Hyperparameter Optimization using Python

Hee-il Hahn

Professor
Department of Information and Communications Engineering
Hankuk University of Foreign Studies
hihahn@hufs.ac.kr

Weight Vector Update

- Neuralnet Learning의 목적
  - Loss function의 값을 가능한 낮추는 weight vector를 찾는다.
  - 즉, weight vector의 최적값을 찾는 문제이다. (이를 optimization이라 부른다.)
  - Weight vector space는 매우 넓고 복잡해서 최적의 솔류션은 쉽게 찾기 어렵다.
  - 특히, deep neural network에서는 weight vector의 수가 업청나게 많아져서 더욱 어렵다.
  - 현재로서는 weight vector에 대한 loss function의 gradient를 구해 기울어진 방향으로 weight vector를 반복적으로 update하여 최적 값에 접근하고자 한다.

Weight Vector Update – cont.

- SGD(stochastic gradient descent method)
  - 지금까지 우리가 배운 gradient descent method로서 다음과 같이 weight vector를 update한다.

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

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
$k \leftarrow 1$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
$k \leftarrow k + 1$
end while

Weight Vector Update – cont.

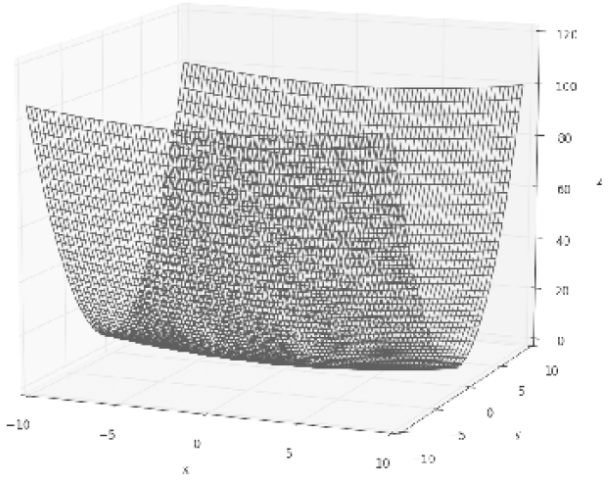
- SGD의 단점
  - SGD는 단순하고 구현하기 쉽지만, 문제에 따라서는 비효율적일 때가 있다.
  - 예를 들어 다음 함수의 최소값을 구하는 문제에서,

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

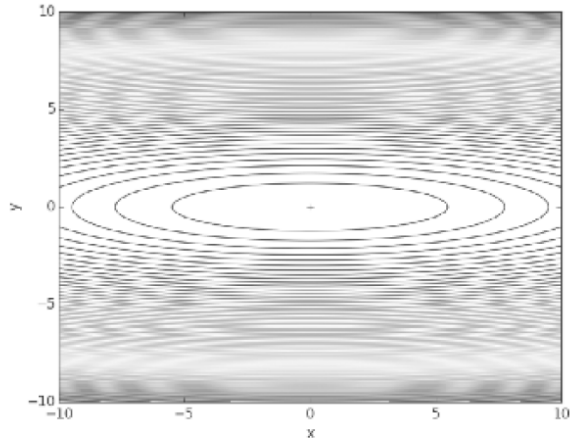
  - 이 함수의 gradient vector는 y축 방향으로는 크고 x축 방향으로는 작다.

Weight Vector Update – cont.

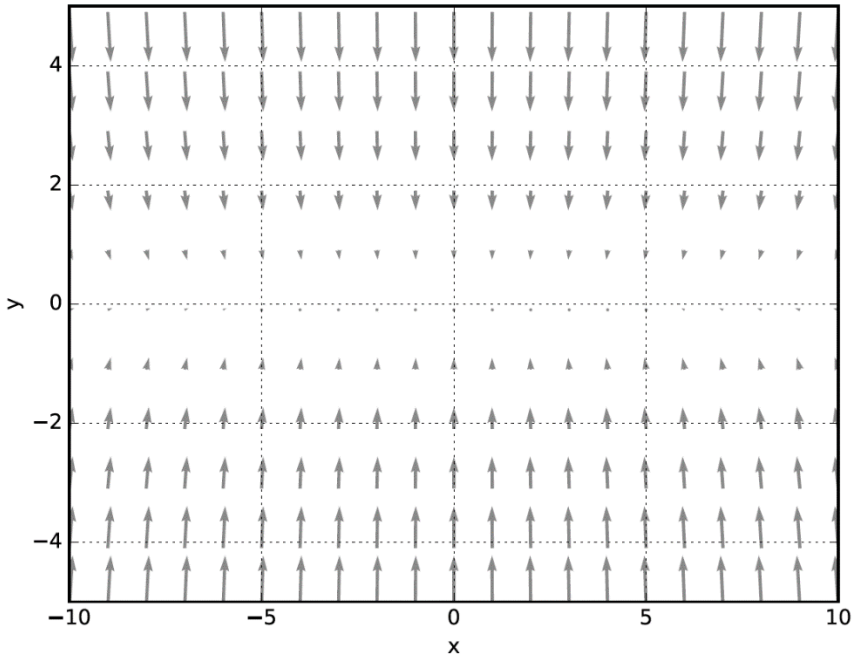
- SGD의 단점 - 계속



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



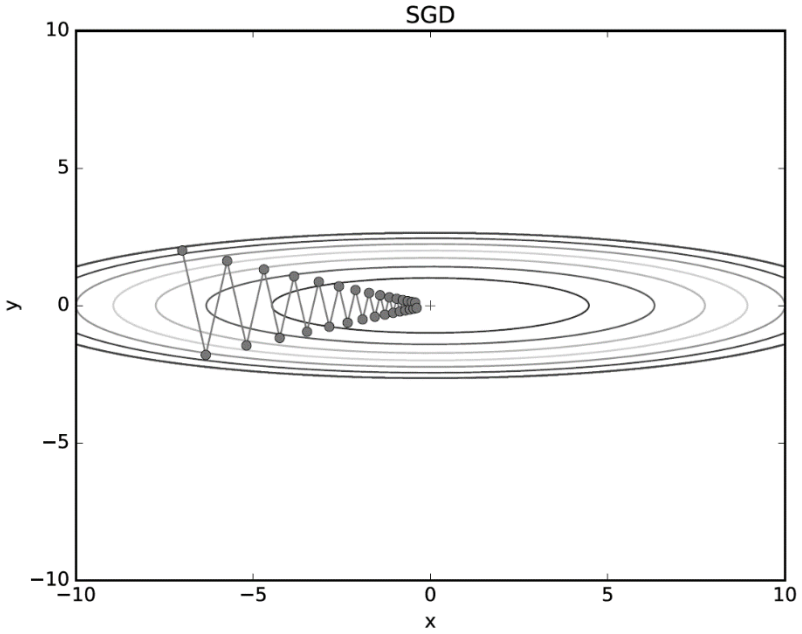
등고선



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

Weight Vector Update – cont.

- SGD의 단점 - 계속
  - 초기값 $(x, y) = (-7, 2)$에서 알고리즘을 수행하면 최소값인 $(0, 0)$까지 지그재그로 이동하므로 비효율적이다.
  - 즉, 방향에 따라 기울기가 달라지는 함수(anisotropy function)에서는 탐색경로가 비효율적이다.
  - 이러한 단점을 개선하기 위해 Momentum, AdaGrad, Adam 등이 제안된다.



SGD에 의한 최적화 update 경로

$$f\big(x(t), y(t)\big) = c$$

$$\rightarrow \frac{df}{dt} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} \rightarrow \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \cdot \left(\frac{dx}{dt}, \frac{dy}{dt}\right) = 0$$

Weight Vector Update – cont.

- Momentum
  - Momentum 기법은 다음과 같이 weight vector를 update시킨다.

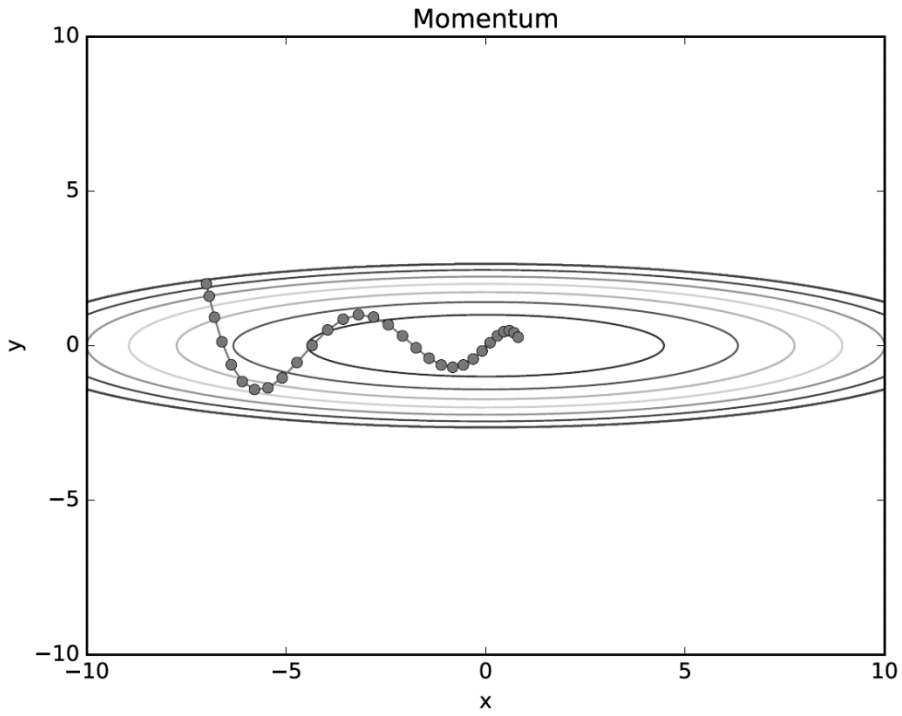
$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

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

  - IIR Filter

$$y[n] - \alpha y[n-1] = x[n]$$

  - x축의 힘은 아주 작지만 방향은 변하지 않아서 한방향으로 일정하게 진행한다.
  - SGD보다 x축 방향으로 빠르게 접근한다.



Momentum에 의한 최적화 update 경로

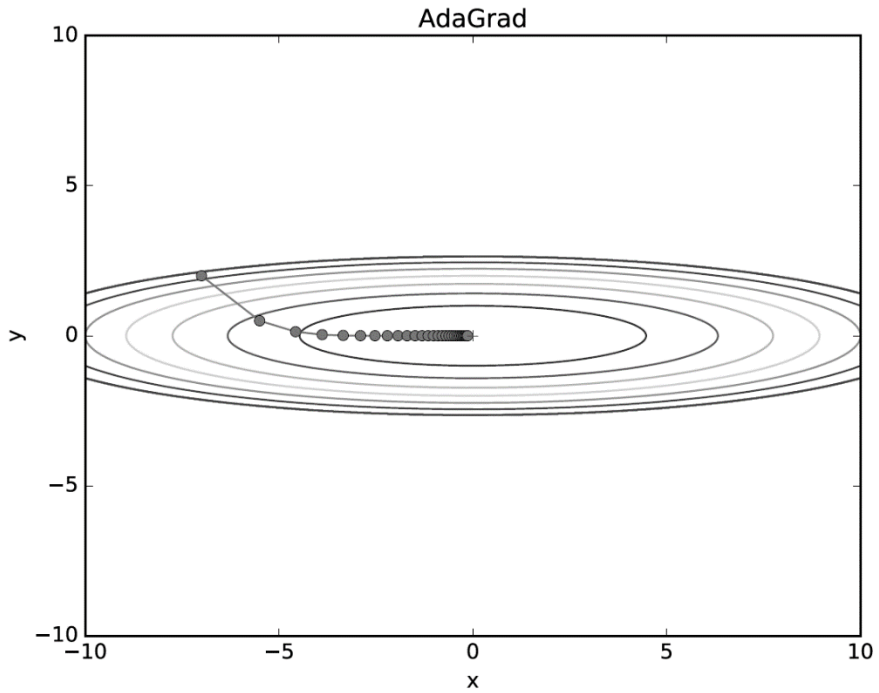
Weight Vector Update – cont.

- AdaGrad
  - Neuralnet에서는 learning rate가 중요하다. 작으면 학습시간이 길어지고 크면 발산할 가능성 있다.
  - AdaGrad는 학습을 진행하면서 learning rate를 점차 줄여가는 방식이다.

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

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

  - Y축 방향은 기울기가 커서 처음에는 크게 움직이지만 시간이 지남에 따라 큰 폭으로 작아진다.



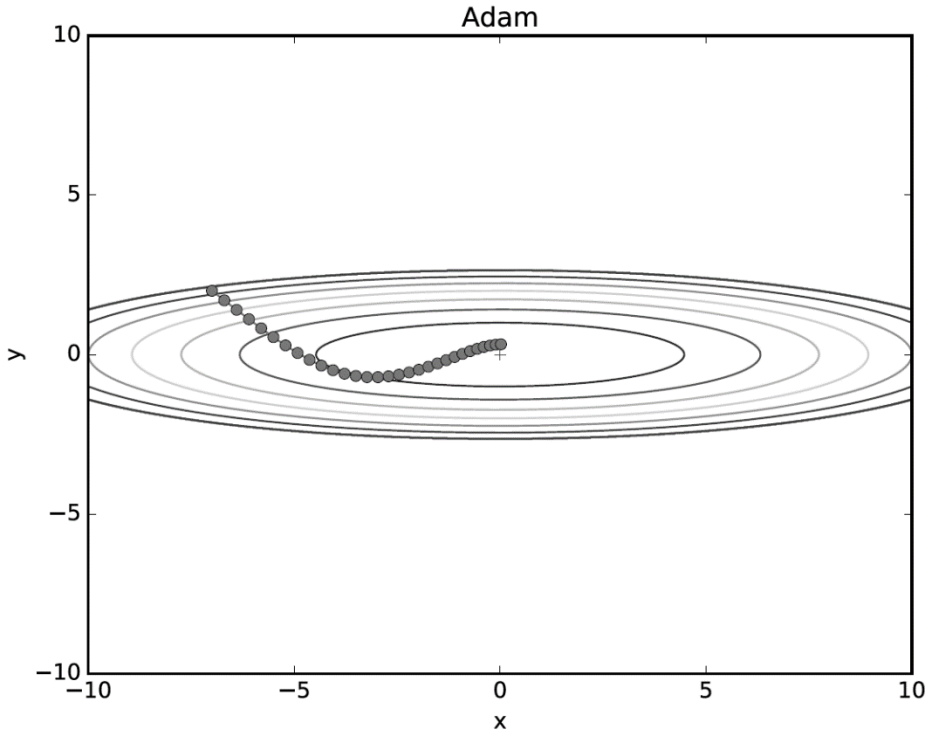
AdaGrad에 의한 최적화 update 경로

Weight Vector Update – cont.

- Adam
  - Momentum과 AdaGrad의 두 기법을 융합한 방식이다.

Algorithm 8.7 The Adam algorithm

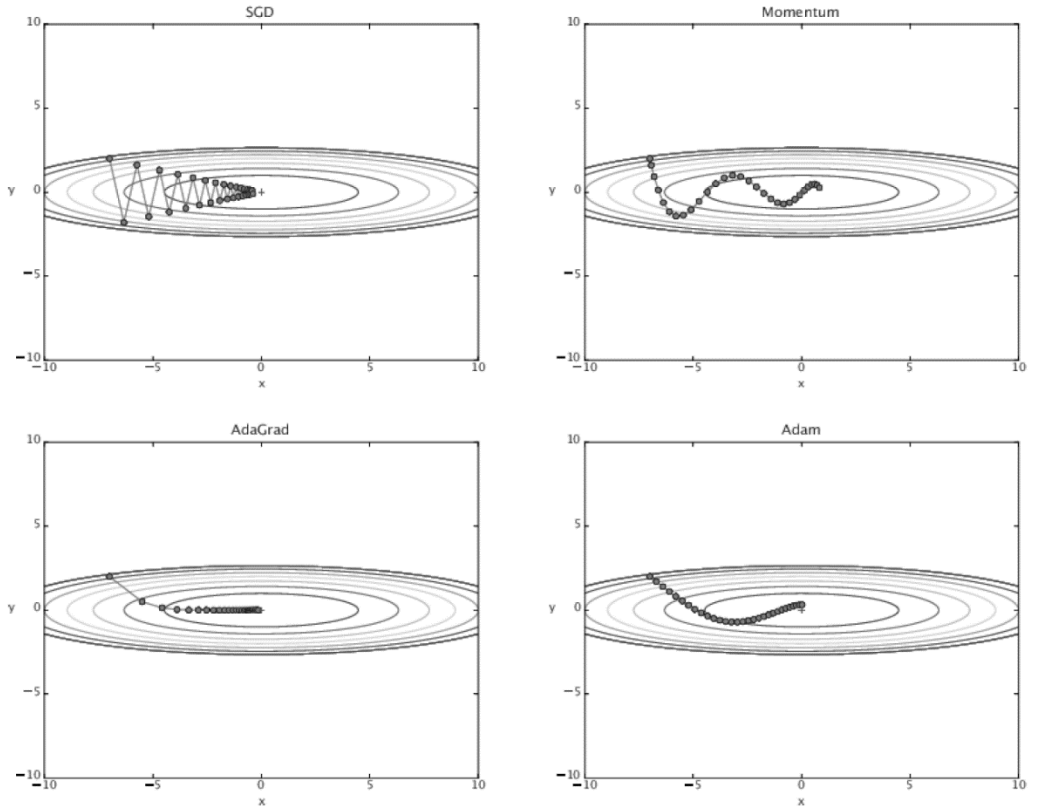
Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while



Adam에 의한 최적화 update 경로

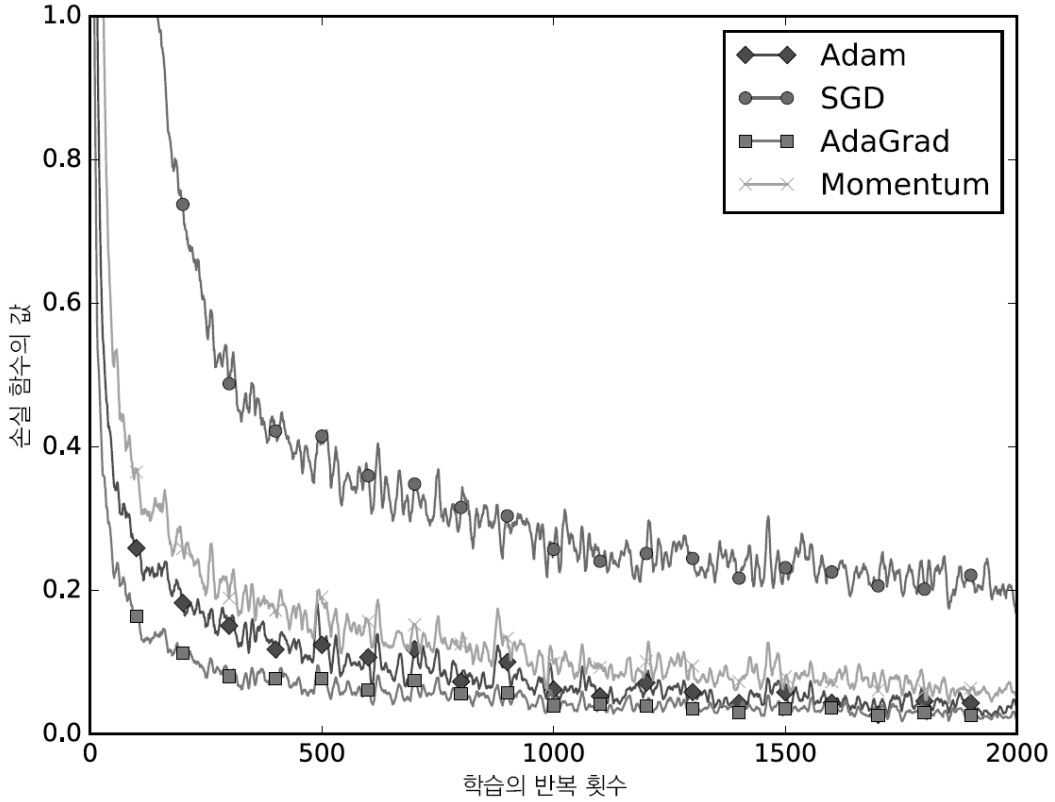
Weight Vector Update – cont.

- 어떤 방식이 제일 좋을까?
  - 응용 문제에 따라 달라지므로 주의해야 한다.
  - Hyperparameter를 어떻게 설정하느냐에 따라서도 결과가 바뀔 수 있다.
  - 모든 문제에서 항상 뛰어난 기법은 없다.
  - Adam을 많이 쓰는 경향이 있다.



Weight Vector Update – cont.

- MNIST Dataset으로 실험했을 때의 성능비교
  - 각 layer가 100개의 뉴런으로 구성된 5-layer NN에서 Relu를 activation function으로 이용했을 때의 성능비교는 다음과 같다.
  - Hyperparameter(learning rate, NN의 구조 등)에 따라 성능이 달라질 수 있다.



Weight Vector 의 초기값 설정

- 초기값을 모두 0으로 정했을 때의 문제점
  - Weight vector의 값이 작아지도록 학습하여야(weight decay) overfitting을 방지할 수 있다..
  - Weight vector의 초기값을 모두 0(또는 동일한 값)으로 설정하면 학습이 올바로 이루어지지 않는다.
  - 즉, back propagation에서 모든 weight vector가 똑같이 update되기 때문에 weight vector를 여러 개 갖는 의미가 사라진다.
  - 이러한 문제를 피하기 위해서는 초기값을 random하게 설정해야 한다.

- Hidden layer의 Activation function 값의 분포 분석 실험

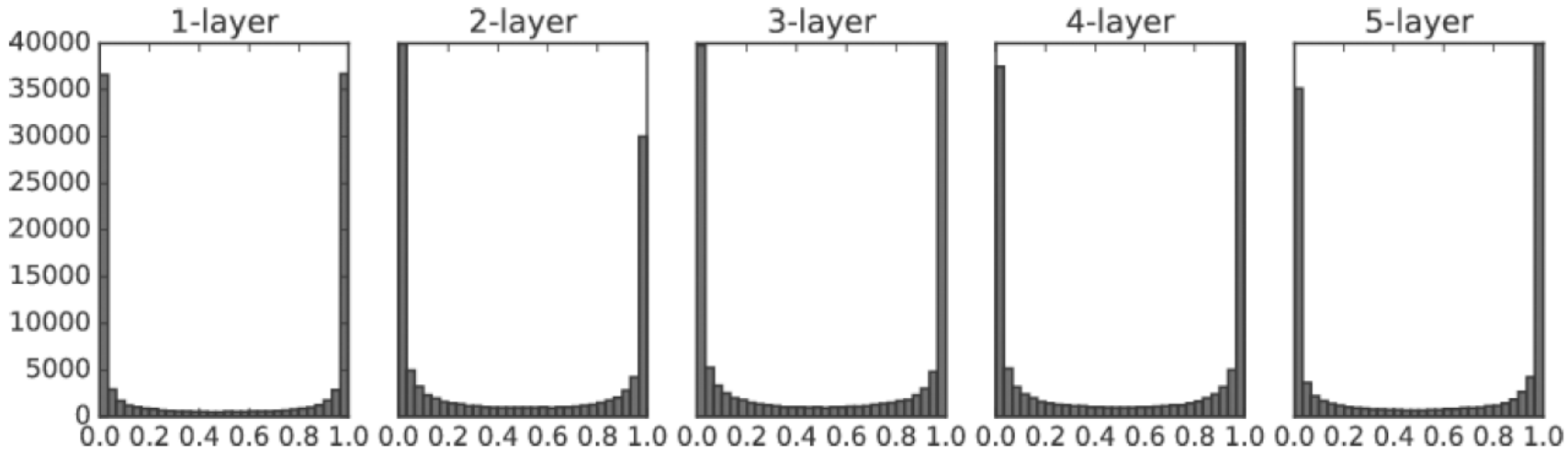
  실험환경 : sigmoid를 activation function으로 사용하는 5-layer NN

  각 layer의 노드는 100개씩,

  입력 데이터는 1000개의 데이터를 정규분포로 random하게 생성

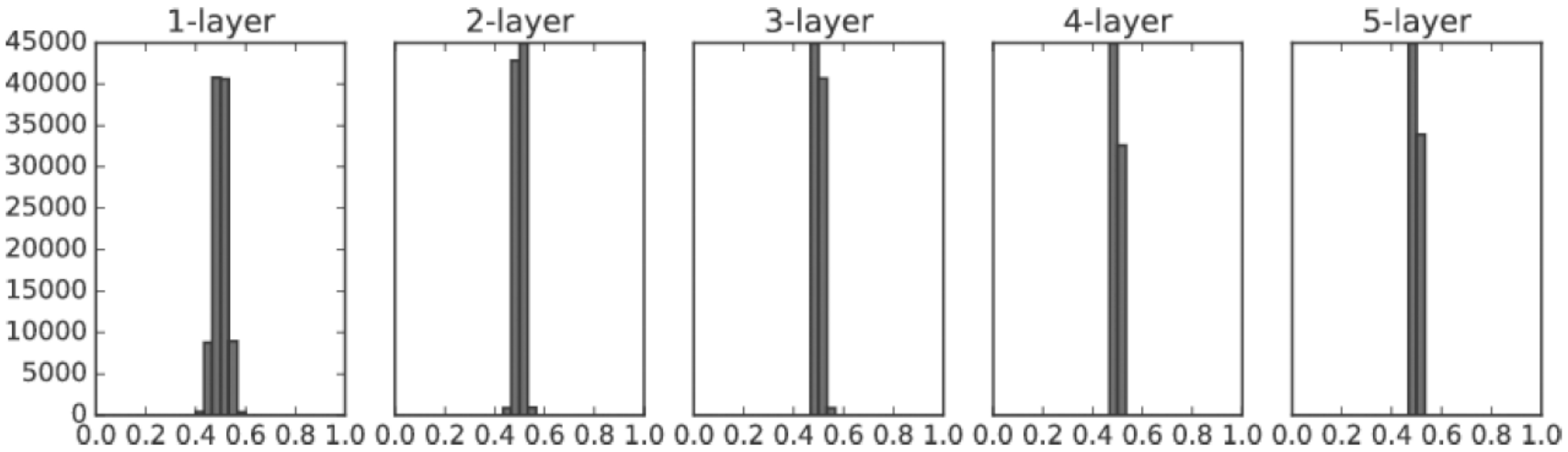
Weight Vector 의 초기값 설정 – cont.

- Weight vector를 표준편차가 1인 정규분포로 초기화
  - 각 층의 활성화 값들이 0과 1에 치우쳐 분포되고 있다.
  - Sigmoid function은 0 또는 1 근방에서 그 미분이 0에 수렴한다.
  - 따라서, 데이터가 0과 1에 치우쳐 분포하게 되면 역전파의 기울기 값이 점점 작아지다가 사라진다.
  - 이러한 현상을 gradient vanishing이라고 부른다.
  - Layer가 증가할 수록 더 심각한 문제가 될 수 있다.



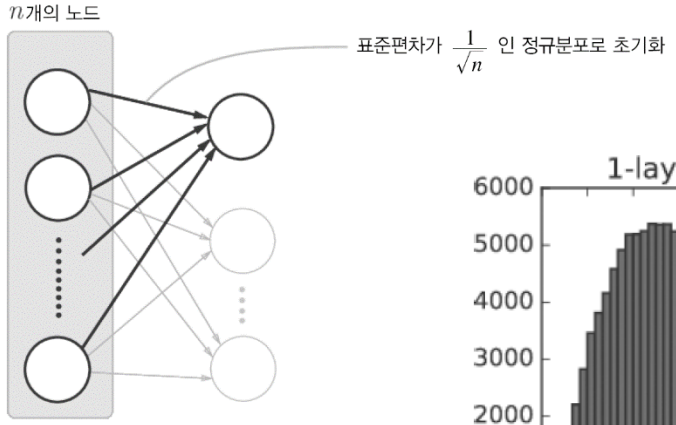
Weight Vector 의 초기값 설정 – cont.

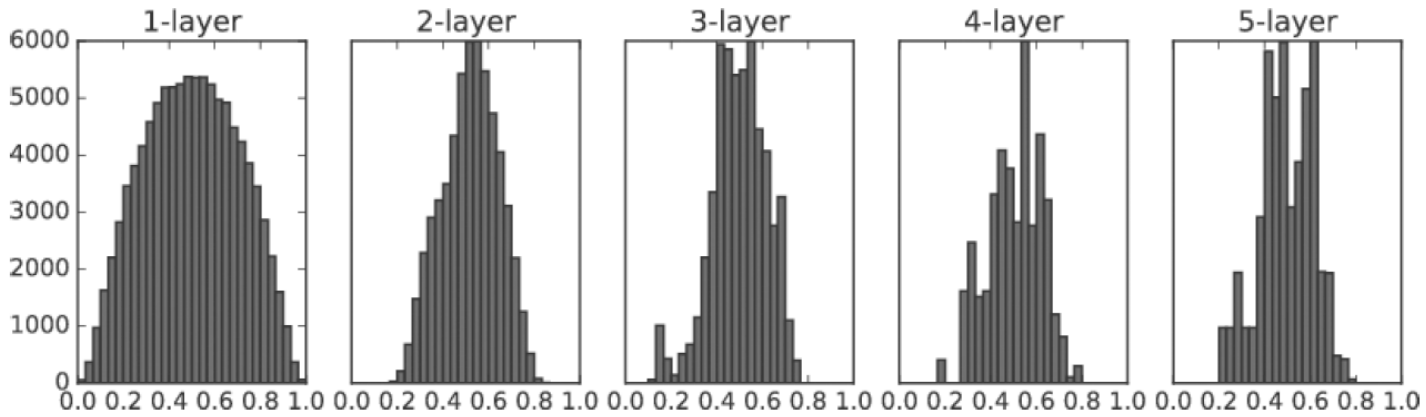
- Weight vector를 표준편차가 0.01인 정규분포로 초기화
  - 각 층의 활성화 값들이 0.5 부근에 치우쳐 분포되고 있다.
  - 이전 경우보다 나쁘진 않지만(gradient vanishing이 일어나지 않지만) 여전히 문제가 많다.
  - 다수의 뉴런이 거의 같은 값을 출력하고 있으니 뉴런을 여러 개 둔 의미가 없다.



Weight Vector 의 초기값 설정 – cont.

- Xavier로 Weight vector 초기화
  - 일반적인 딥러닝 프레임워크들이 표준적으로 이용한다.
  - 각 layer의 활성화 값을 광범위하게 분포시킬 목적으로 weight vector의 적절한 분포를 찾고자 한다.
  - 이전 layer의 수가 n 개라면 random number의 표준편차가 $\frac{1}{\sqrt{n}}$ 인 분포를 사용한다.

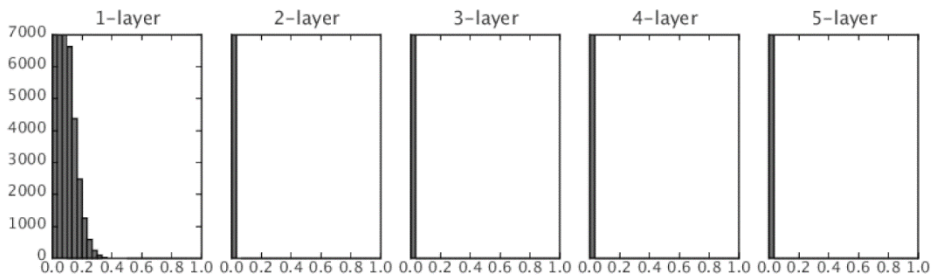




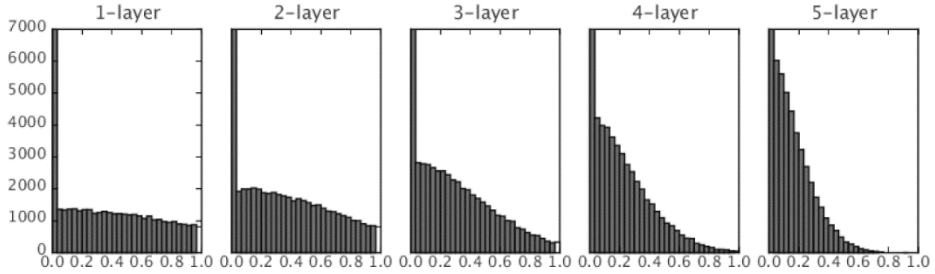
Weight Vector 의 초기값 설정 – cont.

- Relu를 사용할 때의 Weight vector 초기화
  - He 초기값을 이용한다.
  - 이전 layer의 수가 n 개라면

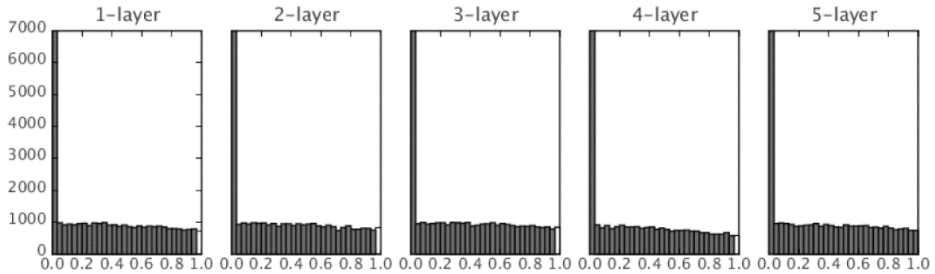
    표준편차가 $\sqrt{\frac{2}{n}}$ 인 분포를 사용한다.
  - Xavier 초기값: 층이 깊어지면서 치우침이 조금씩 커진다.
  - He 초기값: 모든 층에서 균일 분포



표준편차가 0.01인 정규분포를 가중치 초깃값으로 사용한 경우



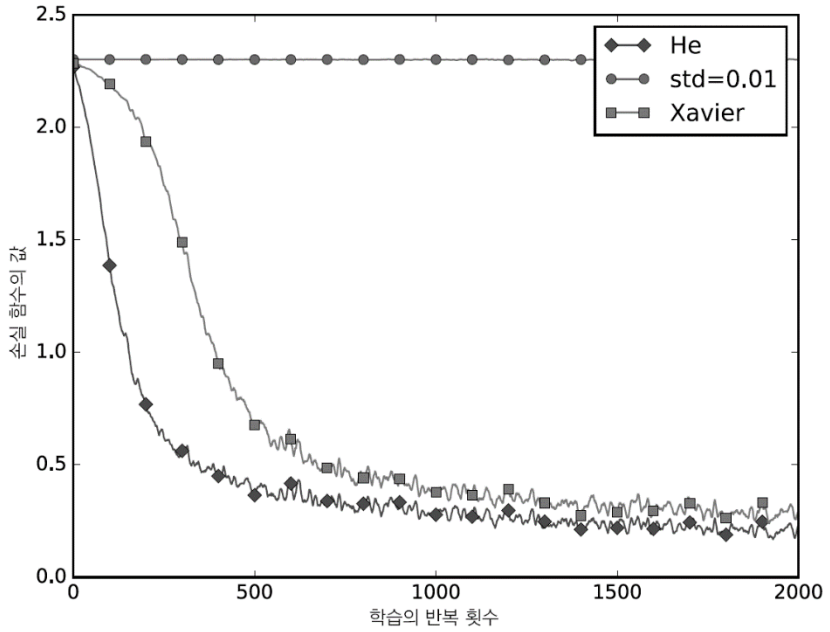
Xavier 초깃값을 사용한 경우



He 초깃값을 사용한 경우

Weight Vector 의 초기값 설정 – cont.

- MNIST Dataset으로 실험한 Weight Vector 초기값에 따른 성능변화
  - 각 layer 별 뉴런 수가 100개인 5-layer NN에서 ReLu를 activation function으로 사용
  - Std=0.01 에서는 학습이 전혀 이루어 지지 않는다.
    즉, forward propagation 때, 너무 작은 값(0 근처로 밀집한 데이터)이 흐르기 때문
    그로 인해, backward propagation 때의 기울기도 작아져 weight vector가 거의 update되지 않는다.
  - Weight vector 초기값: 신경망에서 아주 중요한 포인트



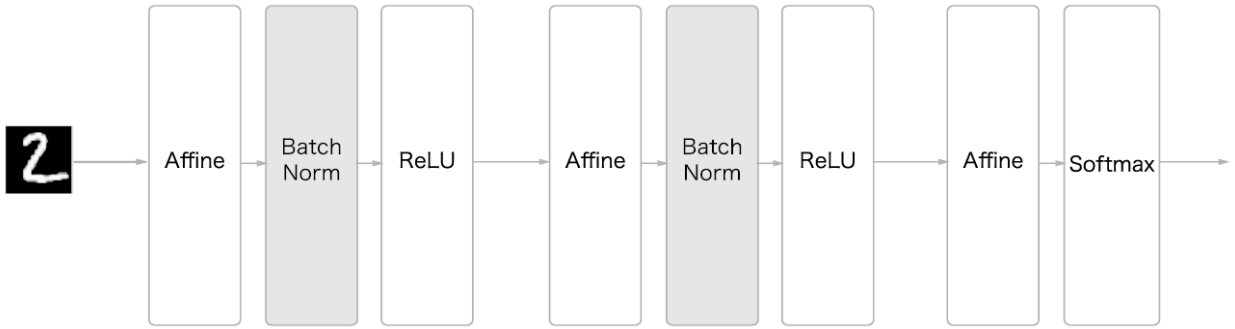
Batch Normalization

- Batch normalization algorithm
  - 각 층이 활성화를 적당히 퍼뜨리도록 강제하는 방법
  - Training 시킬 때 미니배치를 단위로 정규화한다.
  - 즉, 데이터 분포가 mean=0, var=1이 되도록 정규화한다.
  - 수식으로는 다음과 같다.

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

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

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



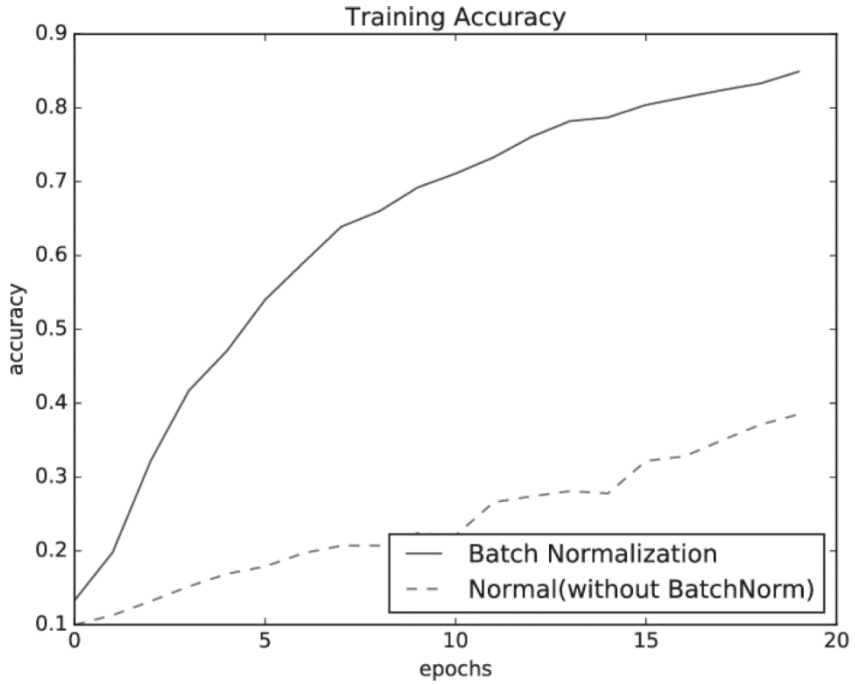
Batch normalization을 이용한 신경망의 예

  - 또, 배치 정규화 계층마다 이 정규화된 데이터에 고유한 확대와 이동을 다음과 같이 수행한다.

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

Batch Normalization – cont.

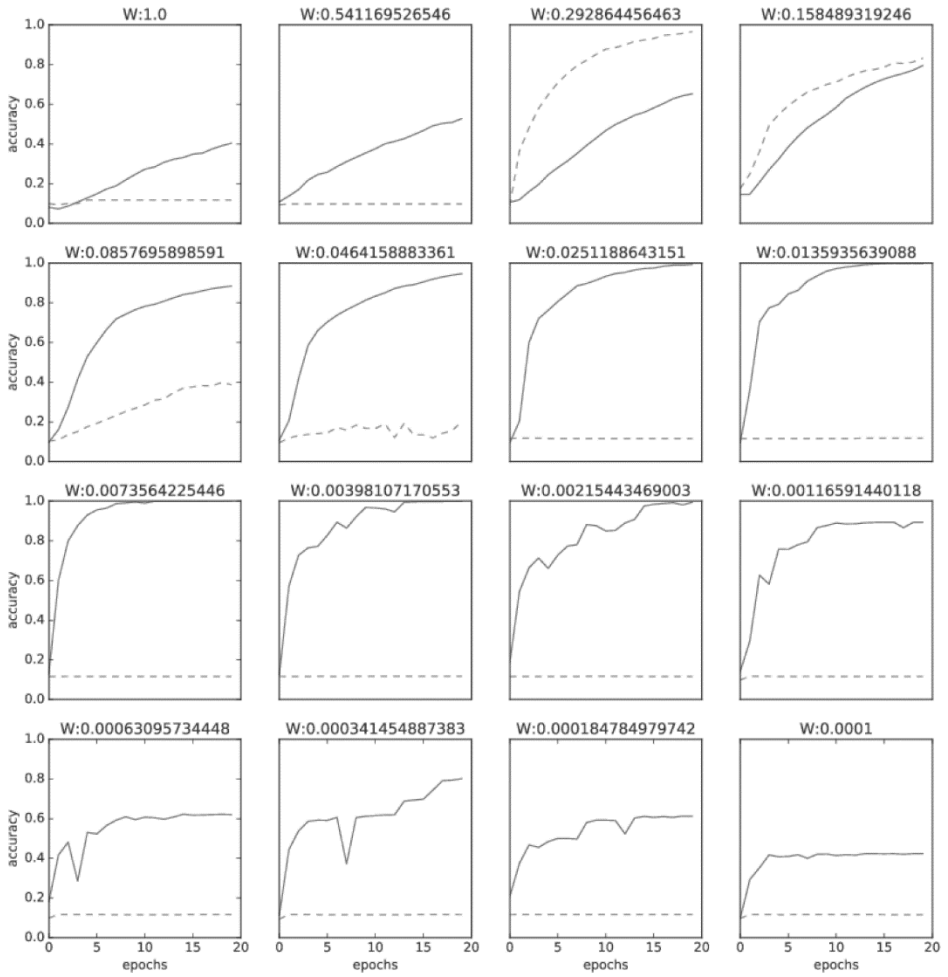
- Batch normalization 의 효과
  - 학습을 빨리 진행할 수 있다(학습속도 개선).
  - Weight vector 초기값에 크게 의존하지 않는다.
  - Overfitting을 억제한다(dropout 등의 필요 감소).



batch normalization이 학습속도를 높인 예

Batch Normalization – cont.

- Weight vector 초기값의 표준편차를 다양하게 바꿔가며 학습경과를 관찰한 그래프
  - 거의 모든 경우에 배치 정규화 시, 학습 진도가 빠르게 나타난다.
  - 배치정규화를 이용하지 않는 경우엔, 초기값이 잘 분포되어 있지 않으면 학습이 잘 이루어지지 않는다.
  - 따라서 배치 정규화를 사용하면 학습이 빨라지고 weight vector 초기값의 영향을 별로 받지 않는다.



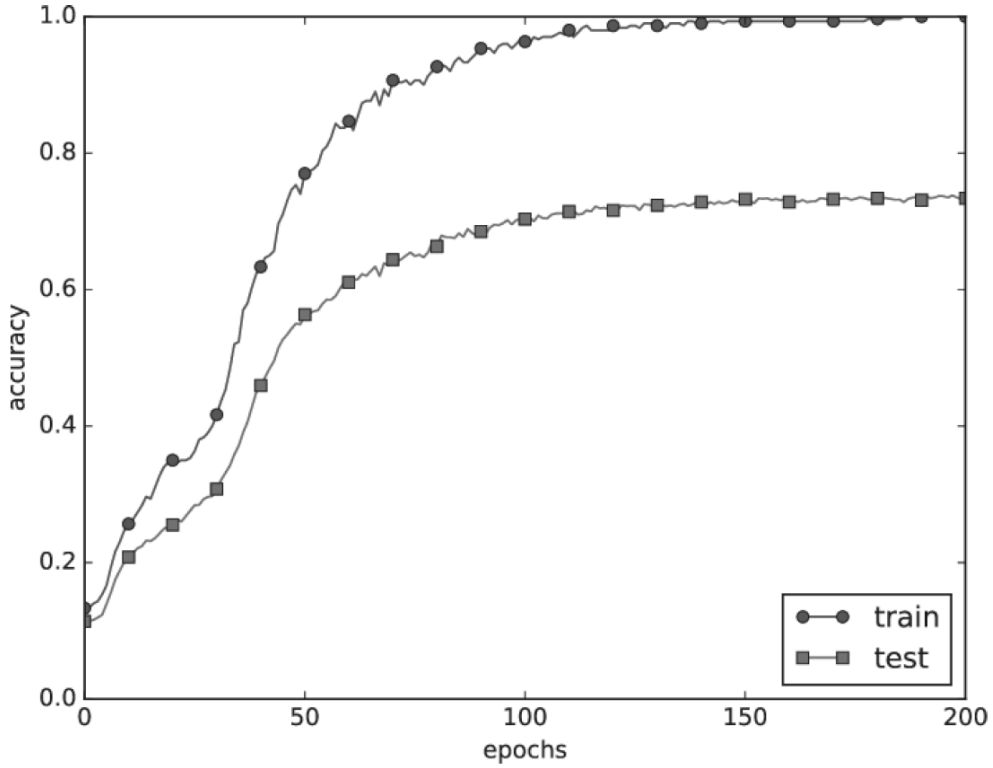
바른 학습을 위해

- Overfitting의 원인
  - 매개변수가 많고 표현력이 높은 모델
  - 학습 데이터가 적음

- Overfitting 실험
  - 60,000개의 학습 데이터 중 300개만 사용하고
  - 7-layer 네트워크를 사용해 네트워크의 복잡성을 높여 강제로 overfitting을 발생시킨다.
  - 이 때, 각 층의 뉴런은 100개, 활성화 함수는 Relu를 사용한다.
  - OverfitTest.ipynb 참조

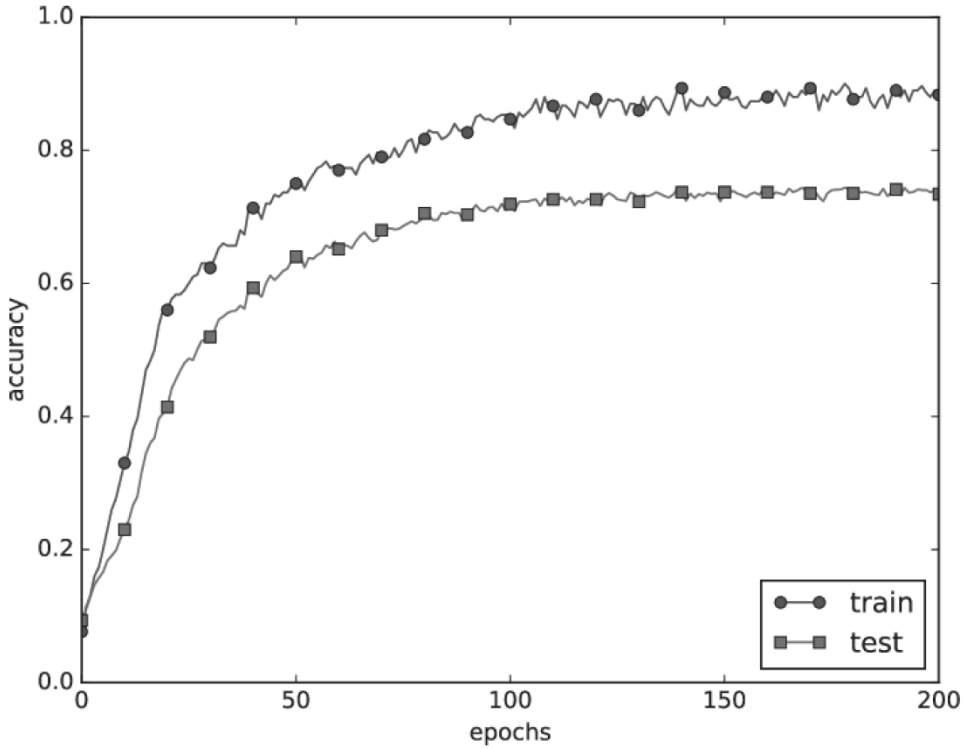
바른 학습을 위해

- Training 데이터와 Test 데이터의 epoch 별 정확도 추이
  - 정확도가 크게 벌어지는 이유는 training data에만 적응해 버린 때문이다.
  - 학습 시 사용하지 않은 테스트 데이터에는 제대로 대응하지 못했다.



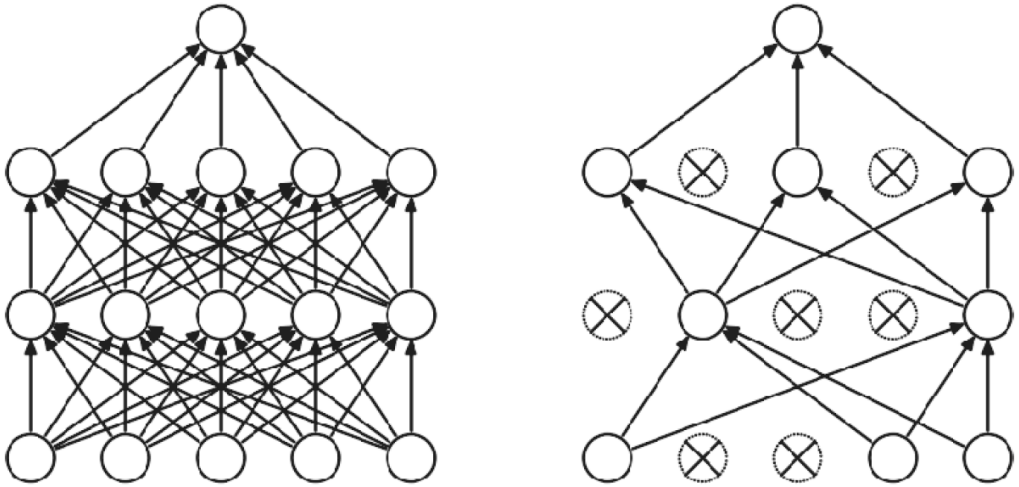
바른 학습을 위해

- Weight Decay
  - Overfitting 억제용으로 많이 이용
  - 학습과정에서 큰 가중치에 대해서는 그에 상응하는 큰 penalt를 부과하여 overfitting을 억제한다.
  - 원래 overfitting은 weight vector의 값이 커서 발생하는 경우가 많기 때문이다.
  - 손실함수에 L2-norm을 더했을 때의 실험결과는 다음과 같다.
  - OverfitWeightDecay.ipynb 참조



바른 학습을 위해

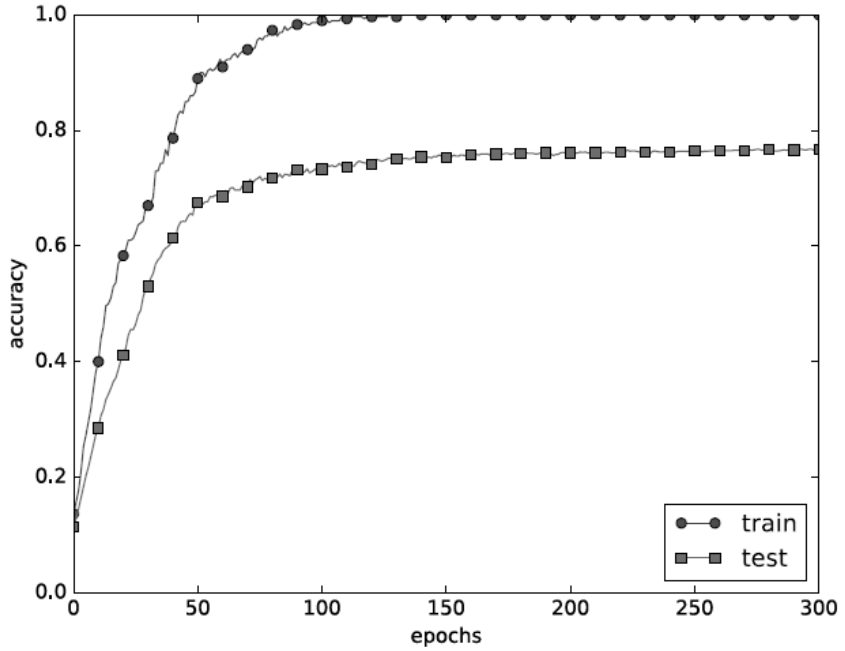
- Dropout
  - 신경망이 복잡해지면 weight decay 만으로는 Overfitting 억제에 한계가 있을 수 있다.
  - 이럴 경우에 dropout 기법을 이용한다.
  - Dropout은 뉴런을 임의로 삭제하면서 학습하는 방법이다.
  - 삭제된 뉴런은 신호를 전달하지 않는다.
  - Training 시에는 데이터를 흘릴 때마다 삭제할 뉴런을 무작위로 선택하고, Test 시에는 모든 뉴런에 신호를 전달한다.
  - Dropout 실험은 MNIST를 대상으로 7-layer network(각 layer의 노드수는 100개, 활성화 함수는 Relu 선택)를 이용하여 실행한다.
  - OverfitDropout.ipynb 참조

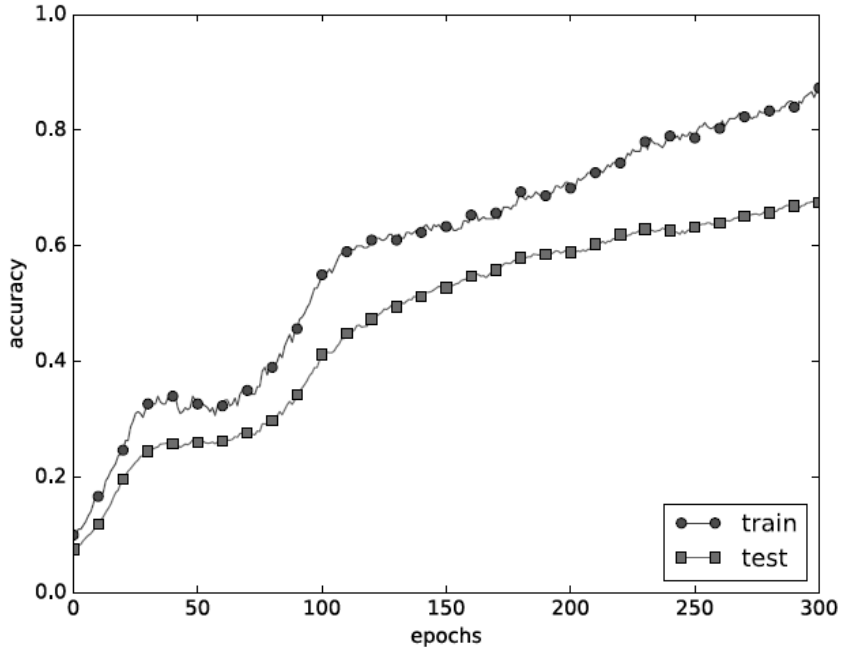


(a) 일반 신경망 (b) 드롭아웃을 적용한 신경망

바른 학습을 위해

- Dropout
  - 왼쪽은 Dropout 없이, 오른쪽은 drooout을 적용한 결과이다.
  - Dropout은 Ensemble Learning과 동일한 효과를 하나의 네트워크로 구현했다고 볼 수 있다.





Hyperparameter Optimization

- Hyperparameters
  - 각 layer의 뉴런 수, 배치크기, learning rate, weight decay parameter 등이 있다.
  - 적절히 설정하지 않으면 모델의 성능이 크게 떨어질 수 있다.

- Validation data
  - Hyperparameters의 성능을 평가할 때에는 test data를 이용하면 안된다.
  - 그 이유는 hyperparameters가 test data에 overfitting되기 때문이다.
  - 즉, hyperparameters의 좋음을 test data로 확인하게 되므로 그 값이 test data에만 적합하도록 조정되어 버리기 때문이다.
  - 그렇게 되면 다른 데이터에는 적응하지 못하니 generalization이 떨어지는 모델이 될 가능성이 높다.
  - 따라서 hyperparameter 전용 확인 데이터가 필요하다.
  - Training data : weight vectors, bias 등 학습
  - Validation data : hyperparameters 성능 평가
  - Test data : 신경망의 generalization 성능 평가

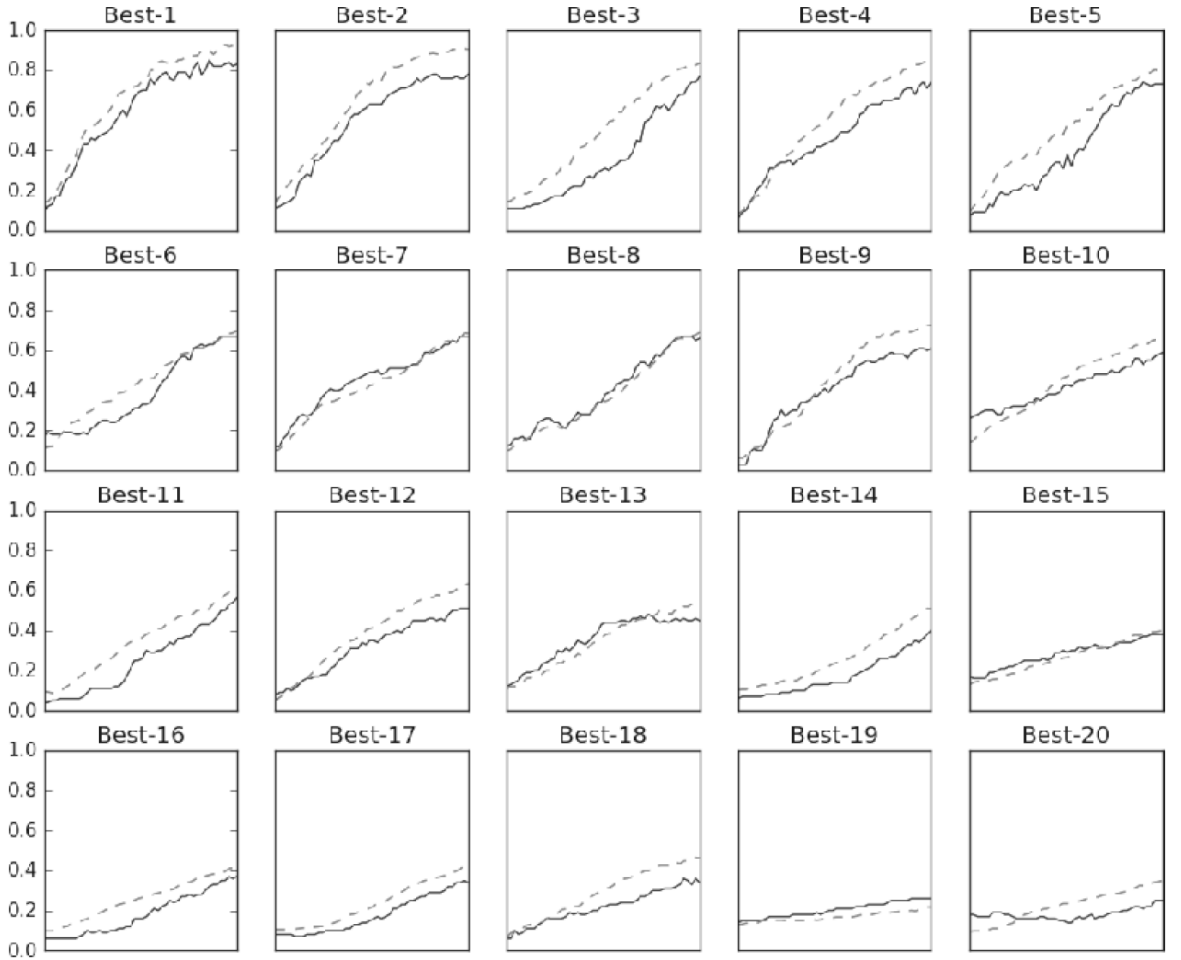
Hyperparameter Optimization – cont.

- Hyperparameter optimization 방법
  - 1단계 : hyperparameter 값의 범위를 설정한다.
  - 2단계 : 설정된 범위에서 hyperparameter 값을 무작위로 추출한다.
  - 3단계 : 2단계에서 샘플링한 hyperparameter 값을 이용하여 학습하고 검증 데이터로 정확도를 평가한다. Epoch 수는 적게 설정한다.
  - 4단계 : 2단계와 3단계를 특정횟수(100회 정도) 반복하며, 그 정확도의 결과를 보고 hyperparameter의 범위를 좁힌다.

- Hyperparameter optimization 실험
  - 그 이유는 hyperparameters가 test data에 overfitting되기 때문이다.
  - 즉, hyperparameters의 좋음을 test data로 확인하게 되므로 그 값이 test data에만 적합하도록 조정되어 버리기 때문이다.
  - 그렇게 되면 다른 데이터에는 적응하지 못하니 generalization이 떨어지는 모델이 될 가능성이 높다.
  - 따라서 hyperparameter 전용 확인 데이터가 필요하다.
  - Training data : weight vectors, bias 등 학습
  - Validation data : hyperparameters 성능 평가
  - Test data : 신경망의 generalization 성능 평가

Hyperparameter Optimization – cont.

- Hyperparameter optimization 실험
  - Weight decay parameter의 범위를 10^-8 ~ 10^-4, 와 learning rate의 범위를 10^-6 ~ 10^-2로 실험한 결과는 다음과 같다.
  - 실선: 검증 데이터에 대한 정확도
  - 점선 : training 데이터에 대한 정확도



수고하셨습니다.