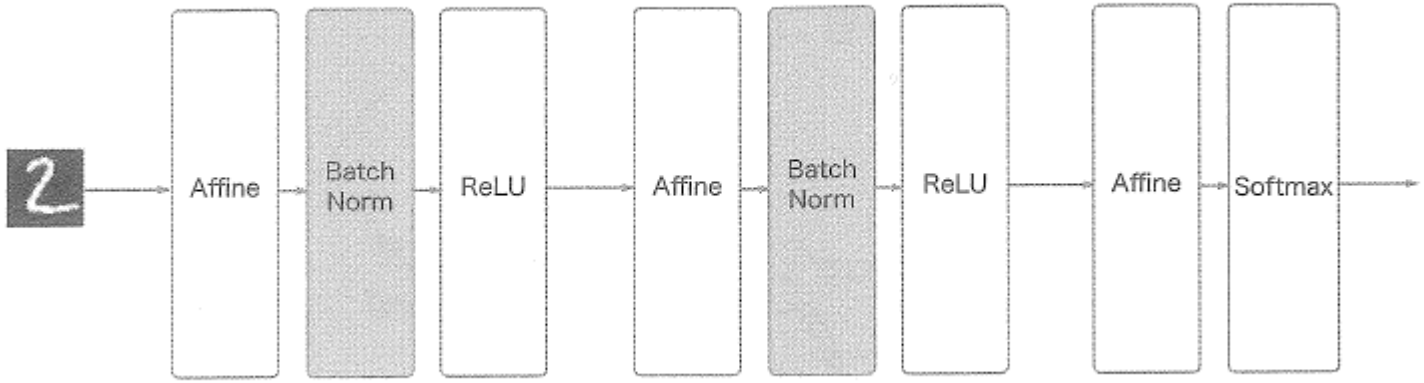
그림 6-15 MNIST 데이터셋으로 살펴본 '가중치의 초깃값'에 따른 비교



Batch Normalization

- 각 층의 활성화를 정규화하면?
  - 각 층에서의 활성화 값이 적당히 분포되도록 조정
- 장점
  - 학습을 빨리 진행할 수 있다(학습속도 개선)
  - 초기값에 크게 의존하지 않는다
  - Overfitting을 억제한다
-

그림 6-16 배치 정규화를 사용한 신경망의 예



Batch Normalization

- 데이터 분포가 평균이 0, 분산이 1이 되도록 정규화
- Scale(이동)과 이동(shift)

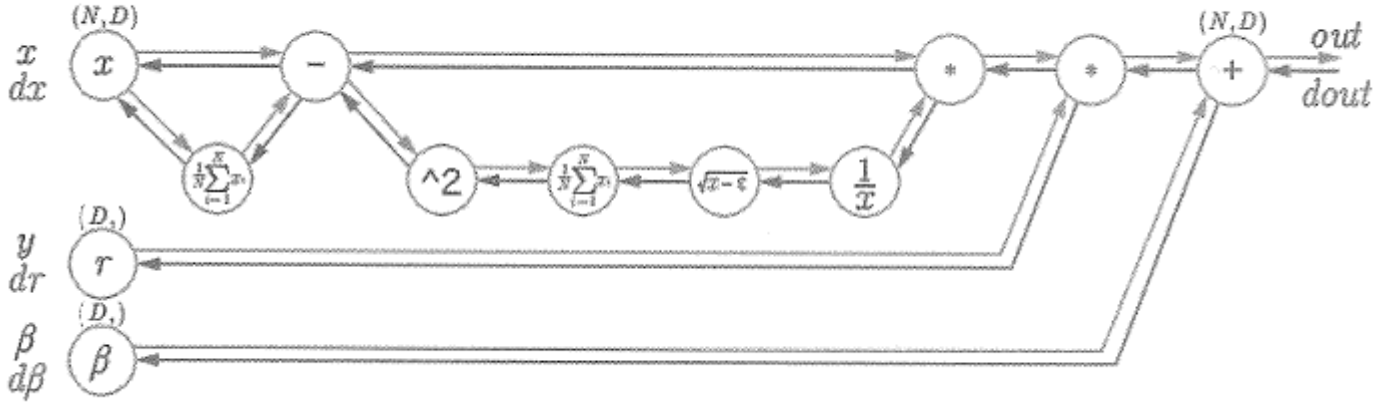
$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

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

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

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

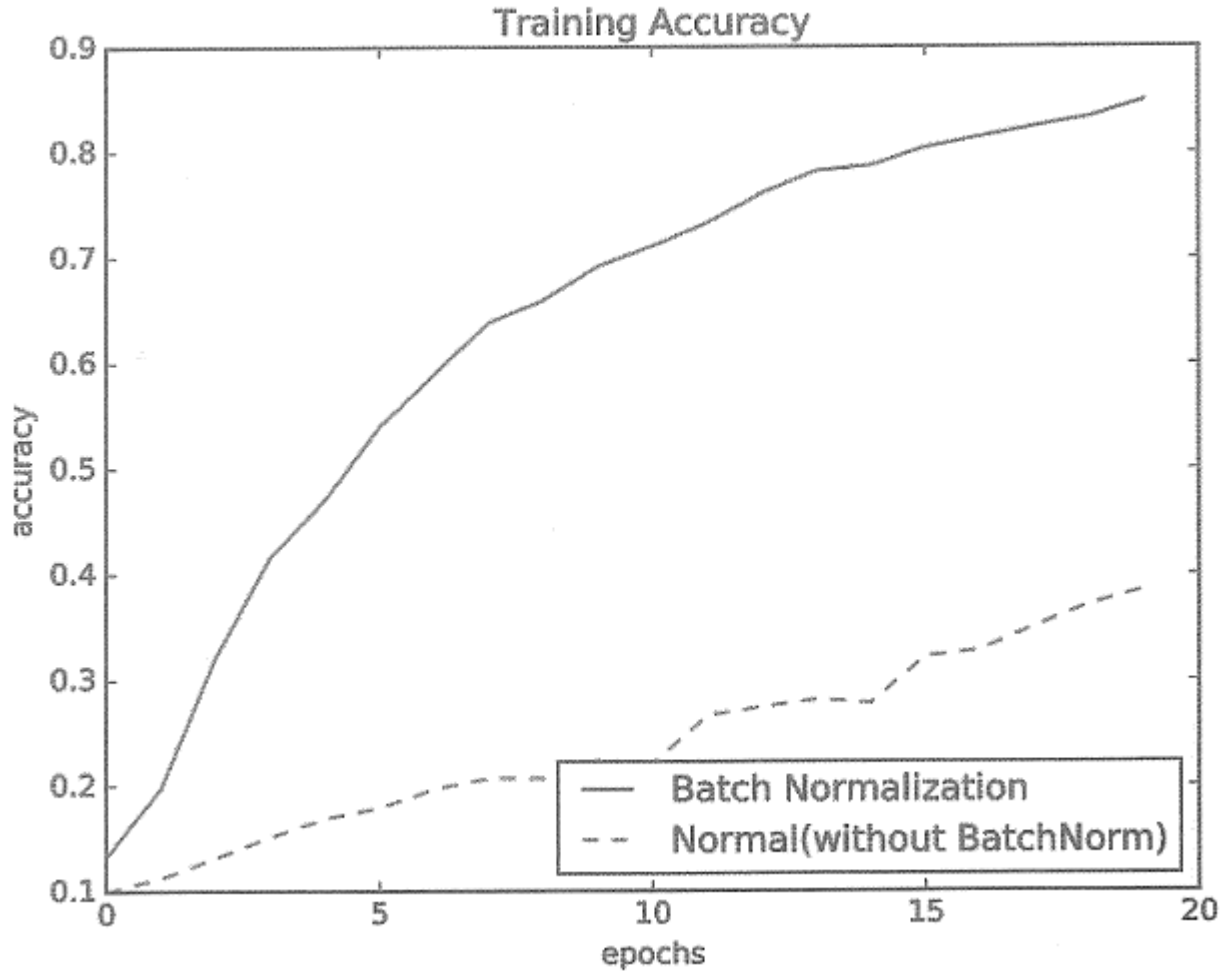
그림 6-17 배치 정규화의 계산 그래프[13]



Batch Normalization

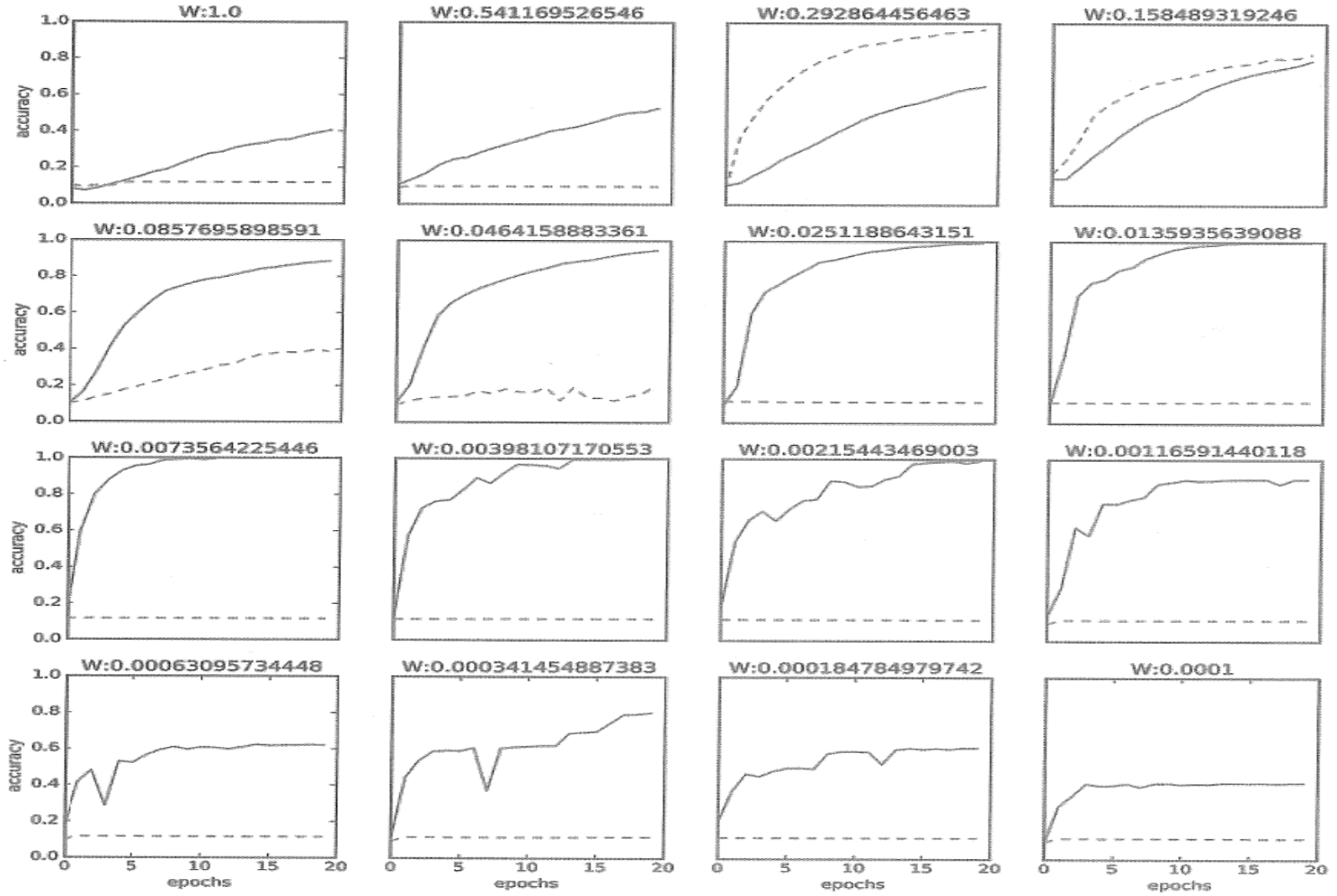
- MNIST 데이터셋
- 06-batch_norm_test.py

그림 6-18 배치 정규화의 효과 : 배치 정규화가 학습 속도를 높인다.



Batch Normalization

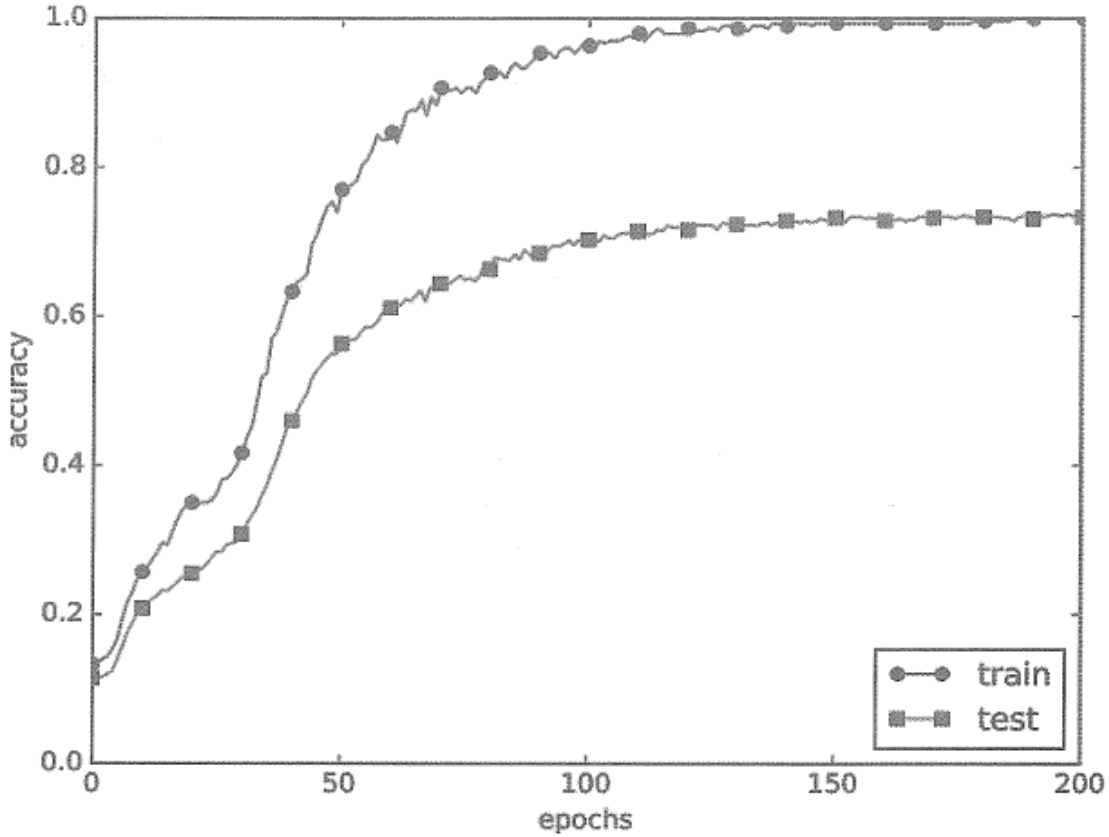
그림 6-19 실선이 배치 정규화를 사용한 경우, 점선이 사용하지 않은 경우 : 가중치 초깃값의 표준편차는 각 그래프 위에 표기



Overfitting

- Overfitting
  - 매개변수가 많고 표현력이 높은 모델일 경우
  - 훈련데이터가 적을 경우
- 06-overfit_weight_decay.py
  - 60,000 개의 데이터셋의 훈련 데이터 중 300개만 사용
  - 7층 네트워크를 사용해 복잡성을 높임
  - 각층의 뉴런은 100개, 활성함수는 ReLU를 사용

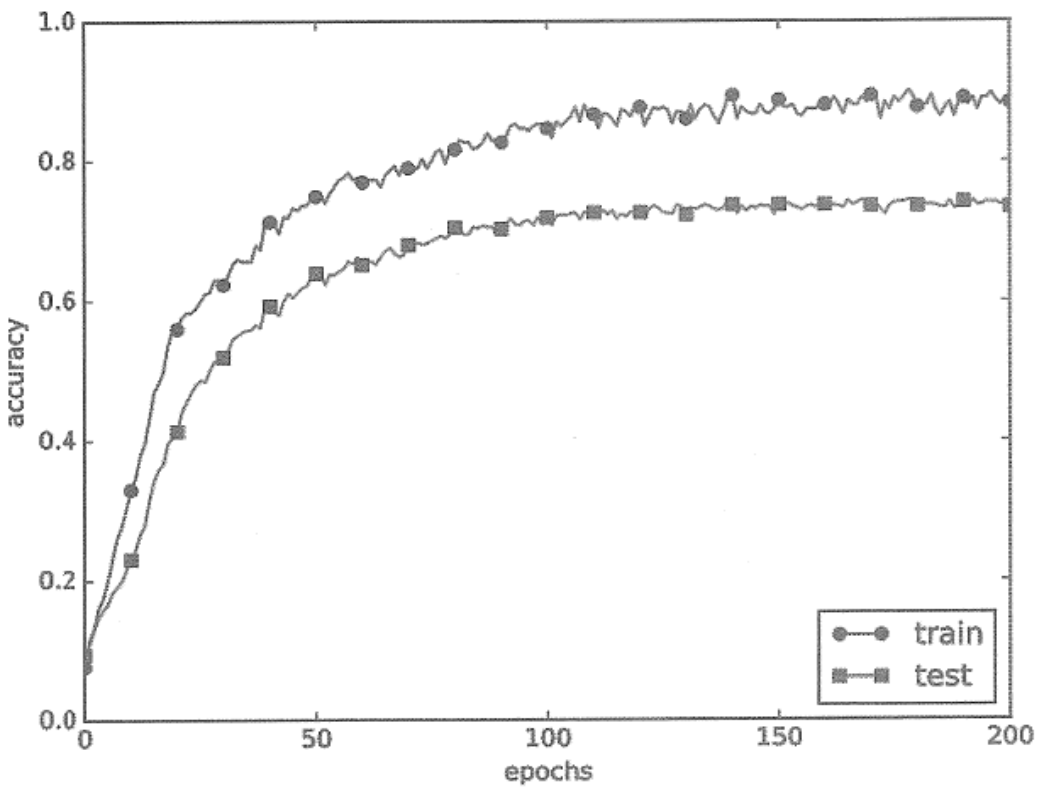
그림 6-20 훈련 데이터(train)와 시험 데이터(test)의 에폭별 정확도 추이



Weight decay

- 학습과정에서 큰 가중치에 대해서는 그에 상응하는 큰 패널티를 부과하여 overfitting억제
  - Overfitting은 가중치 매개변수의 값이 커서 발생하는 경우가 많기 때문
  - 가중치 감소- $\frac{1}{2}\lambda W^2$
  - 06-multi_layer_net.py

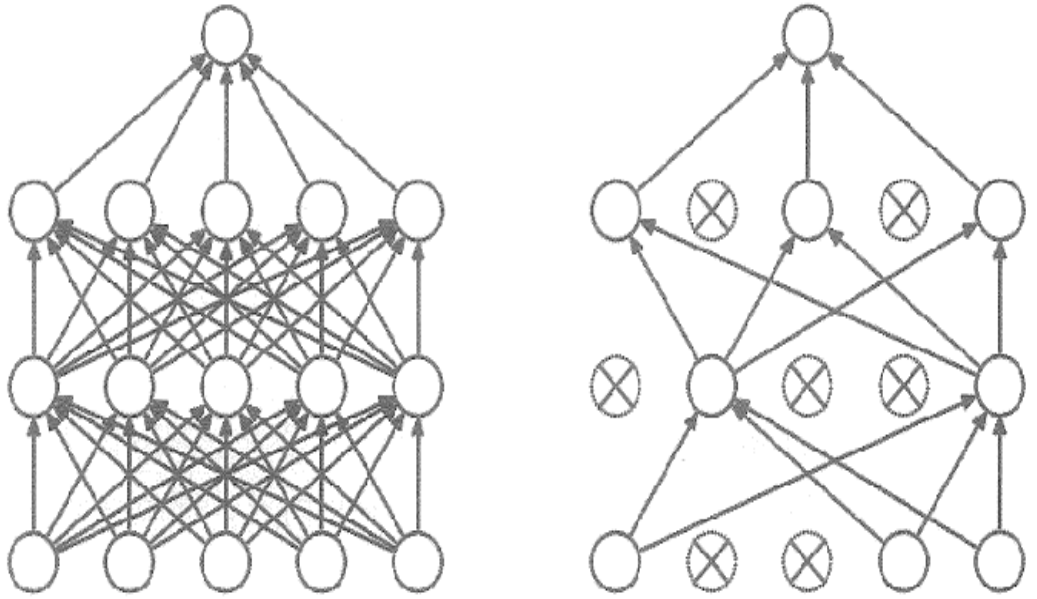
그림 6-21 가중치 감소를 이용한 훈련 데이터(train)와 시험 데이터(test)에 대한 정확도 추이



Dropout

- 뉴런을 임의로 삭제하면서 학습하는 방법
  - 학습시에 은닉층의 뉴런을 무작위로 골라 삭제, 시험 때는 모든 뉴런에 신호를 전달.
  - 시험 때는 각 뉴런의 출력에 훈련 때 삭제한 비율을 곱하여 출력
- 06-overfitdropout.py

그림 6-22 드롭아웃의 개념(문헌[14]에서 인용) : 왼쪽이 일반적인 신경망, 오른쪽이 드롭아웃을 적용한 신경망. 드롭아웃은 뉴런을 무작위로 선택해 삭제하여 신호 전달을 차단한다.

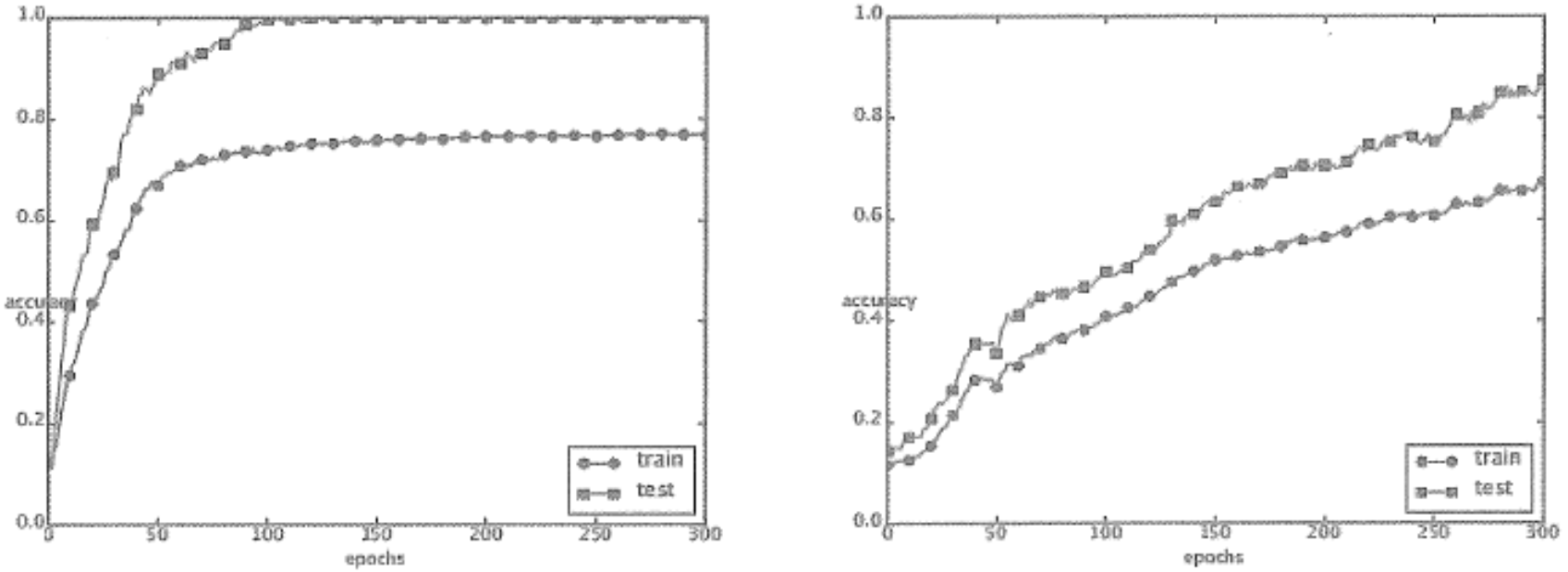


(a) 일반 신경망 (b) 드롭아웃을 적용한 신경망

Dropout

- 7층 네트워크(각 층의 뉴런 수는 100개, 활성화 함수는 ReLU)로 한 결과

그림 6-23 왼쪽은 드롭아웃 없이, 오른쪽은 드롭아웃을 적용한 결과 (dropout_ratio = 0.15)



Ensemble learning

- 개별적으로 학습시킨 여러 모델의 출력을 평균을 내어 추론하는 방식
  - 비슷한 구조의 네트워크를 준비하여 따로따로 학습시키고, 시험 때는 그것들의 출력을 평균을 내어 사용
- Dropout: 학습할 때 뉴런을 무작위로 삭제/추론때 뉴런의 출력에 삭제한 비율을 곱함<->Ensemble: 매번 다른 모델을 학습/추론때 여러 모델의 평균을 냄

Hyperparameter Optimization

- 검증데이터 (validation data) 설정
  - 훈련데이터: 매개변수 학습
  - 검증데이터: 하이퍼파라미터 성능평가
  - 시험 데이터:신경망의 범용성능 평가
- 최적화 (optimization)
  - Hyperparameter의 최적값이 존재하는 범위를 조금씩 줄여나감
  - 대략적인 범위를 설정하고 그 범위에서 무작위로 hyperparameter값을 골라낸 후 그 값으로 정확도를 평가
  - 정확도를 살피면서 여러 번 반복하며 hyperparameter의 최적값의 범위를 좁혀감

Hyperparameter optimization

- **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).

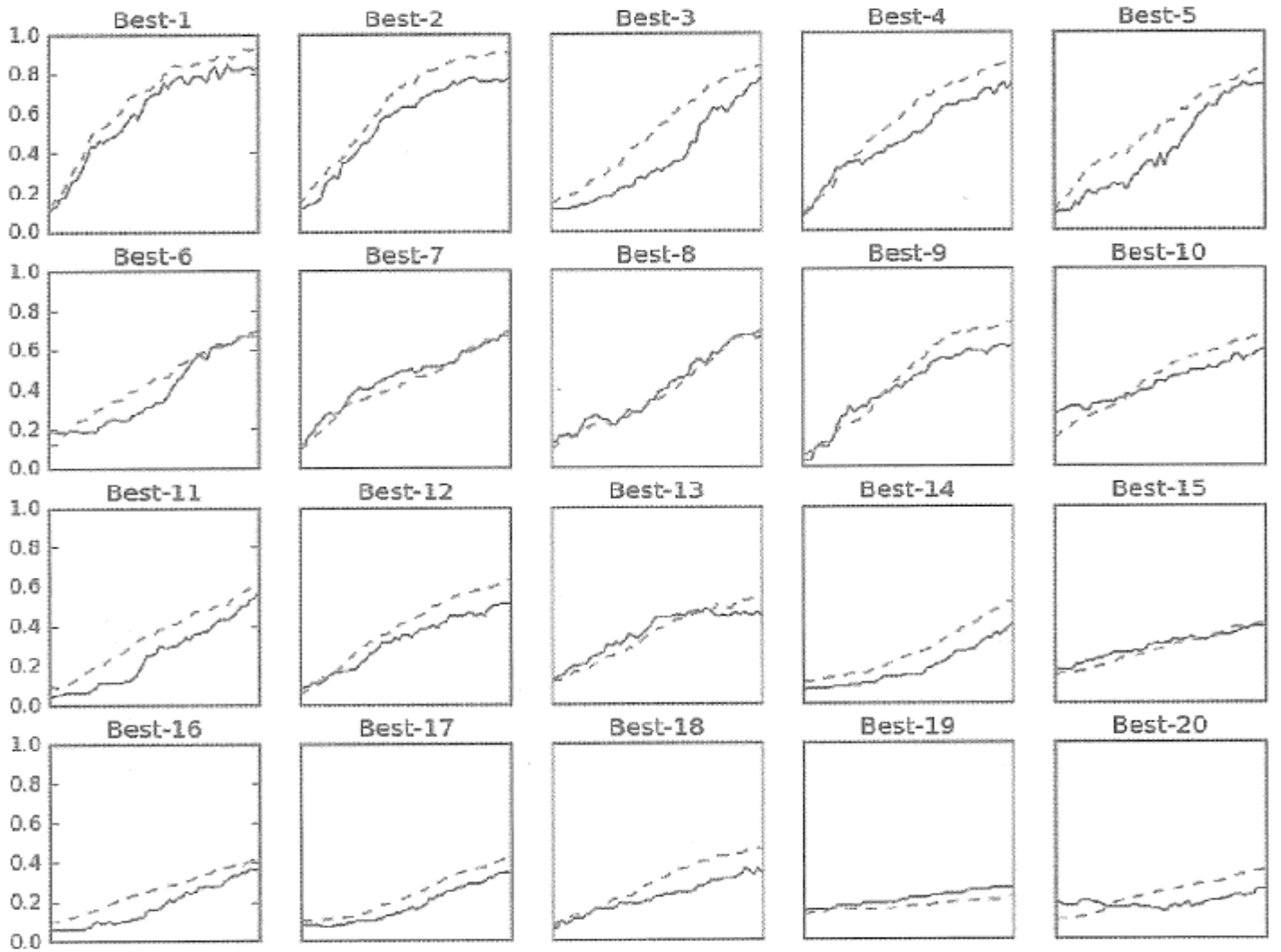
- **3단계**

  1단계와 2단계를 특정 횟수(100회 등) 반복하며, 그 정확도의 결과를 보고 하이퍼파라미터의 범위를 좁힌다.

Hyperparameter optimization

- 06-hyperparameter_optimization.py

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



RoundUP

- 매개변수 갱신 방법에는 SGD, Momentum, AdaGrad, Adam 등이 있다
- 가중치의 초기값을 정하는 방법은 올바른 학습을 하는 데 매우 중요하다
- 가중치의 초기값으로는 Xavier 초기값, He 초기값이 효과적이다
- 배치 정규화를 이용하면 학습을 빠르게 진행할 수 있으며 초기값에 영향을 덜 받게 된다
- Overfitting을 억제하는 정규화 방법으로는 weight decay와 dropout이 있다
- Hyperparameter 값 탐색은 최적값이 존재할 법한 범위를 점차 좁히면서 하는 것이 효과적이다