밑바닥부터 시작하는 딥러닝
6장. 학습 관련 기술

박채원

목차

6.1 매개변수 갱신

현재까지 최적의 매개변수 값을 찾는 단서 -> 매개변수의 기울기 이용 (확률적 경사 하강법, SGD)

$$W \leftarrow W - \eta \frac{\partial f}{\partial W}$$

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

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

optimizer(최적화를 행하는 자, 매개변수 갱신)로 사용

optimizer.update(params, grad) 로 사용
* params: 매개변수 , grad: 기울기 정보

6.1 매개변수 갱신

- 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)

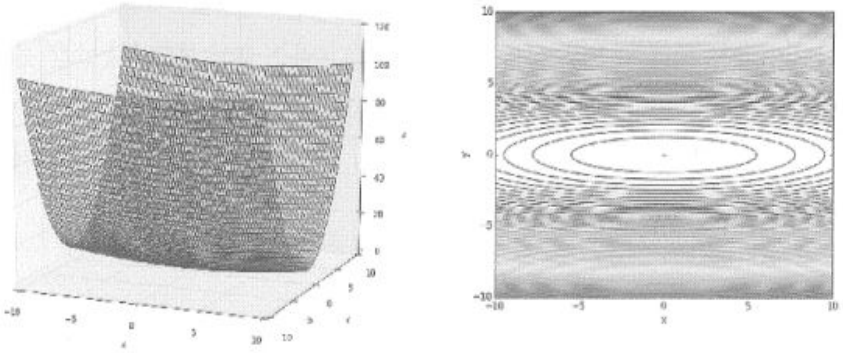
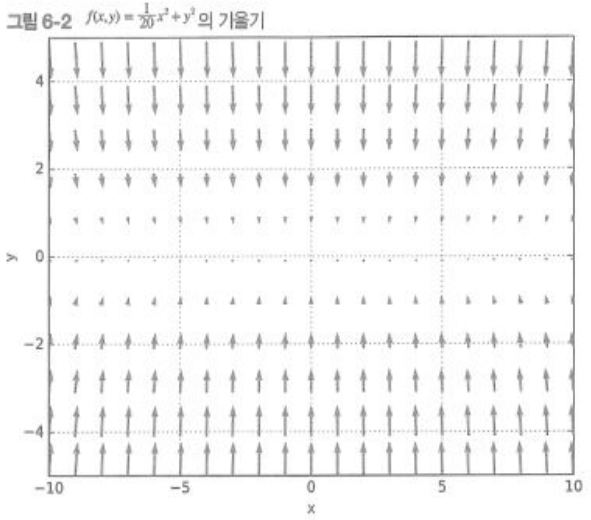
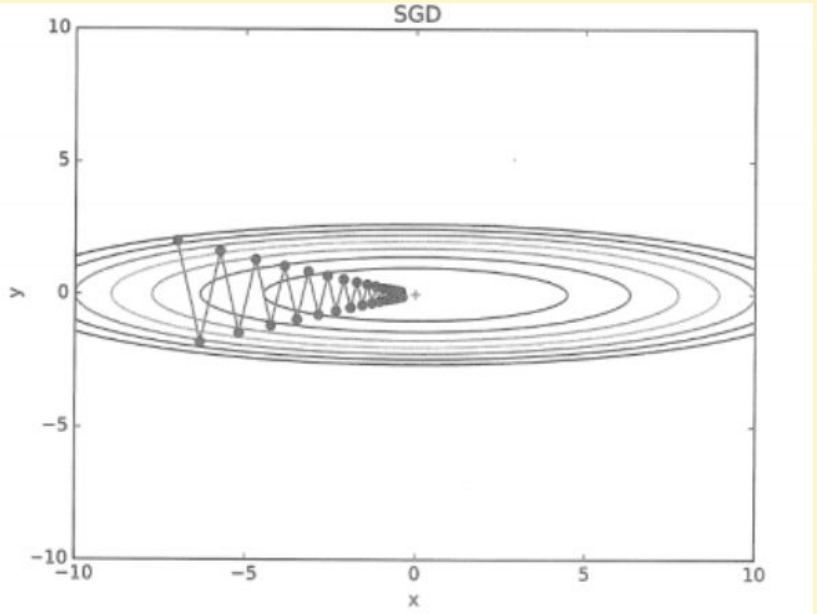


그림 6-2 $f(x,y) = \frac{1}{20}x^2 + y^2$의 기울기



함수 기울기의 특징 : y축 방향은 가파프고 x축 방향은 완만
함수가 최솟값을 갖는 장소는 (0,0)
하지만 기울기는 대체로 (0,0)을 가리키지 않는다.
즉 본래의 최솟값과 다른 방향을 가리킴



<SGD에 의한 최적화 갱신 경로>
비효율적인 움직임
비등방성 함수에선 탐색경로가 비효율적

6.1 매개변수 갱신

- 모멘텀(운동량)

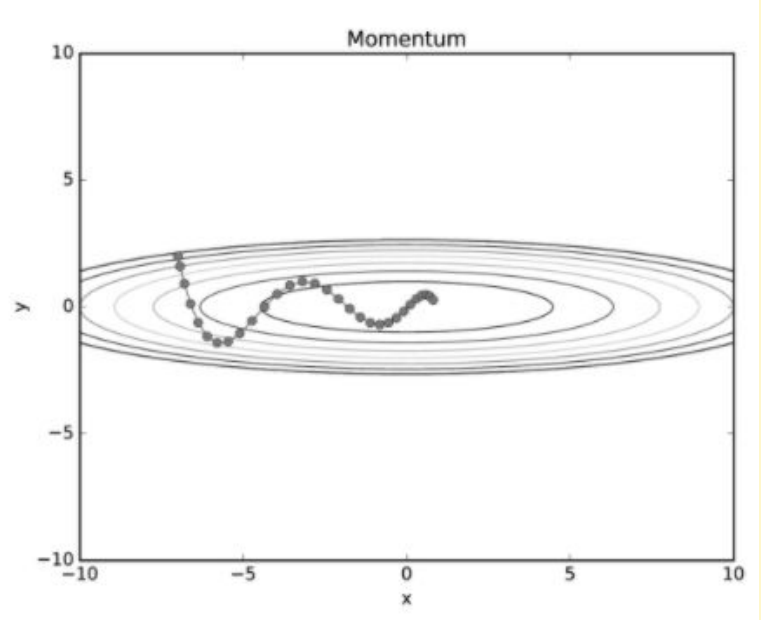
$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + v$$

*v = 속도

알파브이 항은 물체가 아무런 힘을 받지 않을 때 서서히 하강시키는 역할

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

  def update(selfm 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.key():
        self.v[key] = self.momentum*self.v[key] - self.lr*grad[key]
        params[key] += self.v[key]
```



6.1 매개변수 갱신

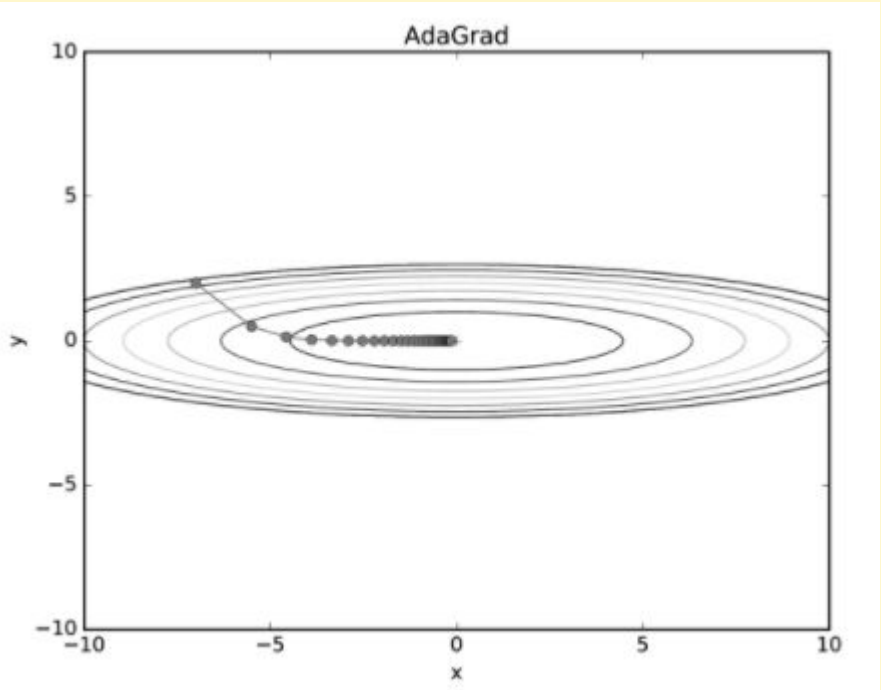
- AdaGrad

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

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

```
class AdaGrad:
  def __init__(self, lr=0.01, momentum=0.9):
    self.lr = lr
    self.h = None

  def update(selfm 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.key():
        self.h[key] += grads[key] * grads[key]
        params[key] -= self.lr * grads[key]/(np.sqrt(self.h[key]+1e-7))
```

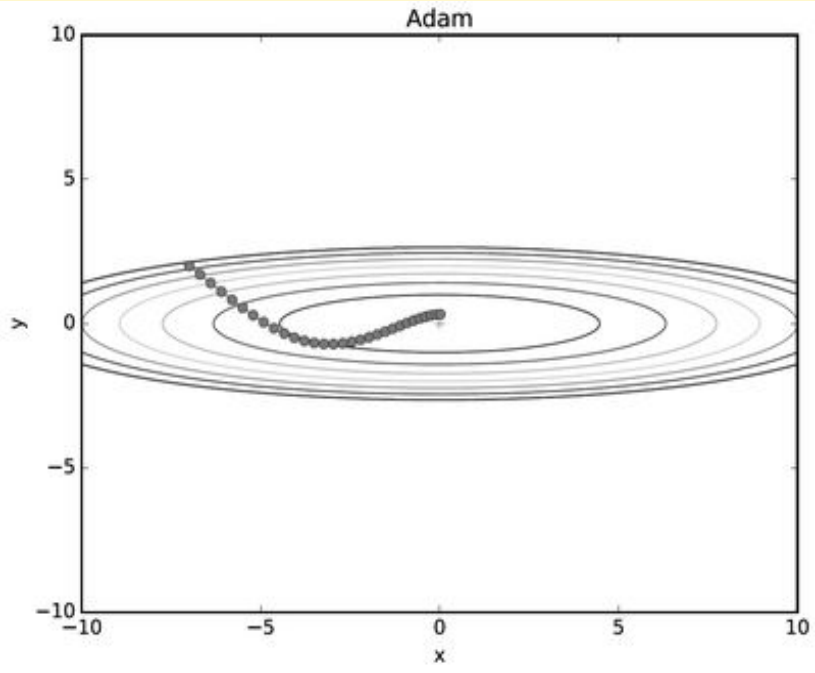
매개변수의 갱신률에 따른 학습률 감소
갱신률↑: 학습률↓



* RMSProp : AdaGrad의 갱신량 0이 되는 문제를 해결한 기법.
먼 과거의 기울기는 서서히 잊고 새로운 기울기 정보를 크게 반영

6.1 매개변수 갱신

- Adam



모멘텀과 AdaGrad, 두 기법을 융합.

- 매개변수 공간을 효율적으로 탐색해준다.
- 하이퍼파라미터의 편향보정이 진행된다.

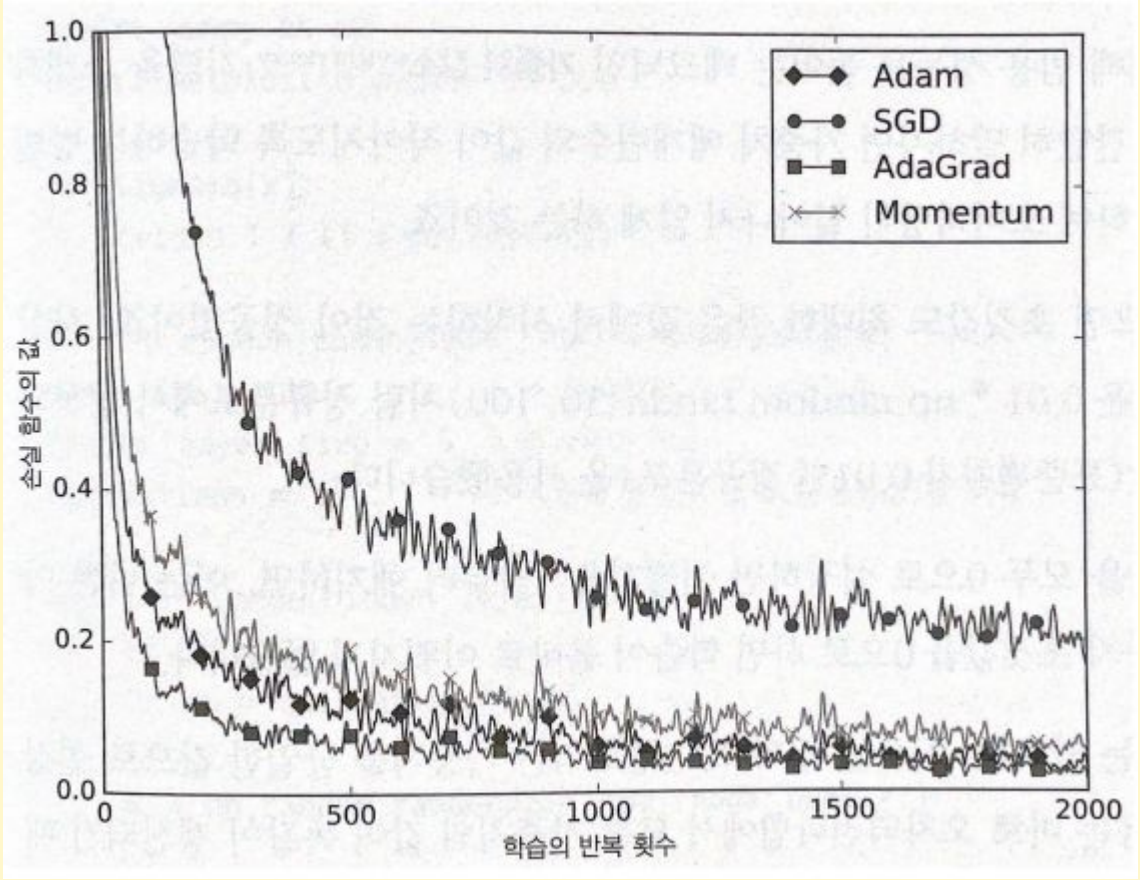
6.1 매개변수 갱신

모든 문제에서 뛰어난 기법은 아직 없다. 각자의 장단점이 존재.

- **MNIST 데이터셋으로 본 갱신 방법 비교**

5개의 층
각 층이 100개의 뉴런
ReLU 활성화 함수
* 하이퍼파라미터인 학습률과 층 깊이 등에 따라 결과는 달라진다.

일반적으로 SGD보다
다른 세 기법이 좋은 결과를
보여줌



6.2 가중치의 초깃값

- 가중치 감소 기법

  가중치 매개변수의 값이 작아지도록 학습하여 오버피팅이 일어나지 않게 하는 것.
  지금까지 가중치의 초깃값은 정규분포에서 생성되는 값을 0.01배 한 작은 값을 사용했다.

  Q. 그러면 가중치 초깃값을 0으로 하면 되는거 아니야?
  A. 오차역전파법에서 모든 가중치의 값이 똑같이 갱신되므로 가중치를 여러 개 갖는 의미가 없어짐.

  이처럼 가중치가 고르게 되어버리는 상황을 막기 위해선 초깃값을 무작위로 선정해야한다.

6.2 가중치의 초깃값

- 은닉층의 활성화 값 분포

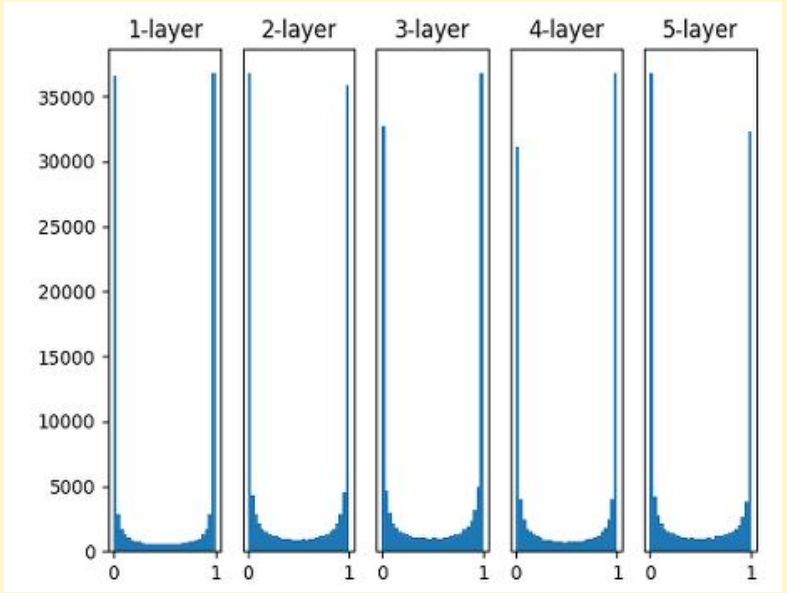
```
import numpy as np
import matplotlib.pyplot as plt


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

input_data = np.random.randn(1000, 100)  # 1000개의 데이터
node_num = 100  # 각 은닉층의 노드(뉴런) 수
hidden_layer_size = 5  # 은닉층이 5개
activations = {}  # 이곳에 활성화 결과를 저장

x = input_data

for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

    w = np.random.randn(node_num, node_num) * 1
    a = np.dot(x, w)
    z = sigmoid(a)
    activations[i] = z
```

층 5개
각 층의 뉴런은 100개씩
입력 데이터로서 1000개의 데이터를
정규분포로 무작위로 생성, 신경망에 흘린다.
활성화 함수 : **sigmoid 함수**

- 이 실험의 목적.
표준편차 값을 바꿔가며 활성화 값들의
분포가 어떻게 변화하는지 관찰.

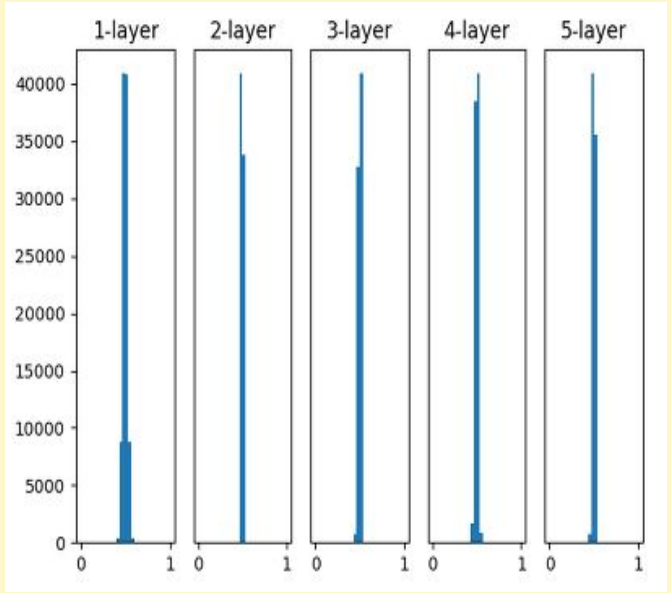
6.2 가중치의 초깃값

- 은닉층의 활성화 값 분포
  w=np.random.randn(node_num, node_num) * **std**



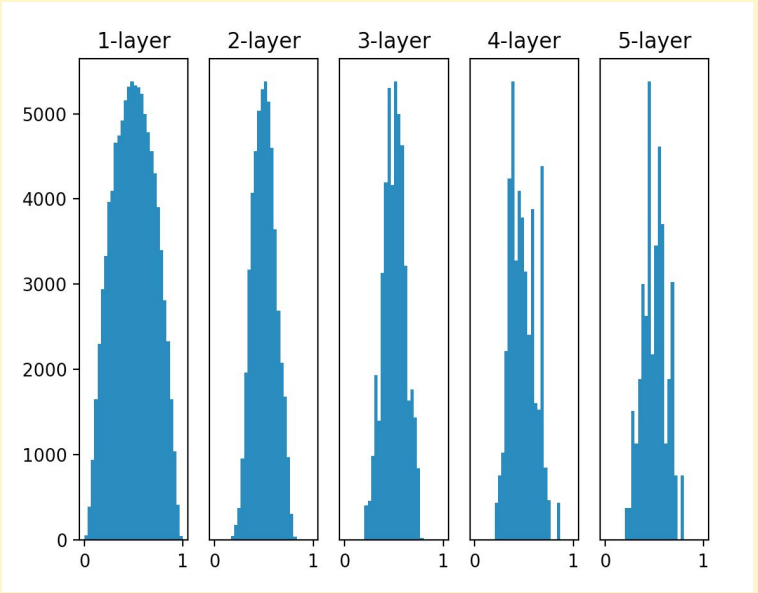
표준편차 1

기울기 소실 문제



표준편차 0.01

표현력 제한 문제



Xavier 초깃값

표준편차 = $\sqrt{\frac{1}{n}}$

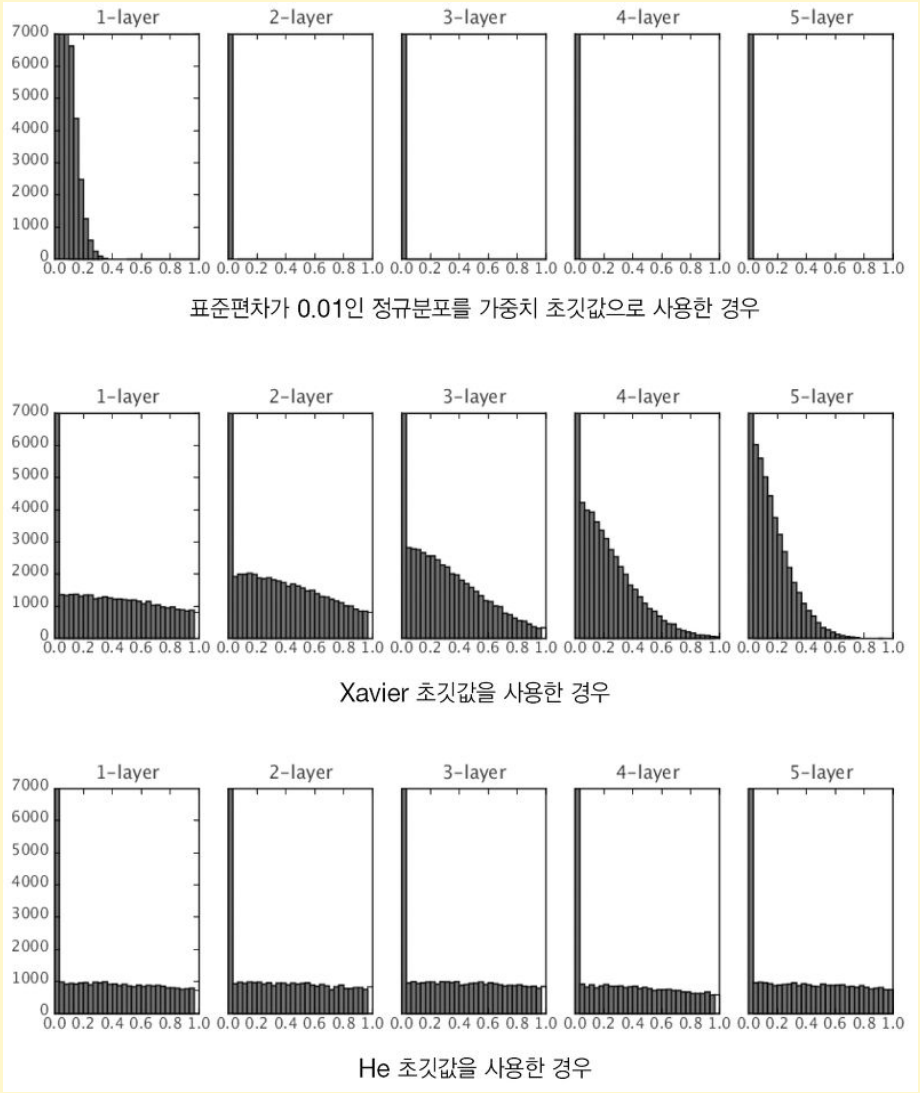
6.2 가중치의 초깃값

- 은닉층의 활성화 값 분포
  + 활성화 함수 **ReLU 사용**

표준편차 0.01 : 활성화값들이 매우 작다.
역전파때 기울기 또한 작아진다.

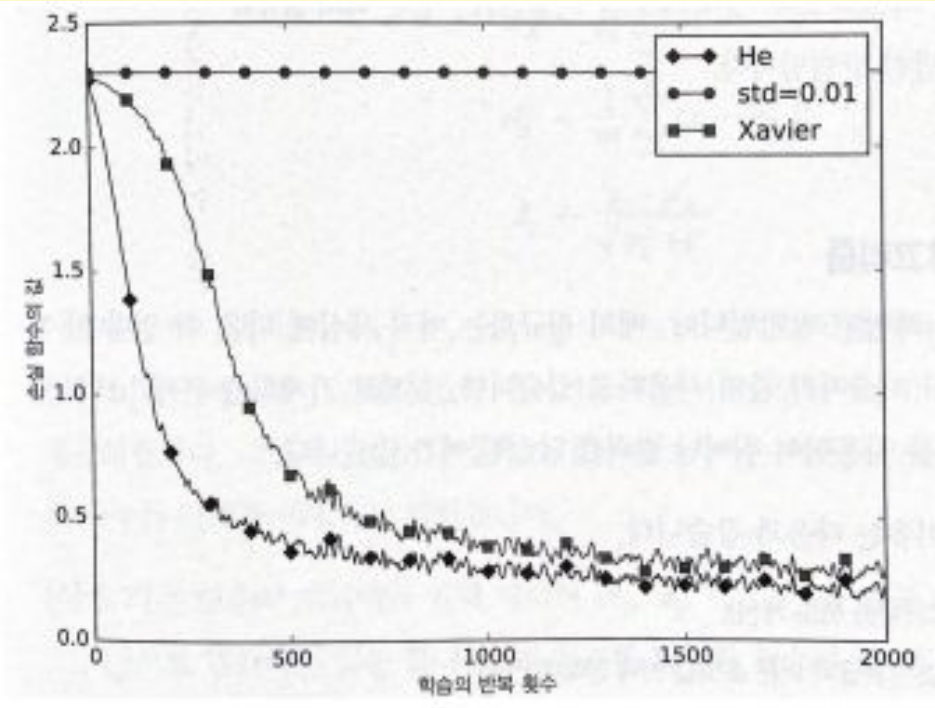
Xavier 초깃값 : 층이 깊어지며 점점 0으로 치우침
기울기 소실 문제

He 초깃값 : 모든 층에서 균일하게 분포
↓
(표준편차 = $\sqrt{\frac{2}{n}}$)



6.2 가중치의 초깃값

- MNIST 데이터셋으로 본 가중치 초깃값 비교



5개의 층
층별 뉴런수가 100개
활성화 함수-ReLU

표준편차=0.01 -> 학습이 전혀 이루어지지 않는다.

Xavier와 He 모두 학습이 순조롭게 이뤄지고 있다.
학습진도는 He 초기값이 더 빠르다.

6.3 배치 정규화

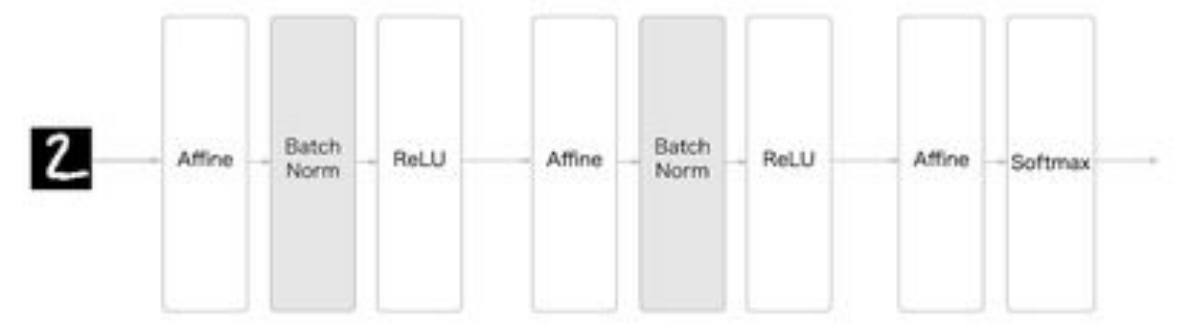
- 배치 정규화
  : 각 층이 활성화를 적당히 퍼뜨리도록 강제 해보면 어떨까?

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

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

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

데이터 분포를 정규화하는 배치 정규화 계층을 신경망에
삽입한다.
미니배치를 단위로 평균은 0, **분산은 1**의 데이터로 정규화한다.
활성화 함수의 앞 혹은 뒤에 삽입.
* 엡실론 값은 0으로 나누는 상태를 예방하기 위한 역할



배치 정규화의 장점

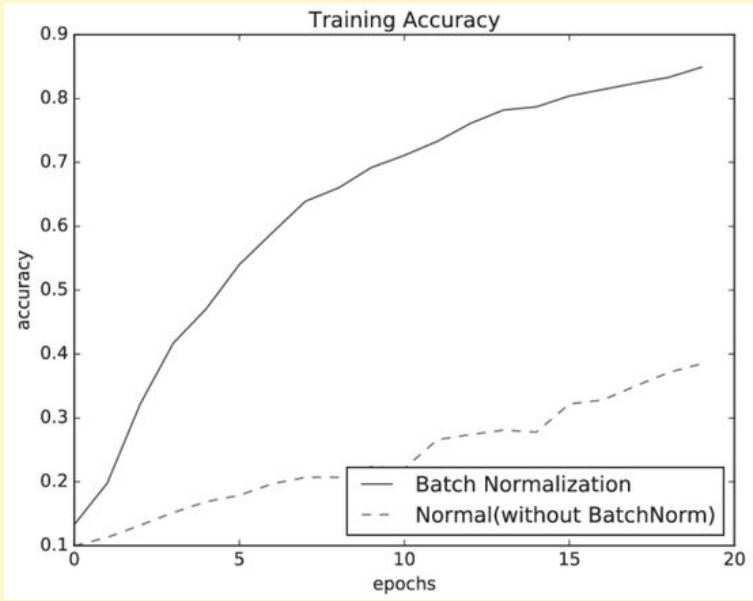
1. 학습을 빨리 진행할 수 있다.
2. 초깃값에 크게 의존하지 않는다.
3. 오버피팅을 억제한다.

6.3 배치 정규화

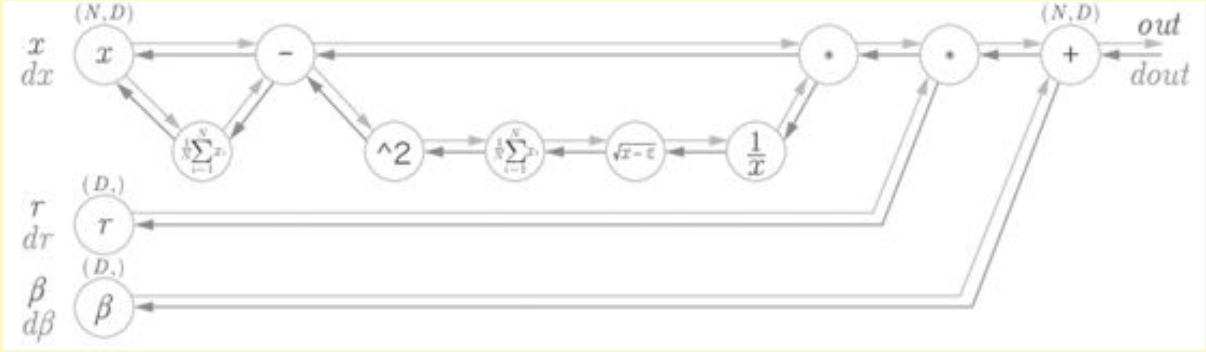
- 배치 정규화 (정규화된 데이터 확대&이동)

$$y_i -> \gamma \hat{x}_i + \beta$$

감마가 확대를, 베타가 이동을 담당.
초기값은 감마=1, 베타=0
학습하면서 적합한 값으로 조정해간다.

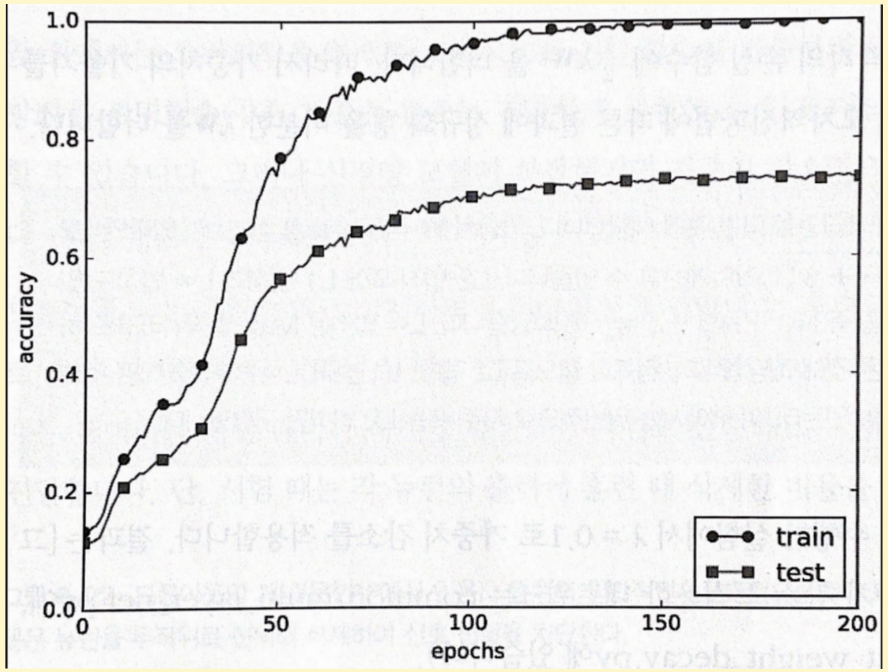


- 배치 정규화의 계산 그래프



6.4 바른 학습을 위해

- 오버피팅



오버피팅이란?
: 신경망이 훈련 데이터에만 지나치게 적응되어 그 외의 데이터에는 제대로 대응하지 못하는 상태.

오버피팅은 주로 다음의 두 경우에 일어난다.
- 매개변수가 많고 표현력이 높은 모델
- 훈련 데이터가 적음

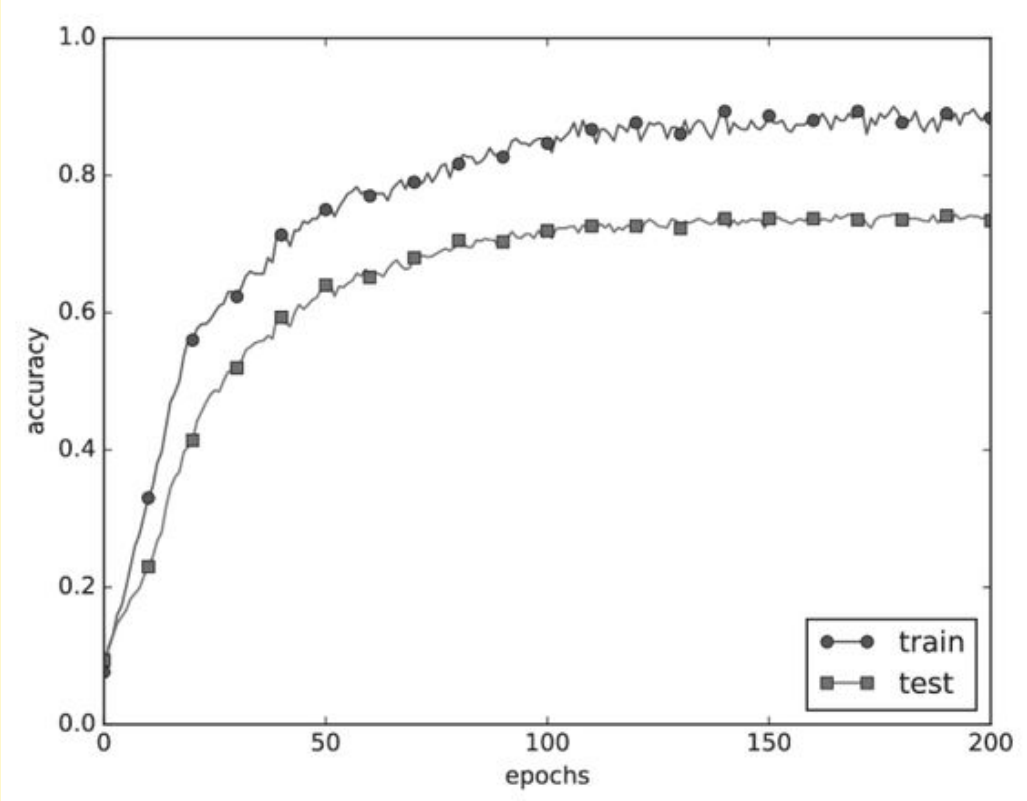
6.4 바른 학습을 위해

- 가중치 감소

가중치의 제곱법칙(L2법칙)을 손실함수에 더한다.
그러면 가중치가 커지는걸 억제할 수 있다.

L2법칙에 따른 가중치 감소는 $\frac{1}{2}\lambda W^2$
*람다는 정규화의 세기를 조절하는
하이퍼파라미터
*람다를 크게 설정할수록 큰 가중치에 대한
패널티.
*1/2는 가중치 감소 값의 미분의 결과인 λW 를
조정하는 역할의 상수 $\frac{1}{2}\lambda W^2$

가중치 감소는 모든 가중치 각각의 손실 함수에
를 더한다.
따라서 가중치의 기울기를 구하는 계산에서는
그동안의 오차역전파법에 따른 결과에 정규화
항을
미분한 λW 를 더한다.

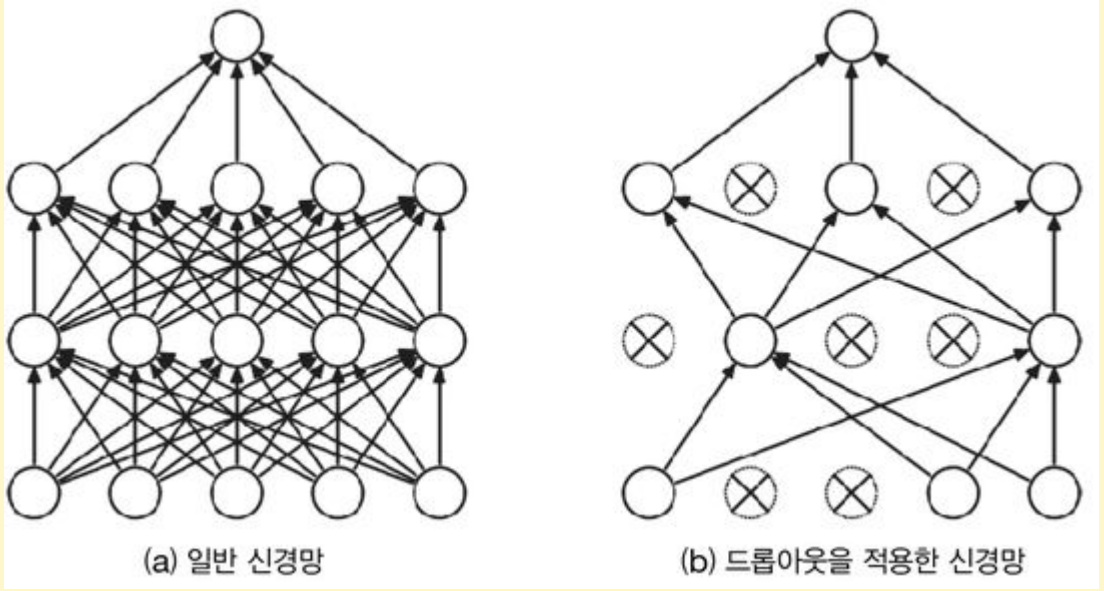


가중치 감소를 이용한 후
학습데이터와 테스트 데이터에 대한
정확도
(람다=0.1)

6.4 바른 학습을 위해

신경망 모델이 복잡해지면 가중치 감소만으로는 대응하기 어려워진다.
이럴 때 흔히 드롭아웃 사용

- 드롭아웃 (앙상블과 비슷한 효과)



(a) 일반 신경망　　(b) 드롭아웃을 적용한 신경망

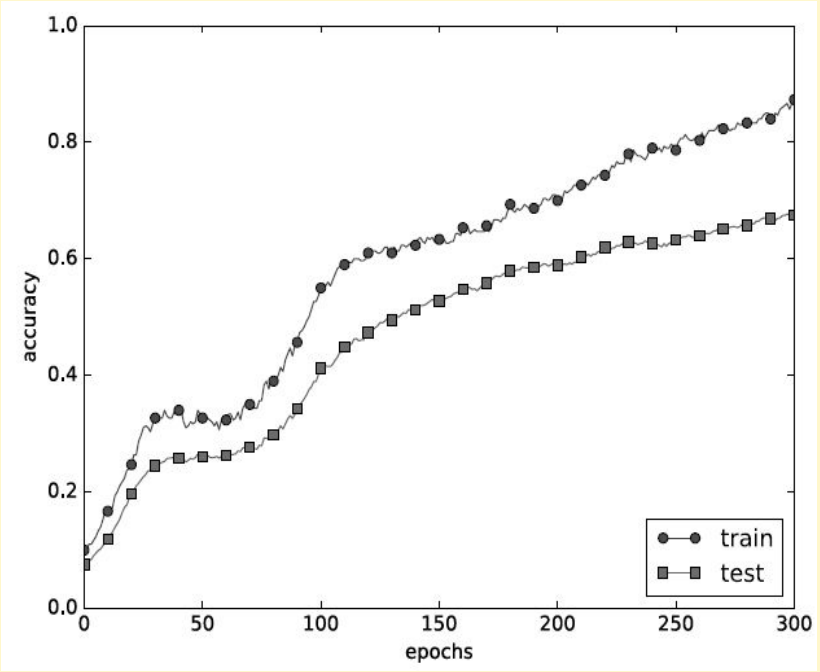
드롭아웃이란?
: 뉴런을 임의로 삭제하면서 학습하는 방법.
훈련때 은닉층의 뉴런을 무작위로 골라 삭제.
시험때는 모든 뉴런에 신호를 전달.
(뉴런의 출력에 훈련 때 삭제한 비율을 곱하여 출력)

6.4 바른 학습을 위해

- 드롭아웃

```
Class Dropout:
  def __init__(self, dropout_ratop=0.5):
    self.dropout_ratio = dropout_ratio
    self.mask = None

  def forward(self, x, train_flag=True):
    if train_flg:
      self.mask = np.random.rand(*x.shape) > self.dropout_ratio
      return x*self.mask
    else:
      return x*(1.0-self.dropout_ratio)

  def backward(self, dout):
    return dout*self.mask
```

훈련 시에는 순전파 때마다 self.mask에 삭제할 뉴런을 False로 표시한다. self.mask는 x와 형상이 같은 배열을 무작위로 생성하고 그 값이 dropout_ratio보다 큰 원소만 true로 설정.



드롭아웃 적용한 결과
dropout_ratio=0.15

6.5 적절한 하이퍼파라미터 값 찾기

현재까진 데이터셋을 훈련데이터와 테스트 데이터로 나눠서 이용.
여기서 하이퍼 파라미터의 성능을 평가할때는 시험 데이터를 사용하면 안된다. -> 오버피팅! -> 범용성능↓

- 검증 데이터(하이퍼파라미터의 적절성을 평가하는 데이터)를 사용해 하이퍼파라미터 최적화

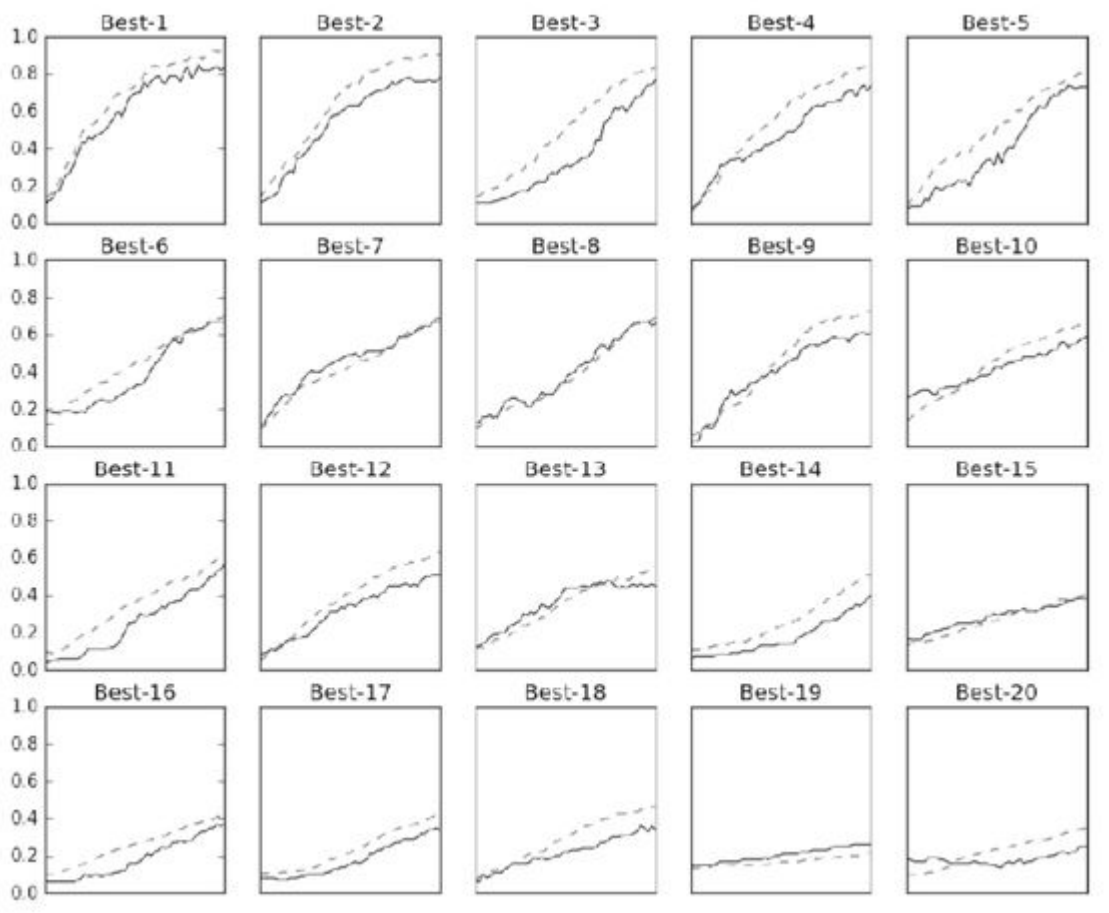
  하이퍼파라미터를 최적화할 때의 핵심은 최적값이 존재하는 **범위**를 조금씩 줄여간다는 것이다.

하이퍼파라미터 최적화 순서

1. 하이퍼파라미터 값의 범위를 설정한다. (대략적으로)
2. 설정된 범위에서 하이퍼파라미터의 값을 무작위로 추출한다.
3. 1단계에서 샘플링한 하이퍼파라미터 값을 사용하여 학습하고, 검증데이터로 정확도를 평가한다. (에폭은 작게 설정)
4. 1단계와 2단계를 특정 횟수(100회 등) 반복하며, 그 정확도 결과를 보고 하이퍼파라미터의 범위를 좁힌다.

6.5 적절한 하이퍼파라미터 값 찾기

ex) weight_decay = 10**np.random.uniform(-8, -4) -> (-8,-4)범위 안에서 무작위하게 선정
lr = 10**np.random.uniform(-6, -2) -> (-6,-2) 범위 안에서 무작위하게 선정



← 검증 데이터의 학습 추이를
정확도가
높은 순서로 나열한 것

```
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
```

Best-5 까지의
하이퍼파라미터 값

6.6 정리

- 이번 장에서 배운 것

1. 매개변수 갱신 방법에는 확률적 경사 하강법 외에도 모멘텀, AdaGrad, Adam
2. 가중치 초깃값을 정하는 방법은 올바른 학습을 하는 데 매우 중요하다.
3. 가중치의 초깃값으로는 Xavier 초깃값과 He 초깃값이 효과적이다.
4. 배치 정규화를 이용하면 학습을 빠르게 진행할 수 있으며, 초깃값에 영향을 덜 받게된다.
5. 오버피팅을 억제하는 정규화 기술로는 가중치 감소와 드롭아웃이 있다.
6. 하이퍼파라미터 값 탐색은 최적 값이 존재할 법한 범위를 점차 좁히면서 하는 것이 효과적이다.

감사합니다
:)