chapter 6 학습 관련 기술들 : 패키지 사용시 개념 제공

6.1 매개변수 갱신 : W, B

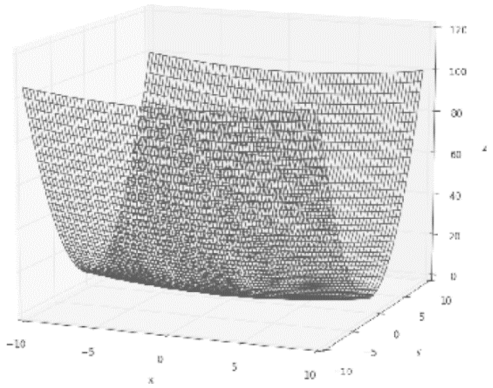
손실함수값을 최소화하는 매개변수 탐색, 최적화 optimization

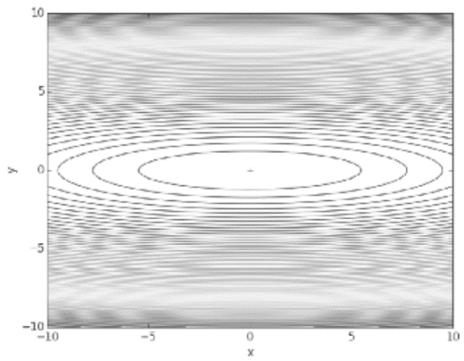
6.1.2 확률적 경사 하강법 (SGD)

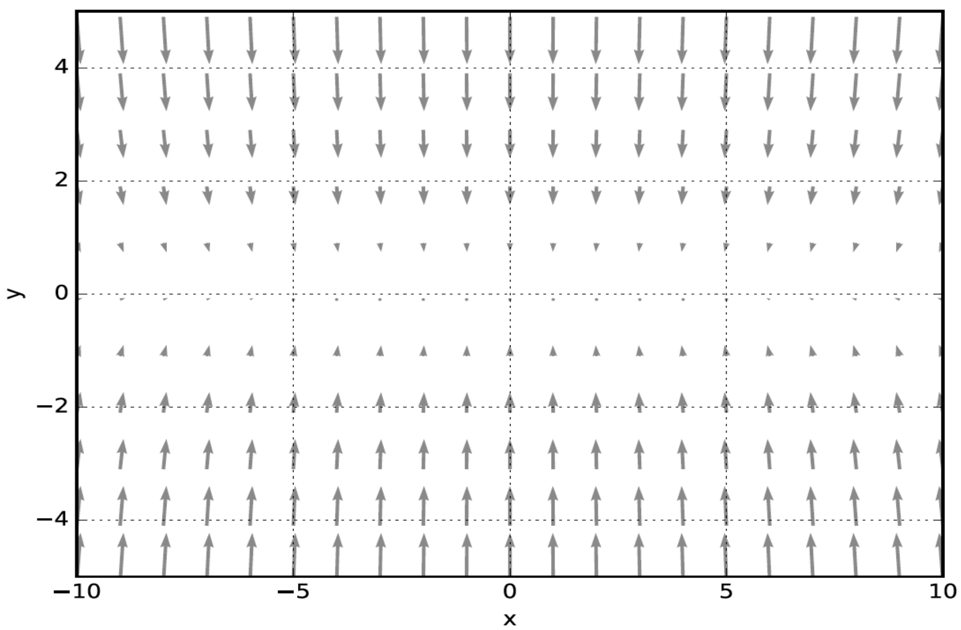
$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

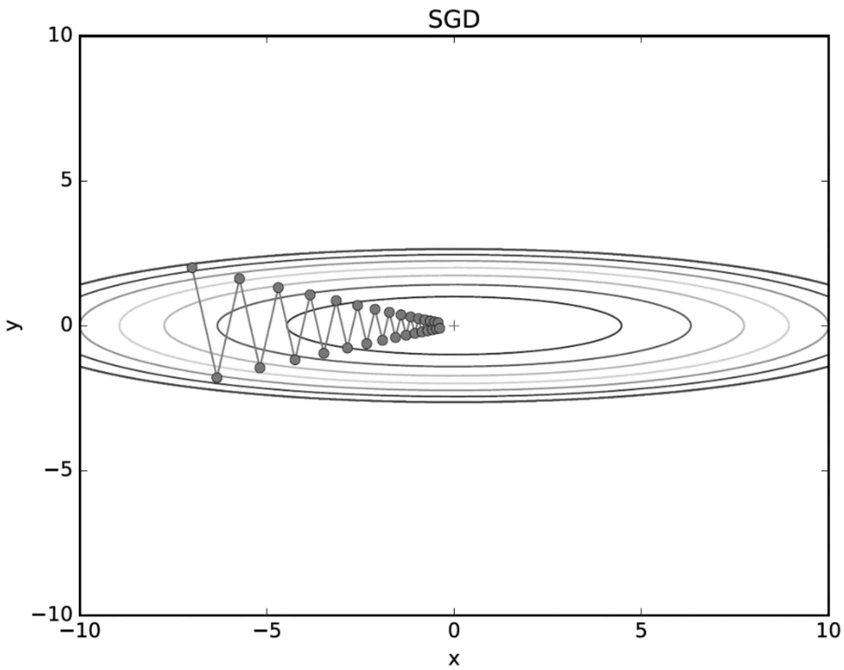
6.1.3 SGD의 단점

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







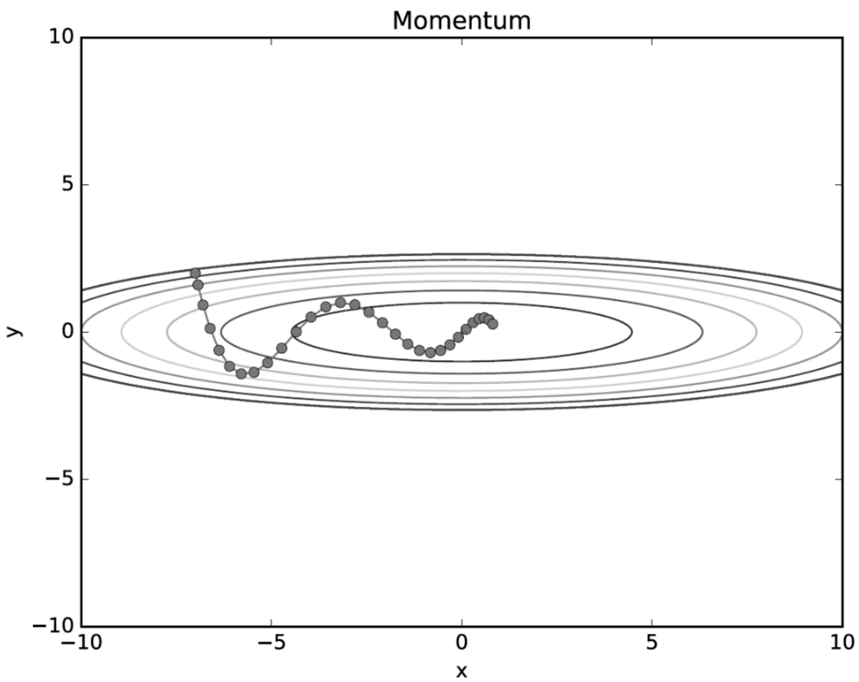


6.1.4 모멘텀: momemtum, 운동량

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

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

common/optimizer.py 의 Momentum class



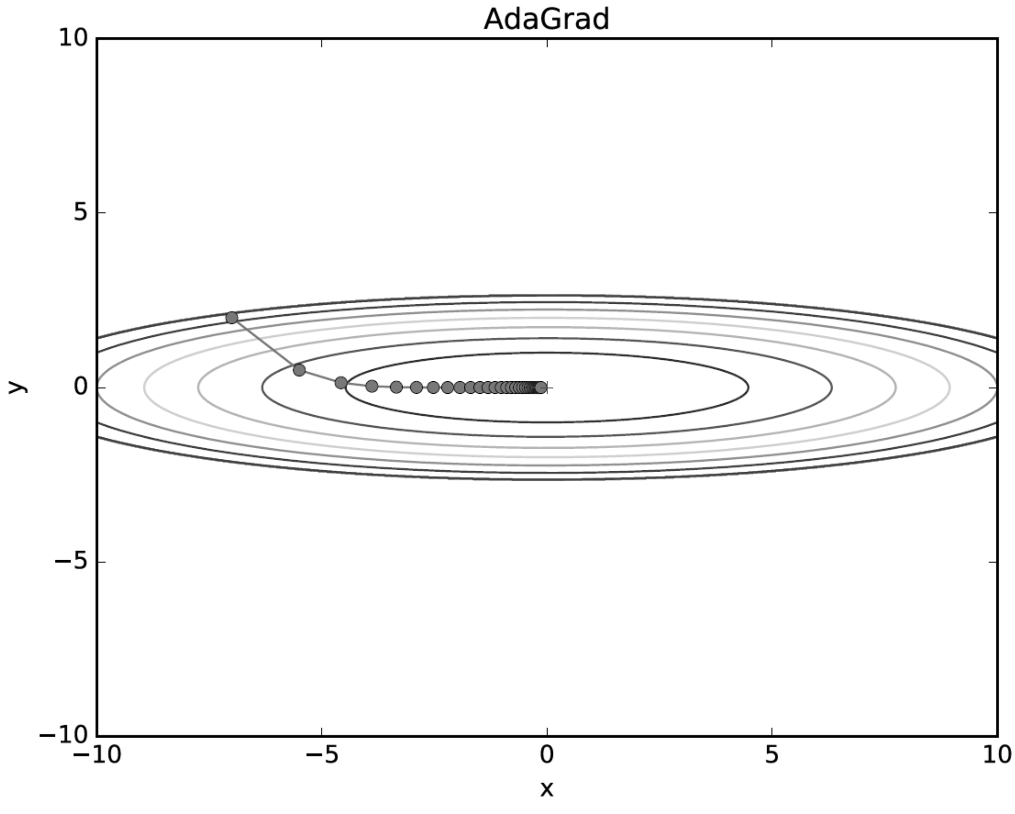
6.1.5 AdaGrad

학습을 진행하면서 학습률 감소

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

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial L}{\partial \mathbf{W}}$$
: 누적된 값으로 나누므로 점점 작아짐

common/optimizer.py 의 AdaGrad class

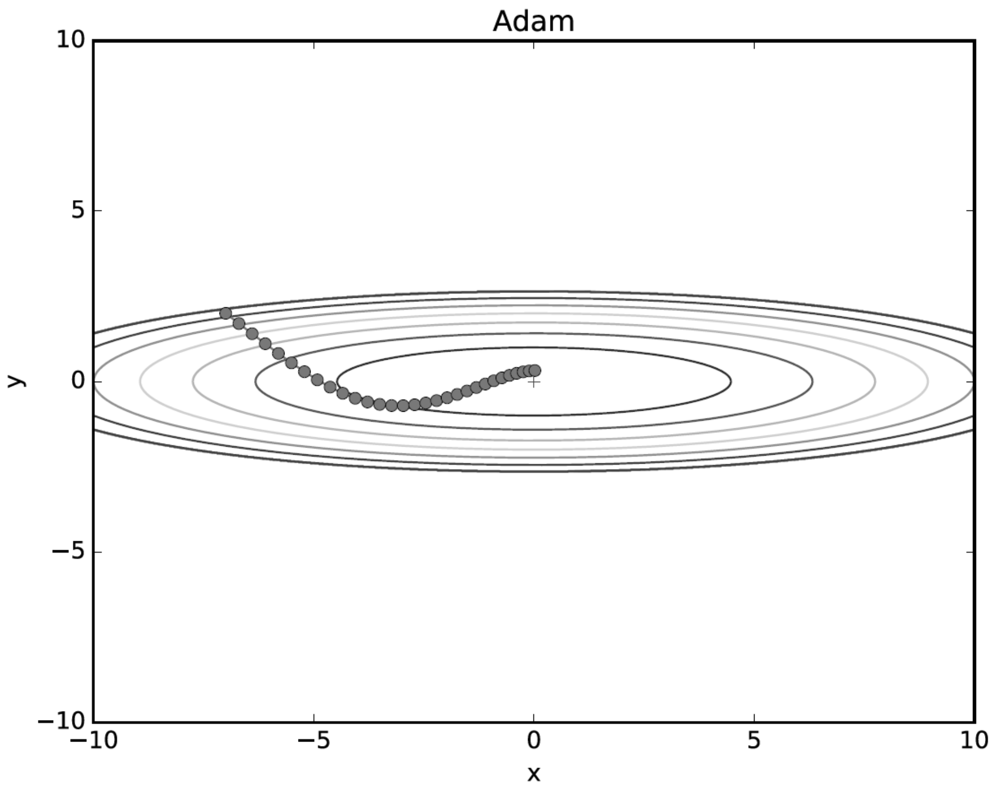


움직임이 점차 작아지고, 지그재그 움직임도 줄어듬

6.1.6 Adam

모멘텀 방법의 이동 성질과 AdaGrad의 학습률 감소 hybrid

common/optimizer.py 의 Adam class



6.1.7 어느 갱신 방법을 이용할 것인가?

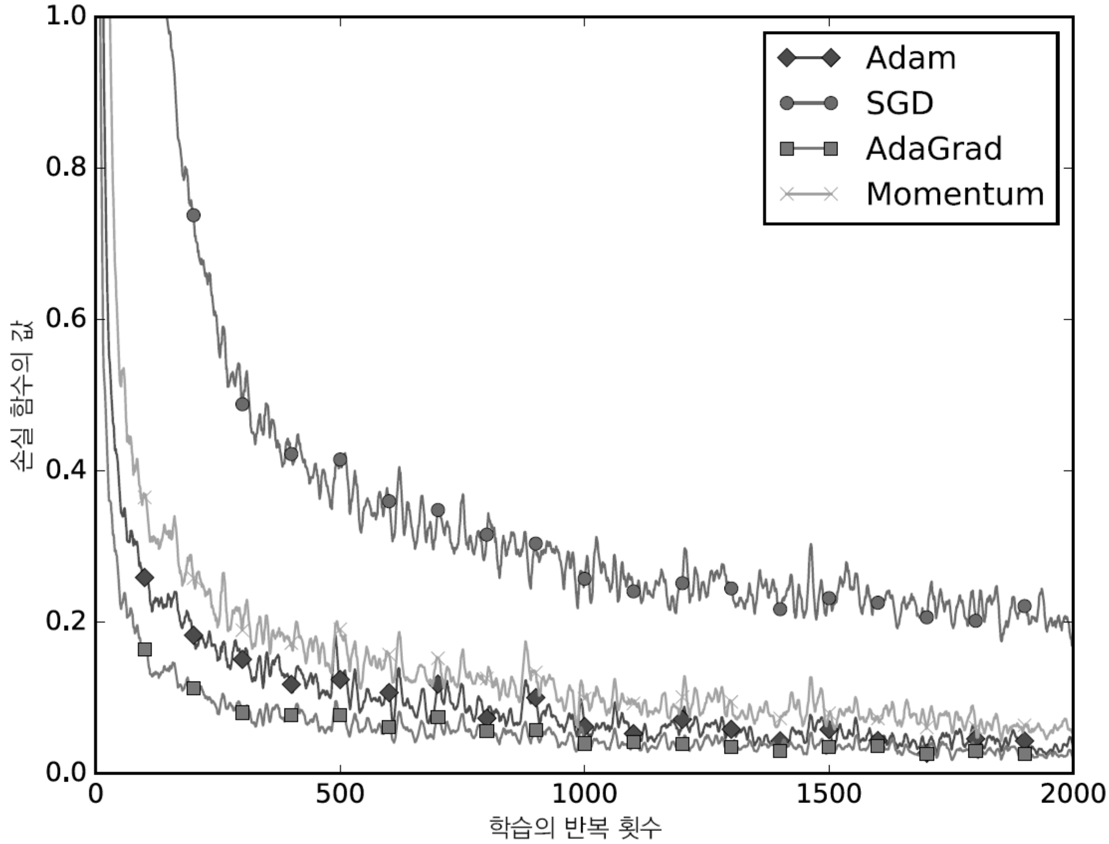
우리 그림들에서는 AdaGrad가 우수한 것처럼 보임

문제마다 다른 성능을 보임

많은 경우 Adam이 우수함

6.1.8 MINIST 데이터셋으로 본 갱신 방법 비교

ch06/optimizer_compare_mnist.py



2022. 4. 13

6.2 가중치의 초기값 : 시작이 중요

6.2.1 초깃값을 0으로 하면?

오버피팅 억제위해 가중치 감소 (L1, L2 제한)

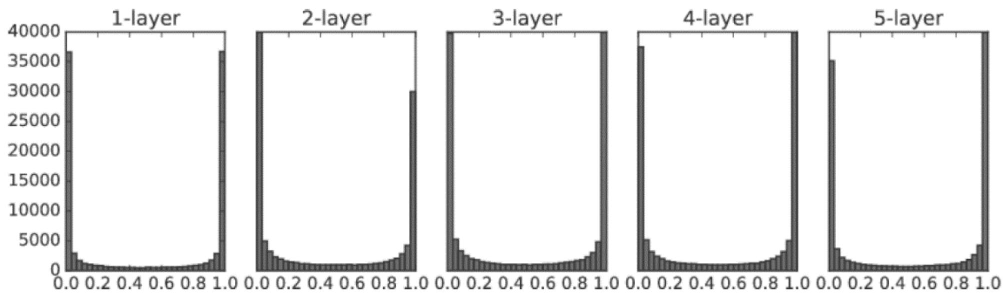
모든 가중치 0 초기화 → 다음 층 뉴런에 같은 값 전달 → 같

은 값으로 역전파 (곱셈 노드 예) → 가중치 갱신 안됨

6.2.2 은닉층의 활성화값 분포

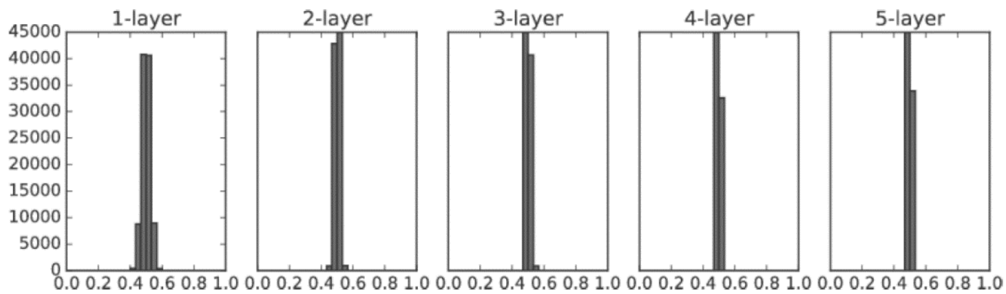
가중값 초기화에 따른 은닉층 값 변화

ch06/weight_init_activation_histogram.py



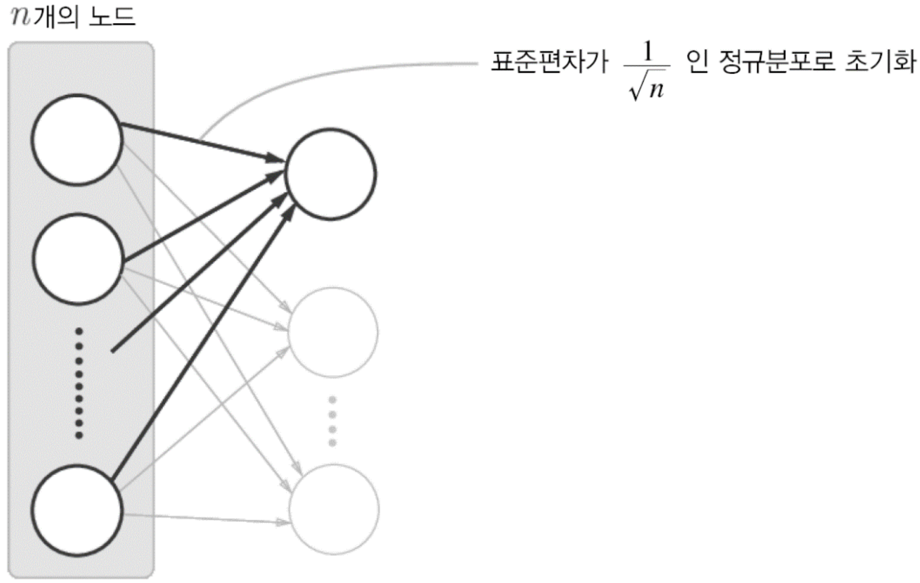
표준편차 1 정규분포로 초기화

시그모이드 값이 0과 1에 집중 → 시그모이드 기울기는 0 → 역전파로 학습이 안됨 ← 기울기소실(gradient vanishing) 문제

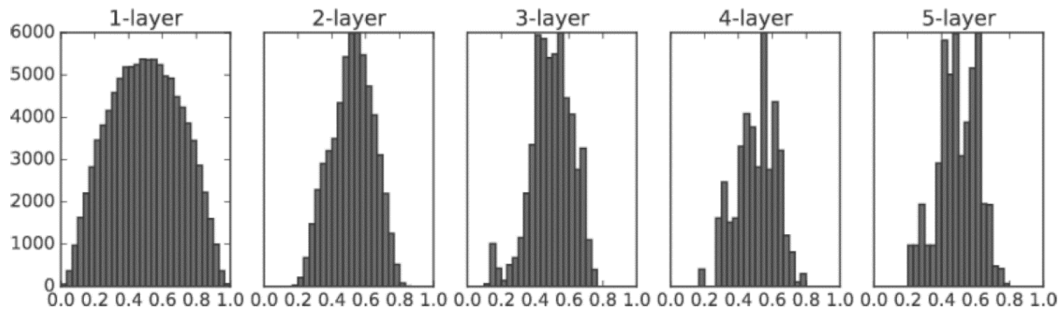


표준편차 0.01 정규분포로 초기화

시그모이드 값이 0.5 집중, 같은 값 출력 → 신경망의 표현력이 부족 → 여러 뉴런 사용 효력 감소로 신경망 성능저하



Xavier 초기값 : 앞층 노드의 수가 많을수록 작은 초기값

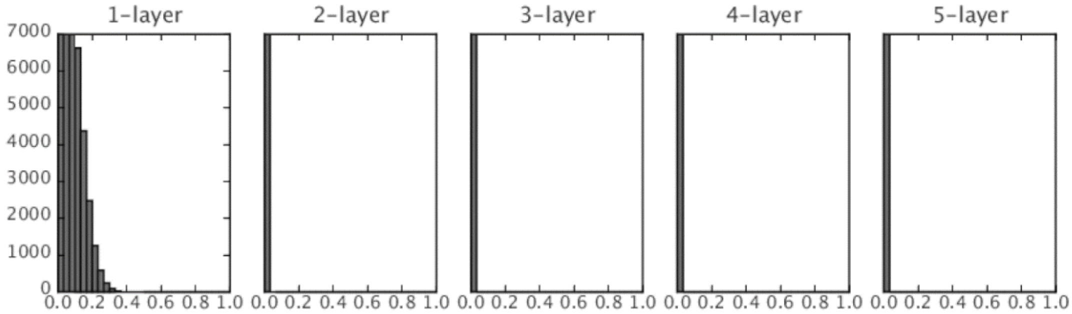


앞 예들보다 넓은 분포, 가중치감소/표현력저하 극복

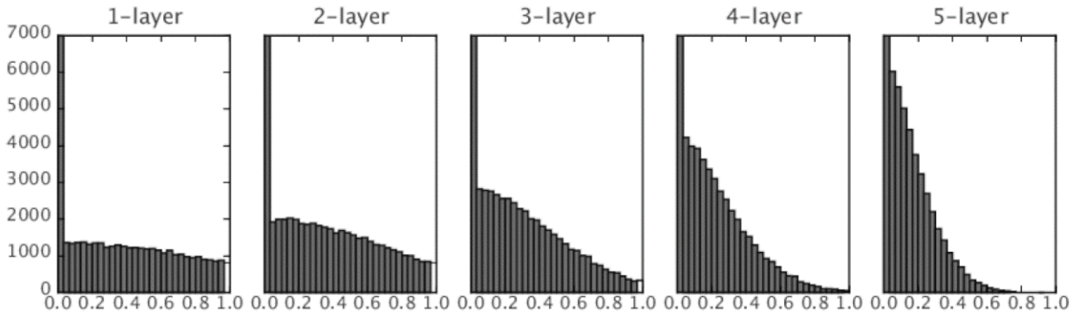
6.2.3 ReLU를 사용할 때의 가중치 초기값

Xavier는 중앙부근이 선형이 함수에 적당

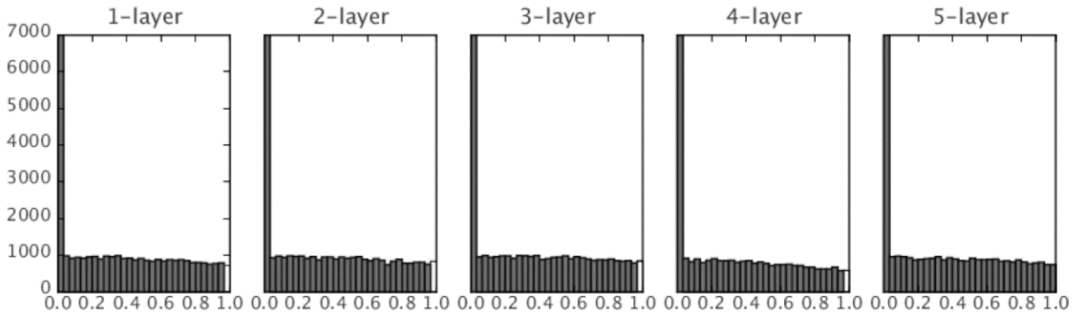
ReLU의 경우에는 He 초기값 사용 $\sqrt{2/n}$ ← 음의 영역이 0인 ReLU 를 위해 더 넓은 초기값 사용



표준편차가 0.01인 정규분포를 가중치 초깃값으로 사용한 경우



Xavier 초깃값을 사용한 경우



He 초깃값을 사용한 경우

위부터 0근처 집중, 점차 치우침, 0 이외 부분도 값 존재

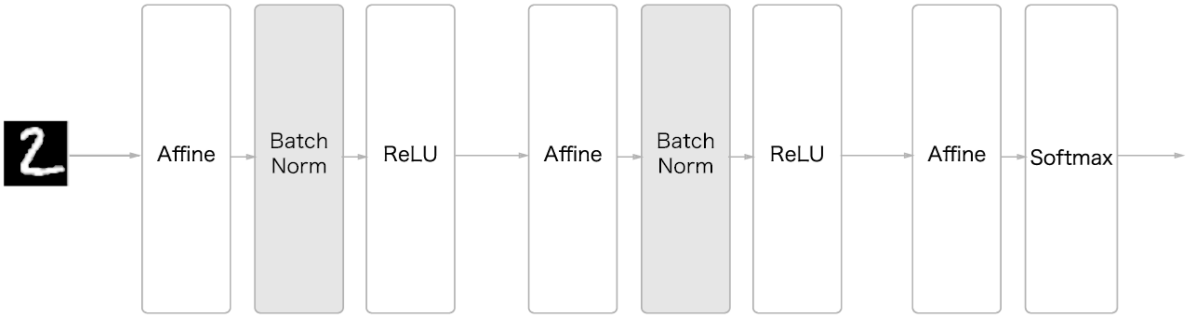
6.2.4 MINIST 데이터셋으로 본 가중치 초깃값 비교

ch06/weight_init_compare.py 읽기와 실행결과 → 앞장의 예측과 비슷한 결과를 보임

6.3 배치 정규화 (Batch Normalization)

활성화값 분포를 균등하게 퍼지게 함

- 학습 속도 개선
- 초기값에 의존도 감소
- 오버피팅 억제 (드롭아웃 필요성 감소)



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

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

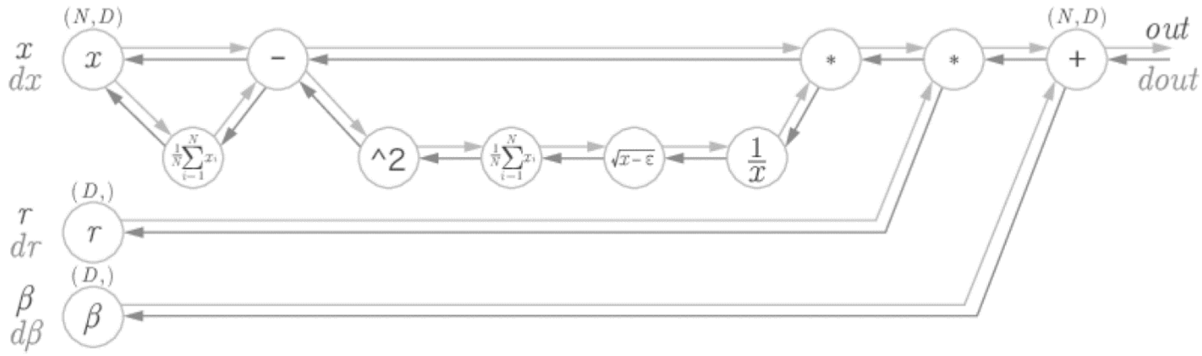
$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

평균 0, 분산 1로 정규화

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

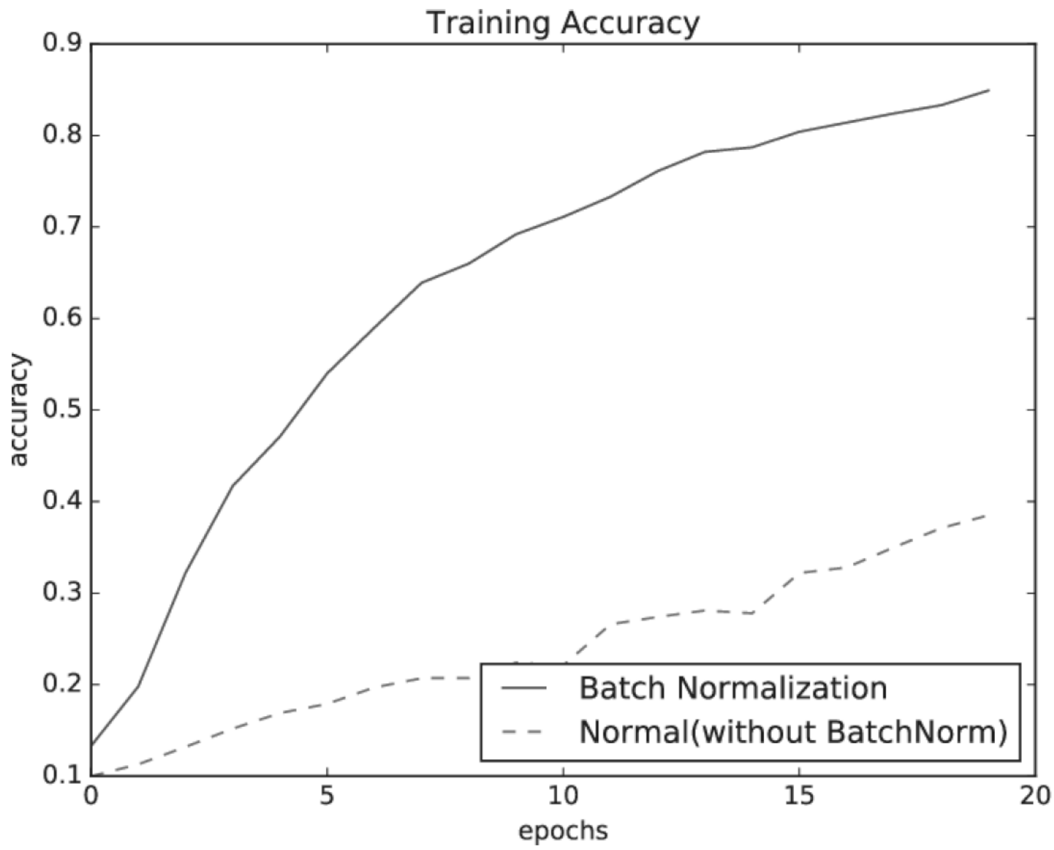
확대와 이동,

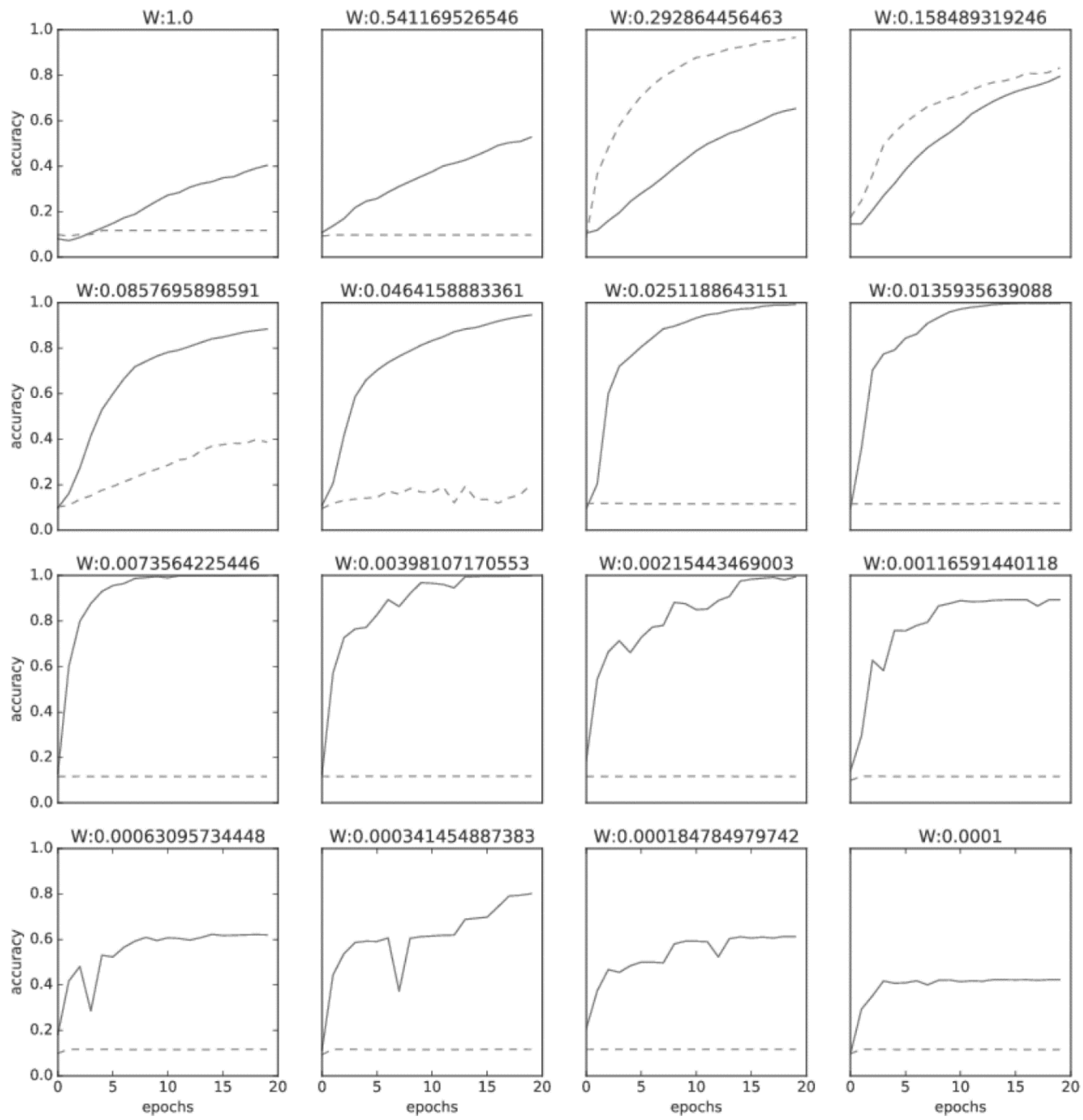
배치과정의 계산 그래프와 역전파 : 설명은 Frederik Kratzenrt 블로그 참조



6.3.2 배치 정규화의 효과

ch06/batch_norm_test.py





배치정규화가 없는 경우에 초기값이 적절한 구간에서만 학습되는 반면 배치정규화를 하면 대부분의 경우에 학습이 됨

6.4 바른 학습을 위해

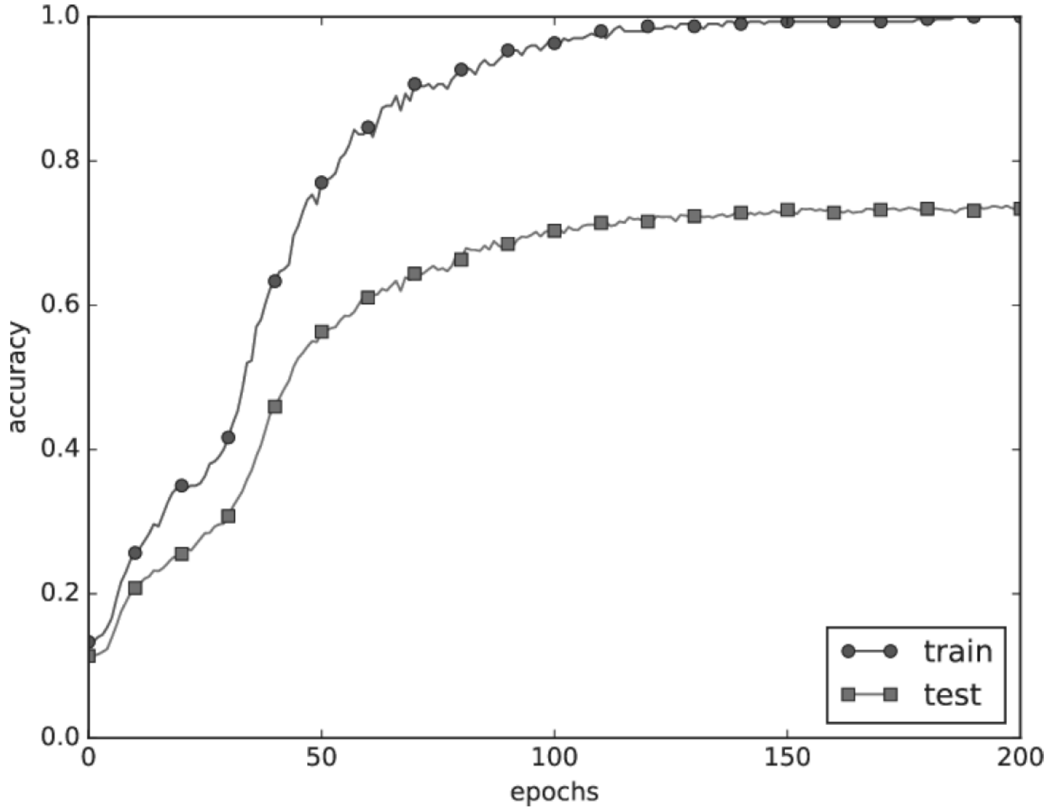
6.4.1 오버피팅 (overfitting: 과적합)

-매개변수가 많고 표현력이 높은 모델

- 훈련데이터가 적음

훈련 데이터는 300개만, 7층의 100 뉴런을 갖는 오버피팅 유도모델

ch06/overfit_weight_decay.py 실행후 train과 test의 정확도차이 큼



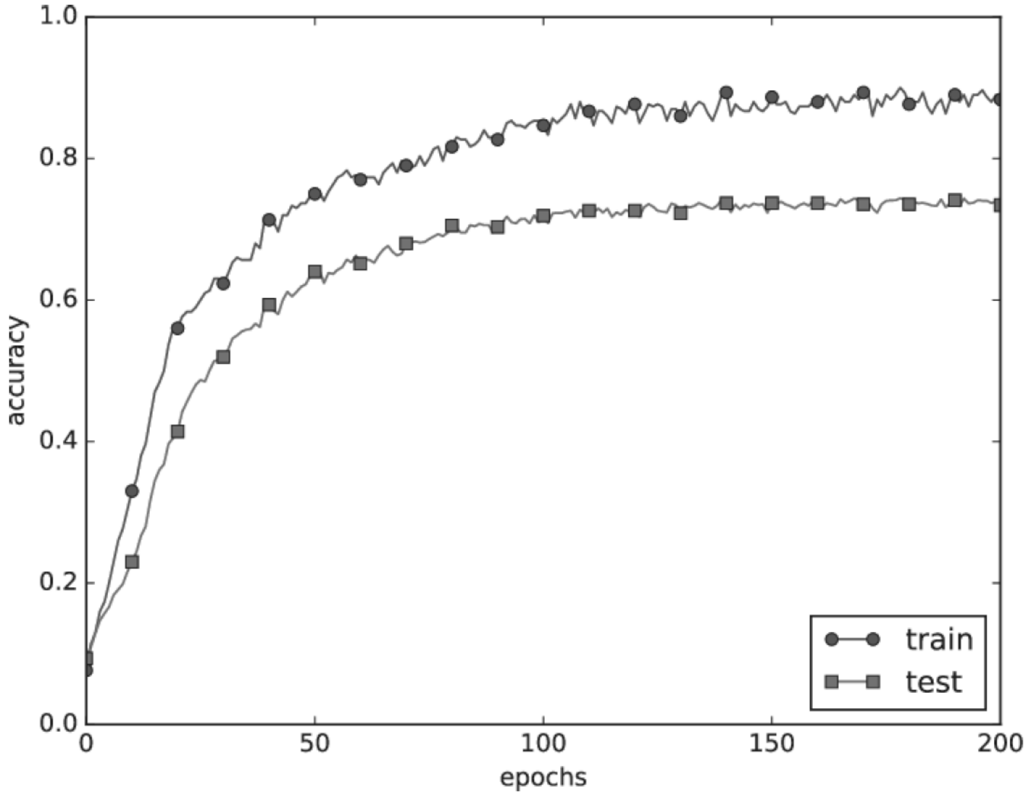
6.4.2 가중치 감소

손실함수에 $\frac{1}{2}\lambda W^2$를 추가

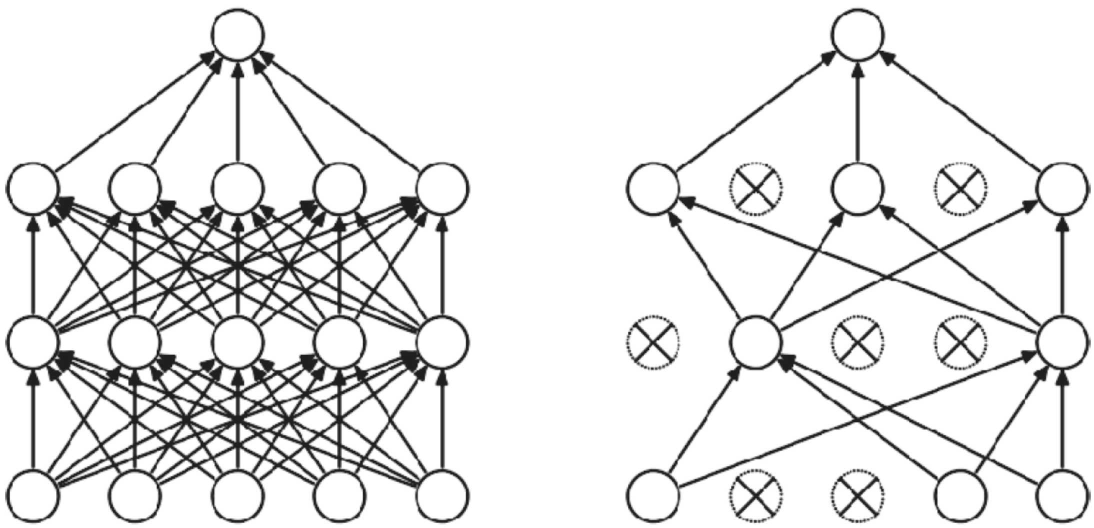
weight_decay += 0.5 * self.weight_decay_lambda * np.sum(W ** 2)

오차역전파에 미분한 λW를 더함

grads['W' + str(idx)] = self.layers['Affine' + str(idx)].dW + self.weight_decay_lambda * self.layers['Affine' + str(idx)].W common/multi_layer_net.py 에 구현



6.4.3 드롭아웃: 오버피팅 억제

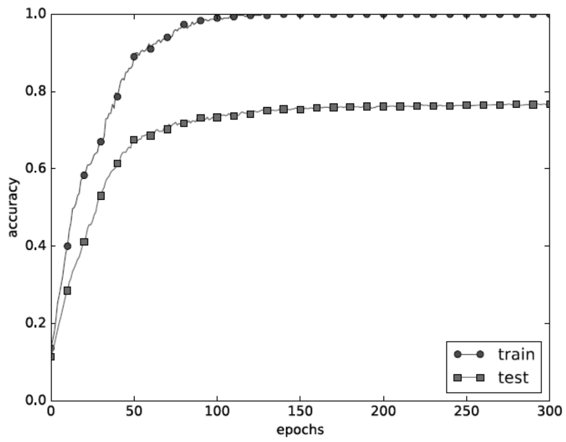


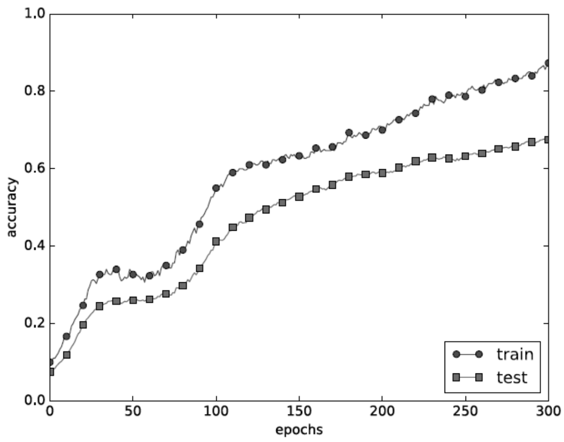
(a) 일반 신경망

(b) 드롭아웃을 적용한 신경망

훈련때 train_flag 에 유의

ReLU 와 같은 mask 기술 : common/layers.py: class Dropout 읽기





학습과 평가 정확도는 오른쪽이 줄어듬

그러나, 평가 정확도가 감소하였으므로, 결과적으로는 역효과

** 앙상블 학습 : 여러 모델을 사용하여 평균 이용

드롭아웃은 매번 다른 모델이 학습되는 효과 → 유사 앙상블 학습

6.5 적절한 하이퍼파라미터 hyper parameter 값 찾기

학습으로 결정할 수 없는 파라미터: 각 층의 뉴런 수, 배치 크기, 학습률, 가중치 감소, 신경망 구조(배치 정규화, 드롭아웃), 활성화 함수

6.5.1 검증 데이터

하이퍼 파라미터 검증 데이터:

test 데이터를 이 용도로 사용하면 하이퍼 파라미터가 test 데이터에 적합하게 조정되므로 범용 성능 측정 불가

훈련데이터: 매개변수 학습

검증데이터: 하이퍼 파라미터 최적 선택

시험데이터: 신경망의 범용 성능 평가

common/util.py 검증데이터 구성 shuffle_dataset(x, t) 이용

6.5.2 하이퍼파라미터 최적화 : 실험자의 지혜와 직관, working knowledge 중요 (sw의 직업)

hyper parameter 를 0.001 에서 1000 사이: 10의 계승으로 탐색 또는 무작위 추출

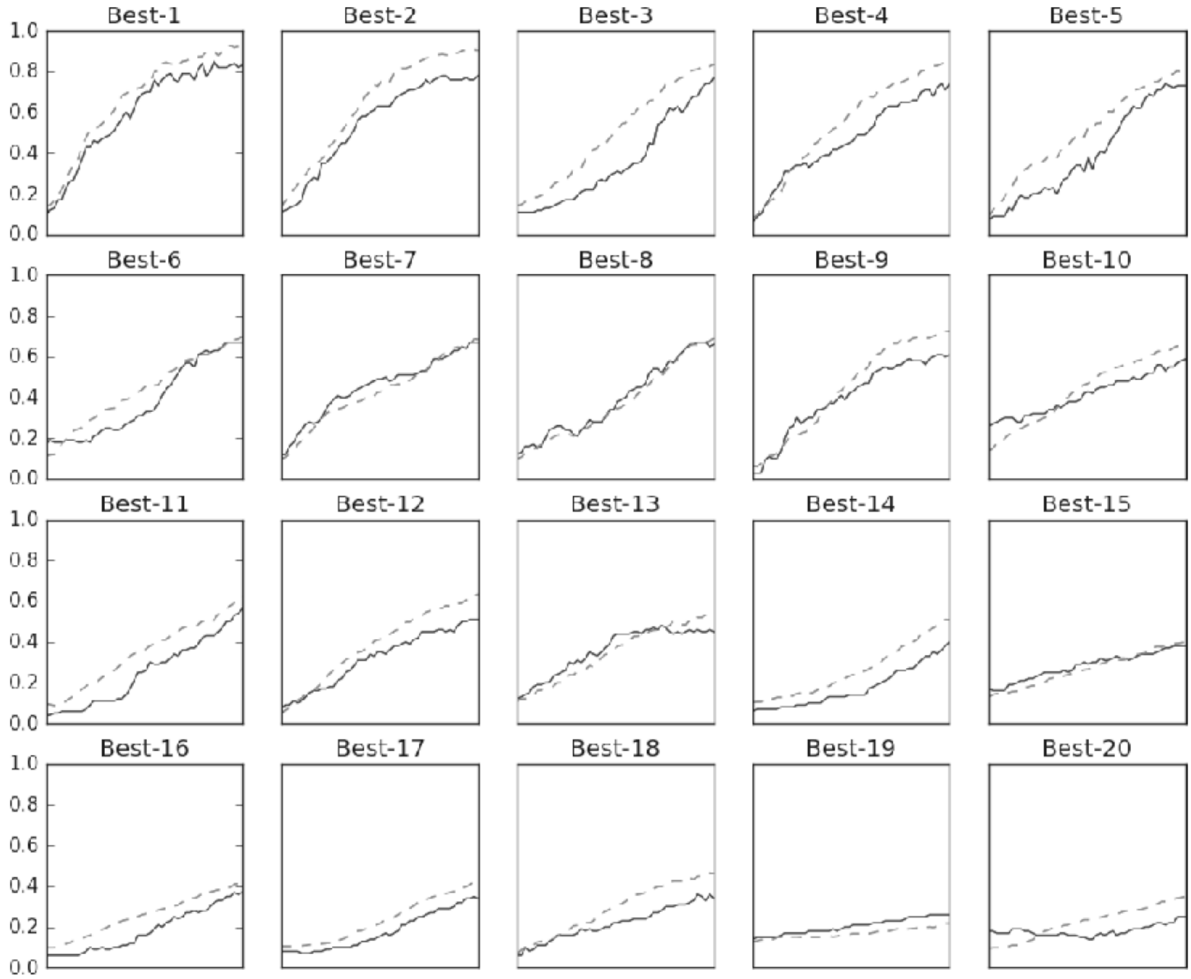
학습 반복 회수를 작게 하여 빠르게 실험 진행

6.5.3 하이퍼 파라미터 최적화 구현하기

학습률과 가중치 감소 최적화 예

```
    # 탐색한 하이퍼 파라미터의 범위 지정================
        weight_decay = 10 ** np.random.uniform(-8, -4)
```

lr = 10 ** np.random.uniform(-6, -2)



Best-1(val acc:0.8) | lr:0.008577586341744887, weight decay:3.775067339785634e-05

Best-2(val acc:0.76) | lr:0.009972315777645619, weight decay:1.7790345069747974e-07

Best-3(val acc:0.7) | lr:0.005173499210694182, weight decay:6.0478829643480846e-06

Best-4(val acc:0.7) | lr:0.005408544309264094, weight decay:2.406100296246451e-08

Best-5(val acc:0.63) | lr:0.004115447996094075, weight decay:5.2933876790196986e-08