분석 멘토링 C조
5주차

발표자: 곽민지

CH 6
학습 관련 기술들

1. 매개변수 갱신

매개변수 갱신

신경망 학습의 목적

: 손실함수의 값을 가능한 낮추는 매개변수를 찾는 것

→ 최적화

매개변수 갱신

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

확률적 경사 하강법

데이터를 무작위로 선정하여 경사 하강법을 적용하는 매개변수 갱신 방법

추출된 데이터 한 개에 대해서 그라디언트를 계산

매개변수 갱신

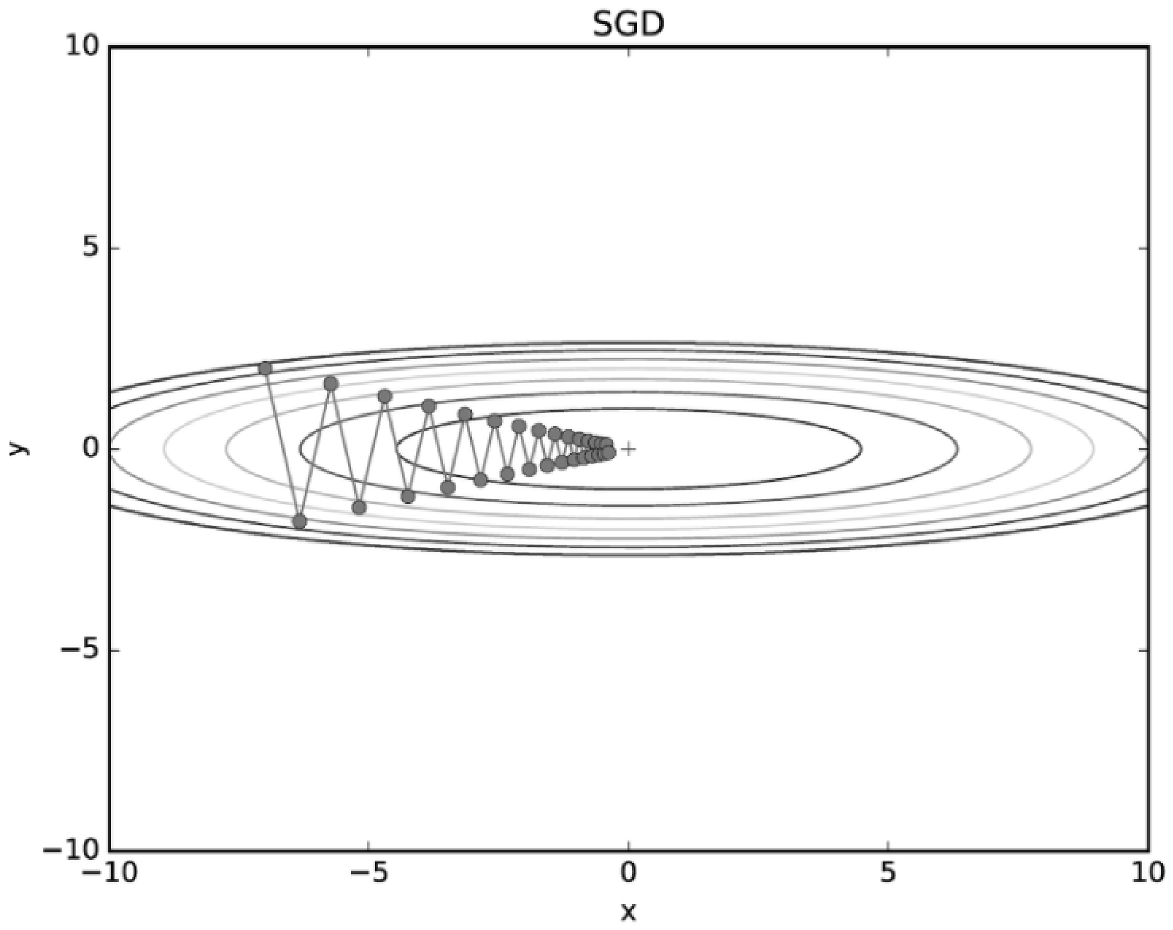

```python
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr

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

→ SGD 클래스 구현

```python
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) ###
    ...
```

→ 매개변수 갱신 진행

매개변수 갱신

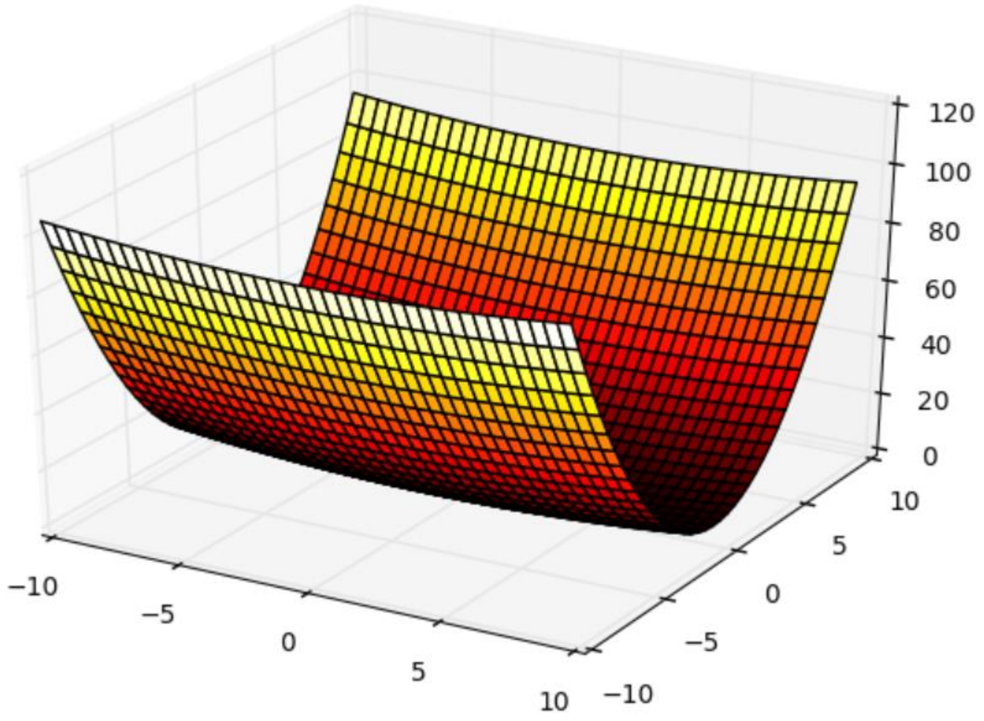


- 비등방성 함수에서 탐색 경로가 비효율적
- 지그재그로 탐색하는 근본원인은 기울어진 방향이 본래의 최솟값과 다른 방향을 가리켜서임.

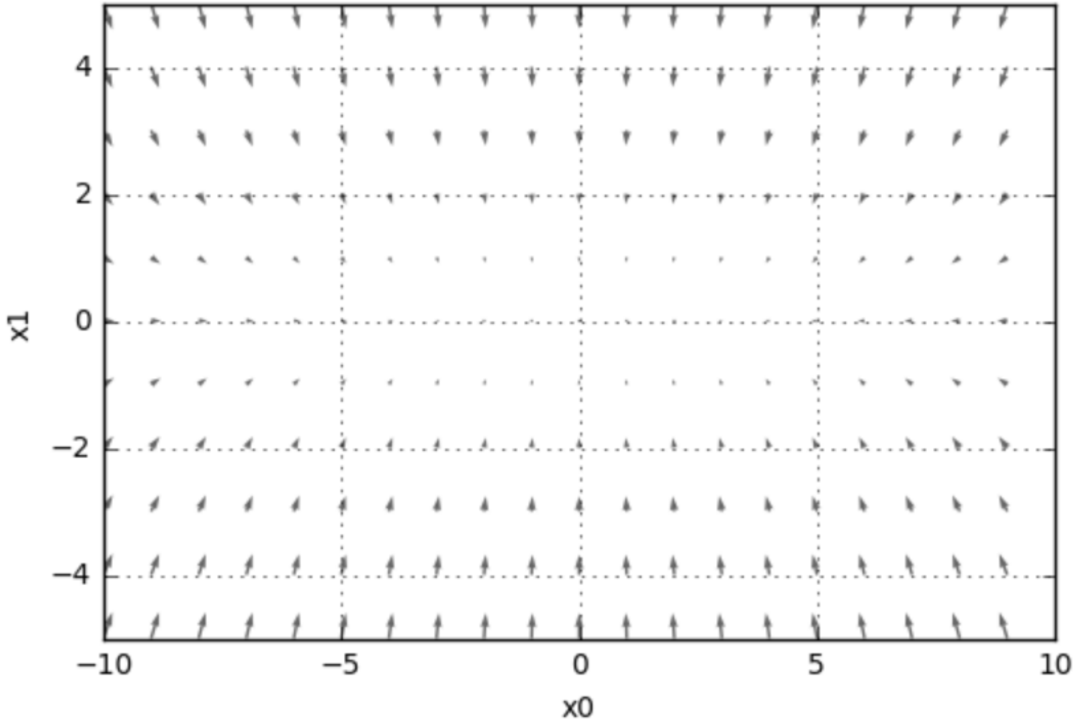
＊ 비등방성 함수 : 방향에 따라 성질(기울기)가 달라지는 함수

매개변수 갱신

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



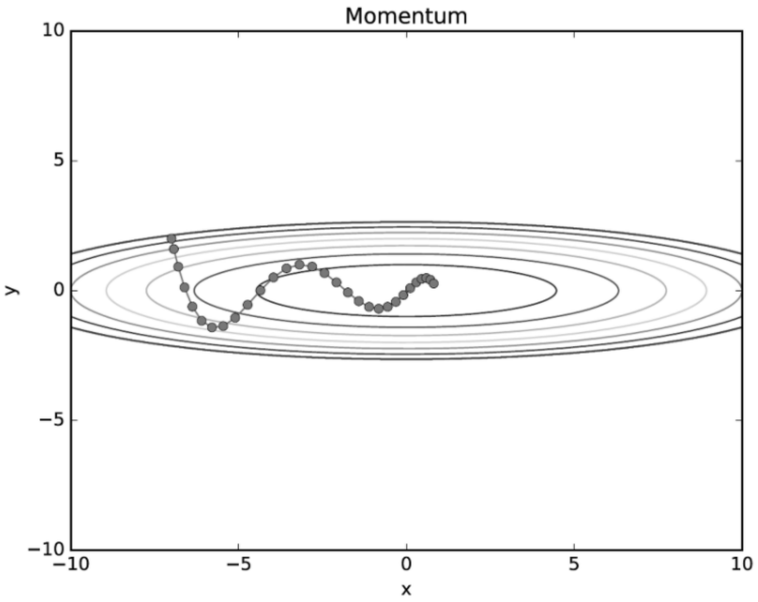
함수의 그래프



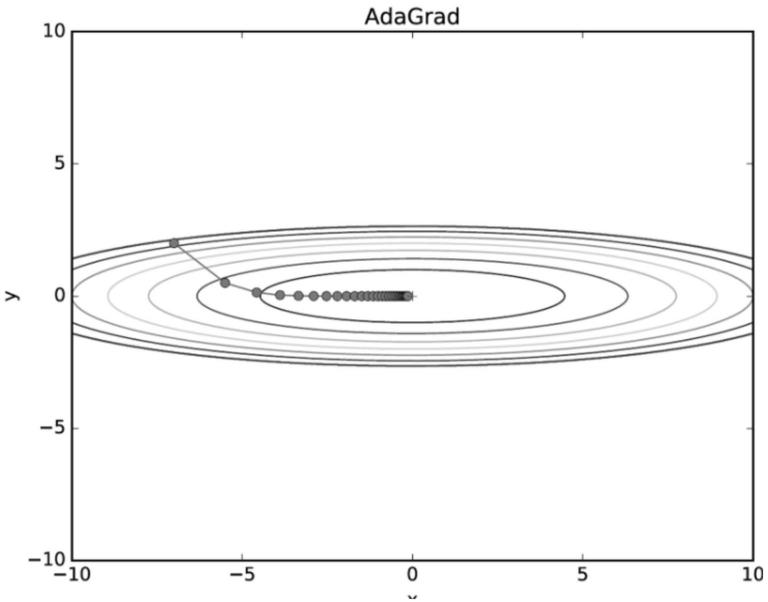
함수의 기울기

매개변수 갱신

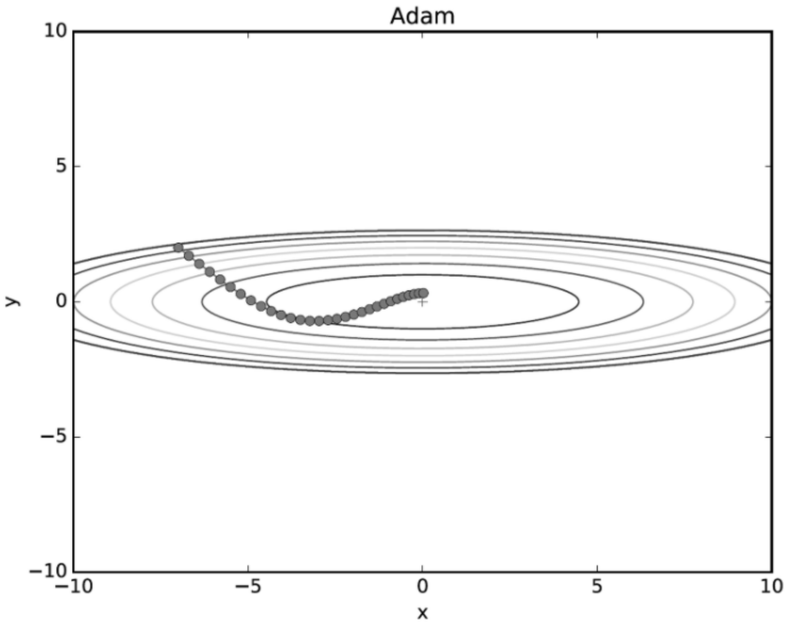
SGD의 단점을 개선한 방법



모멘텀



AdaGrad



Adam

매개변수 갱신

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

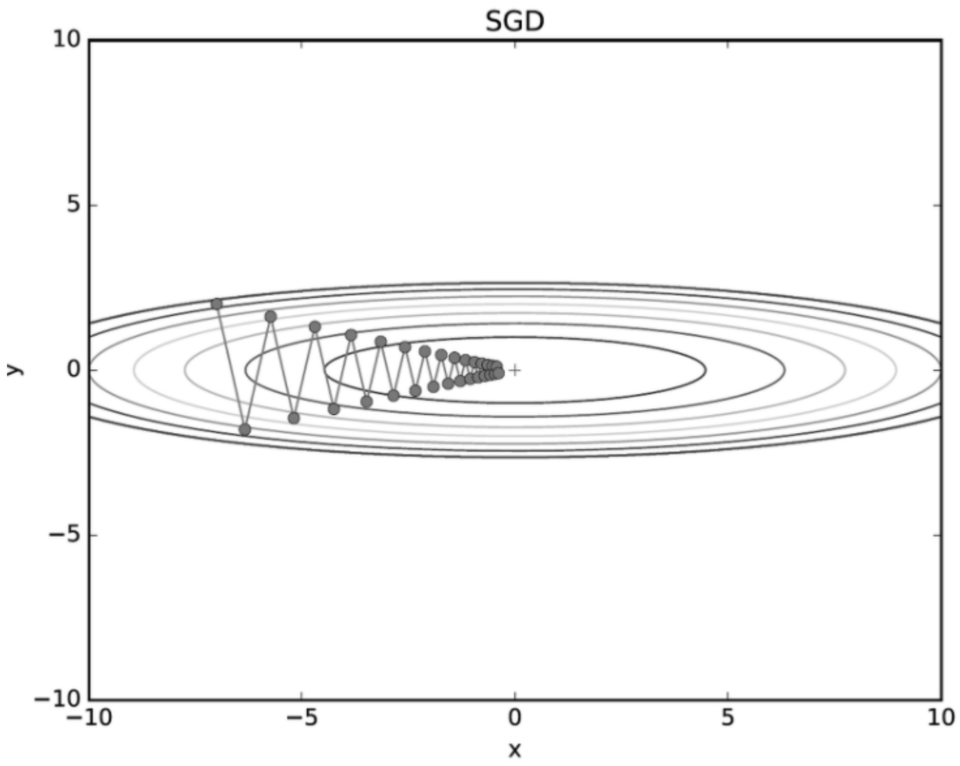
$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}$$

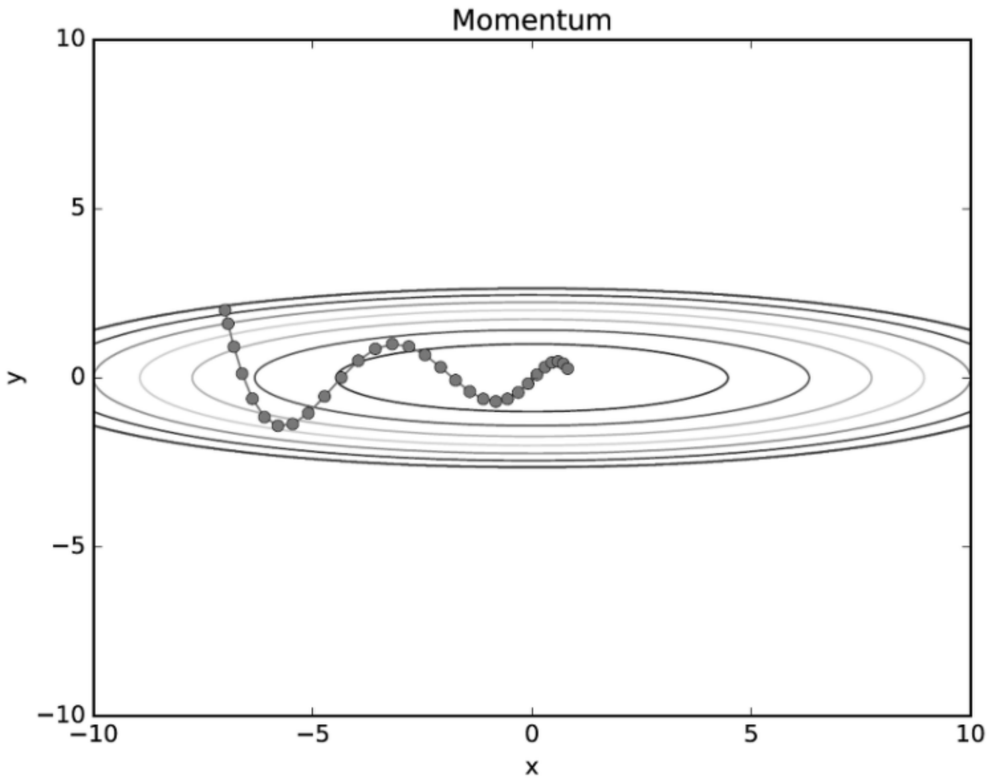
모멘텀

확률적 경사 하강법에 속도의 개념의 더함

매개변수 갱신

③ 모멘텀





탐색 경로가 지그재그로 크게 변하는 SGD → 보다 빠르게 최적점으로 수렴하는 모멘텀

매개변수 갱신

④ AdaGrad

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

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

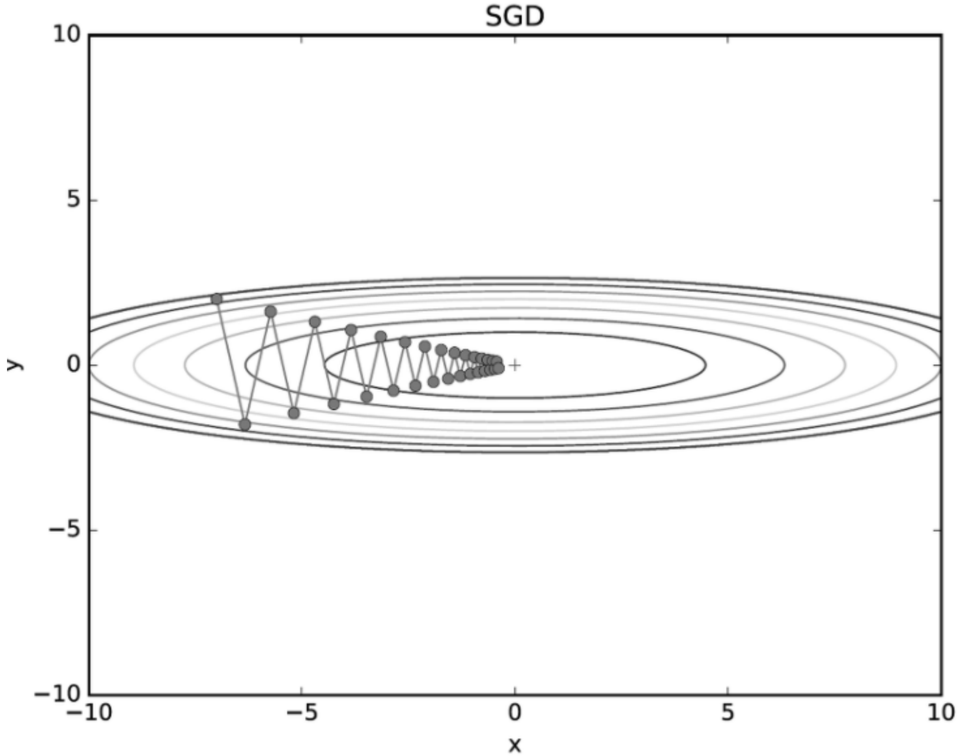
AdaGrad

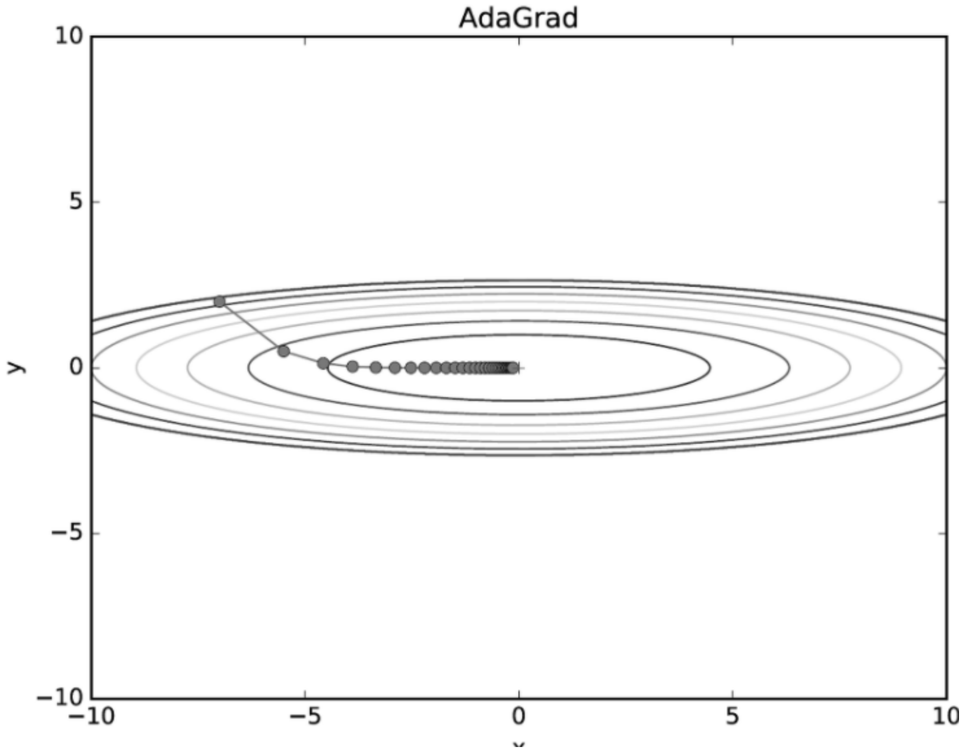
학습을 진행하면서 학습률을 점차 줄이는 ‘학습률 감소 기법’을 적용.

개별 매개변수에 적응적으로 학습률을 조정하면서 학습 진행.

매개변수 갱신

④ AdaGrad



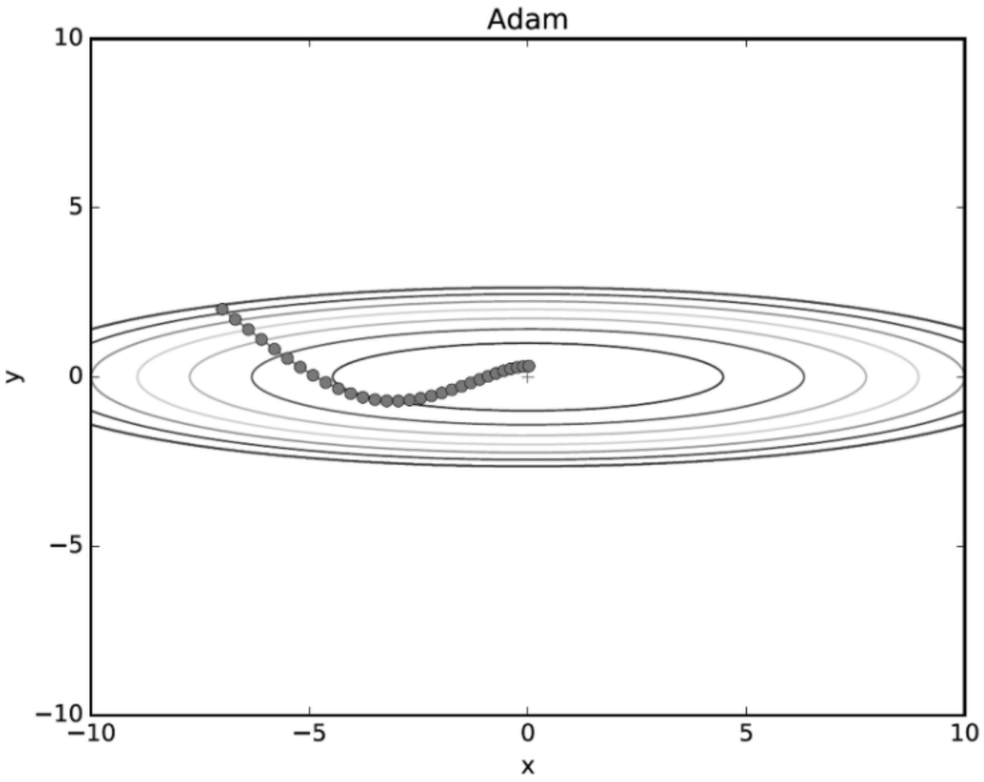


탐색 경로가 지그재그로 크게 변하는 SGD → 갱신 강도가 빠르게 약해지고 지그재그 움직임 감소

매개변수 갱신

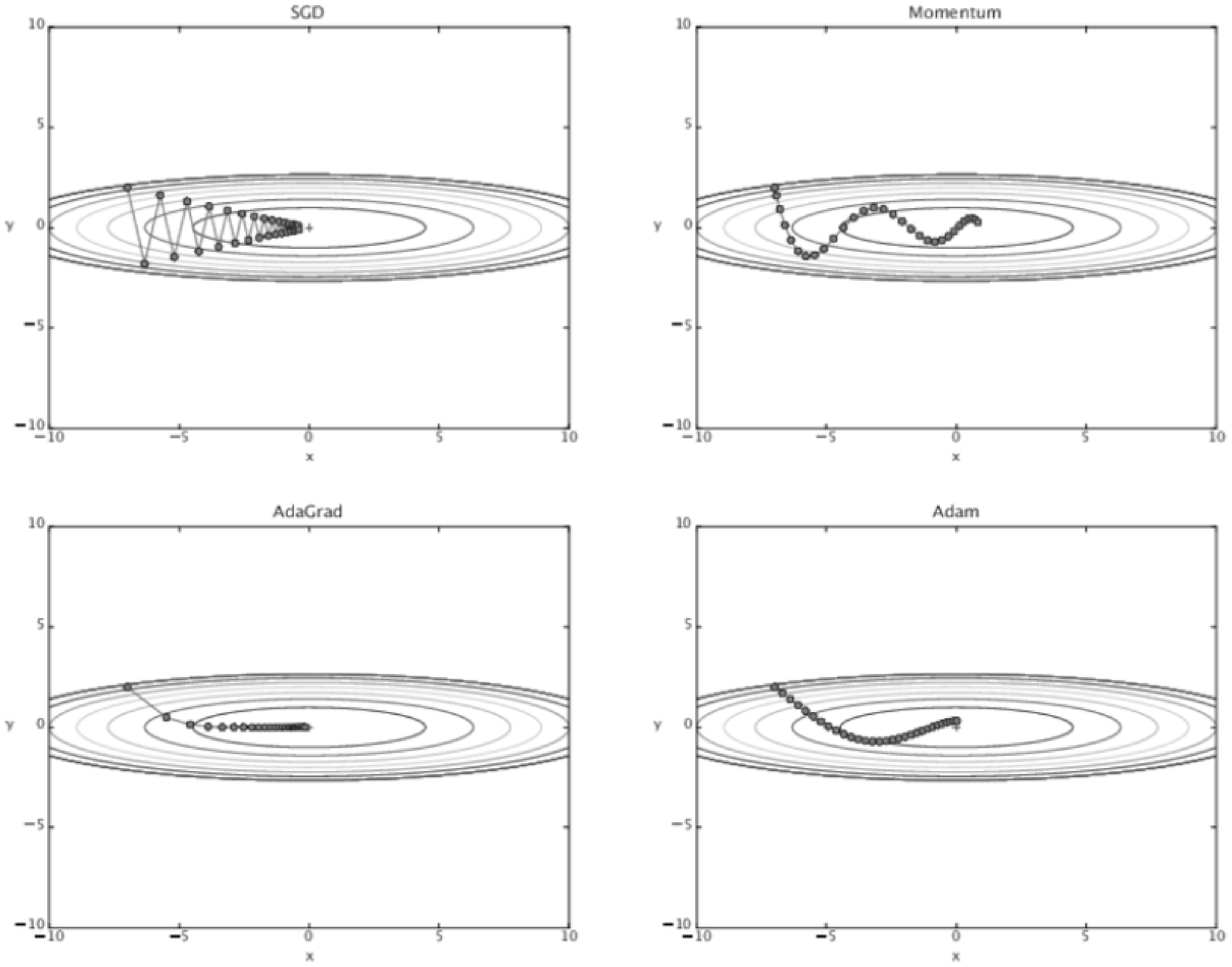
Adam

모멘텀 + AdaGrad



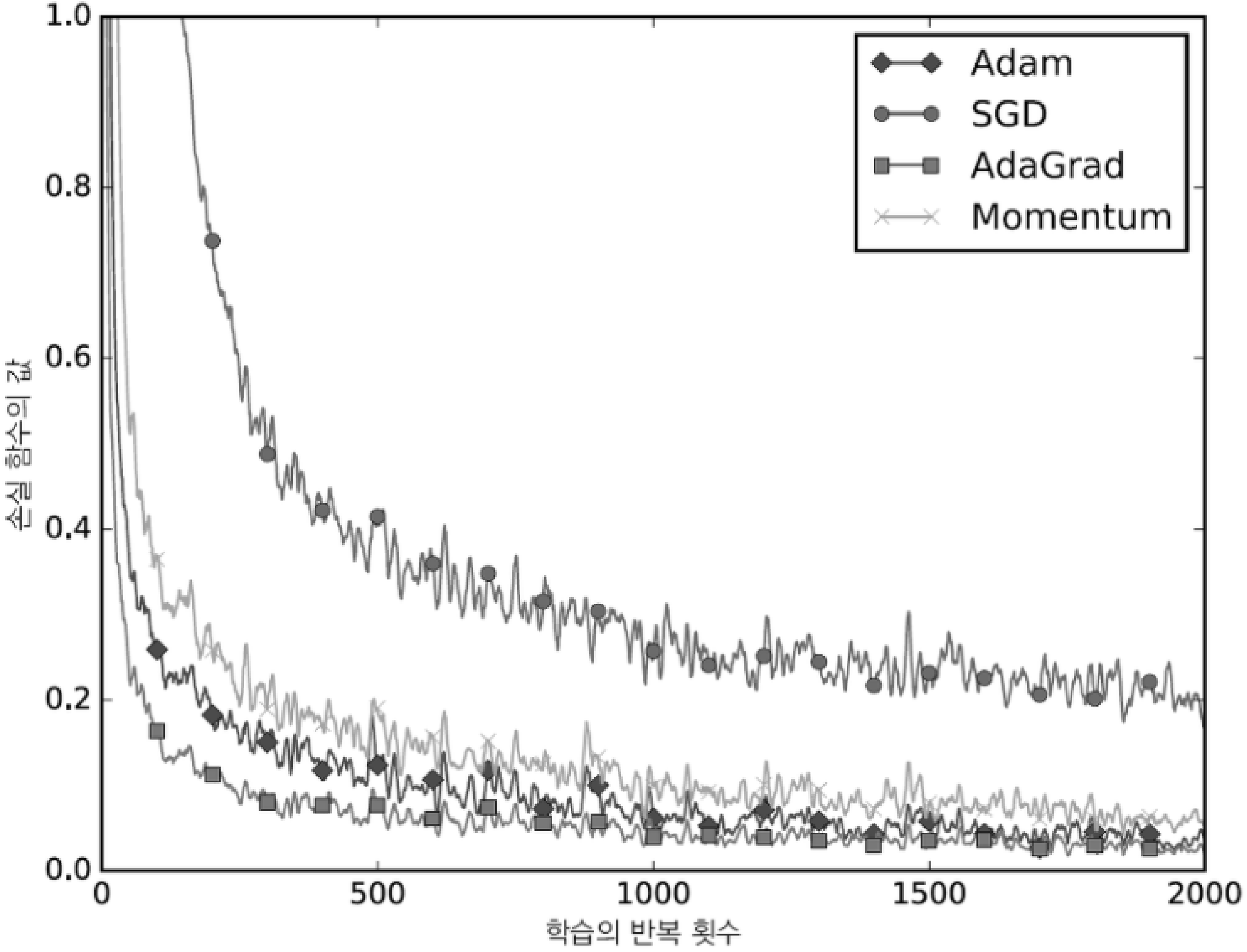
매개변수 갱신

⑥ 어느 갱신 방법을 이용할 것인가?



매개변수 갱신

⑥ 어느 갱신 방법을 이용할 것인가?



2. 가중치의 초깃값

가중치의 초깃값

가중치 초깃값을 0으로 하면?

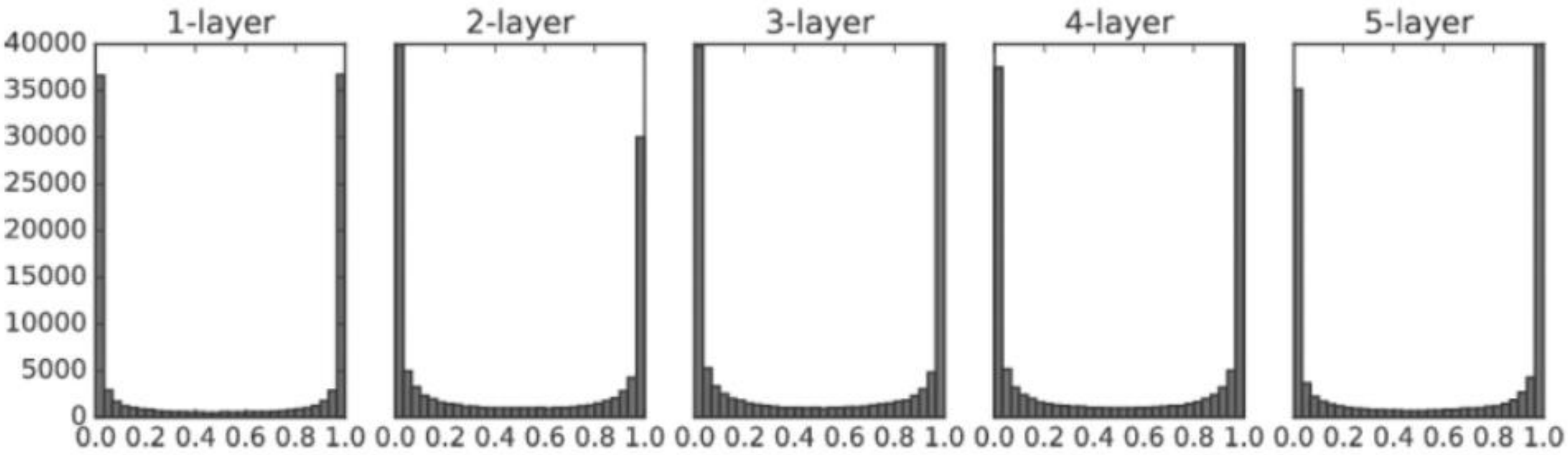
→ 학습이 올바르게 이루어지지 않는다.

가중치를 균일하게 설정하면 오차역전파법에서 모든 가중치의 값이 똑같이 갱신되기 때문

⇨ 가중치의 초깃값은 무작위로 설정

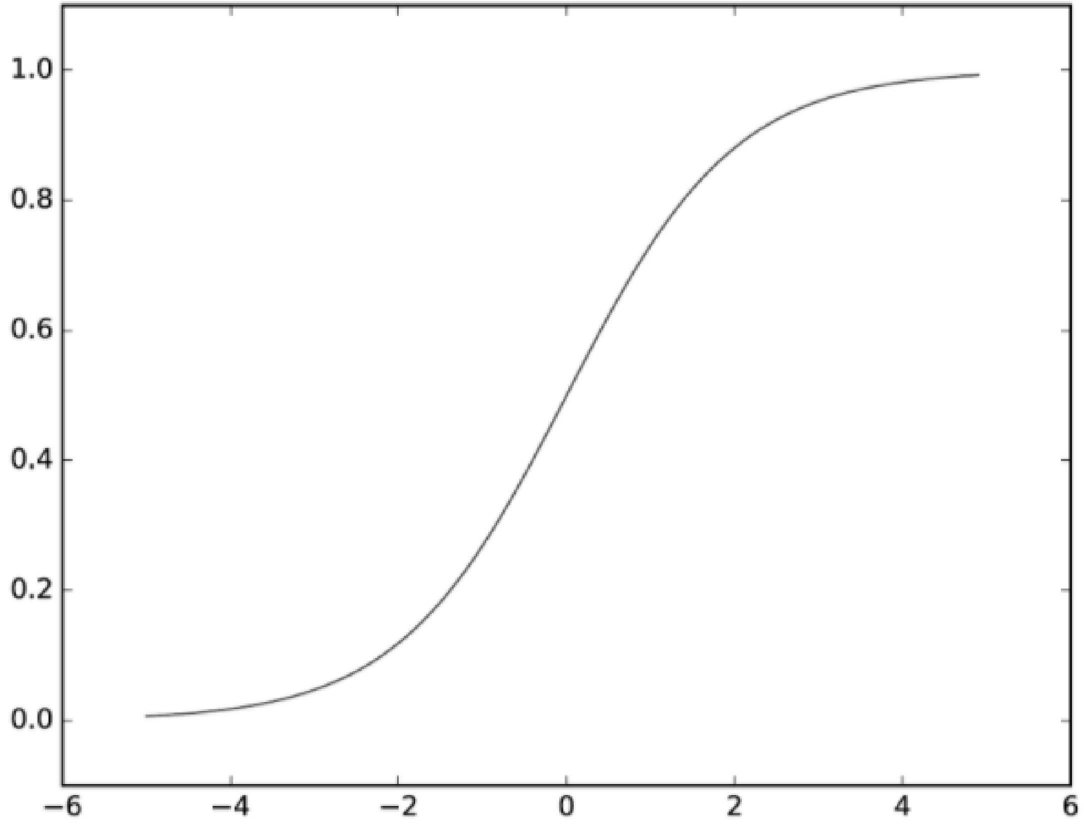
가중치의 초깃값

② 은닉층의 활성화값 분포



가중치를 표준편차가 1인 정규분포로 초기화

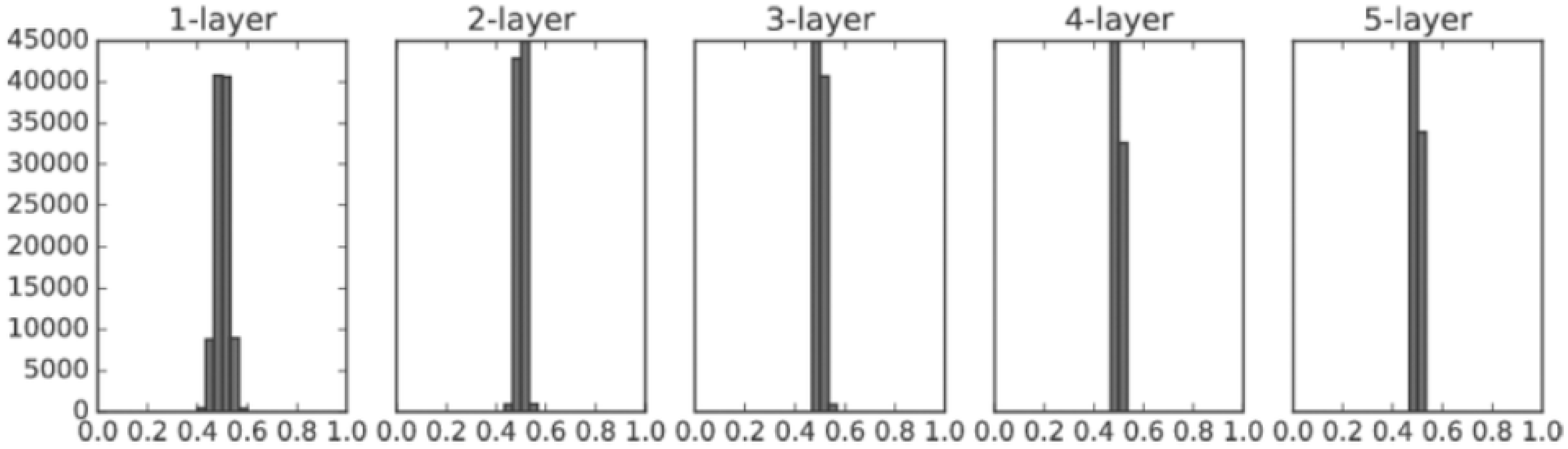
가중치의 초깃값



sigmoid

가중치의 초깃값

② 은닉층의 활성화값 분포



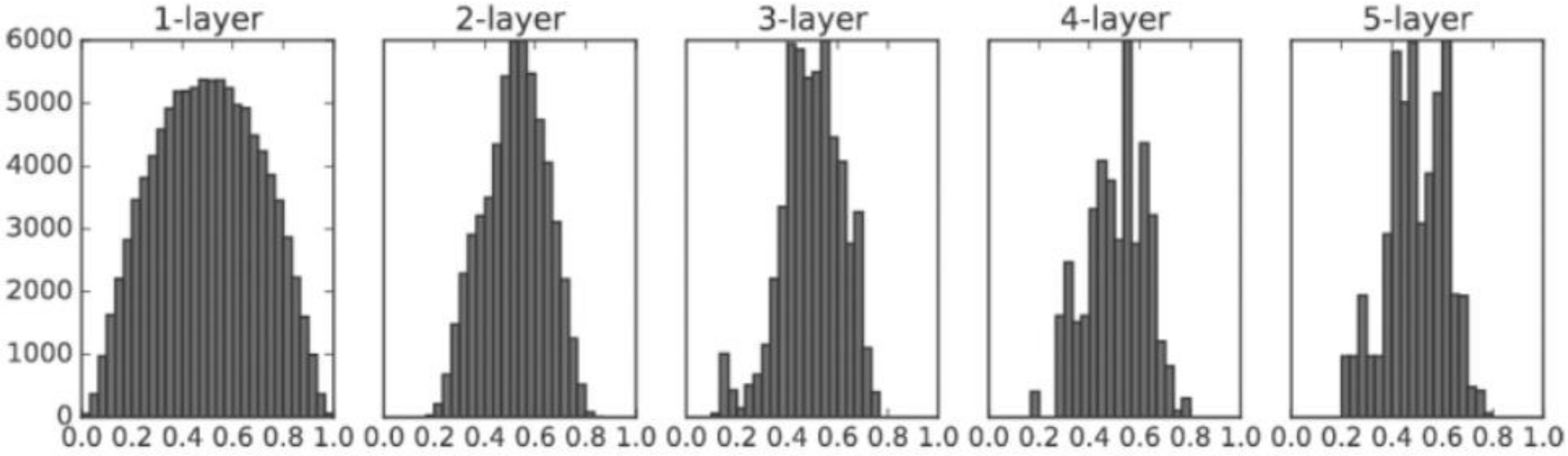
가중치를 표준편차가 0.01인 정규분포로 초기화

가중치의 초깃값

Xavier 초깃값

- 활성화 함수가 sigmoid, tanh함수일 때 사용

- 앞 계층의 노드 수가 n일 때 표준편차 $1/\sqrt{n}$인 가중치 분포

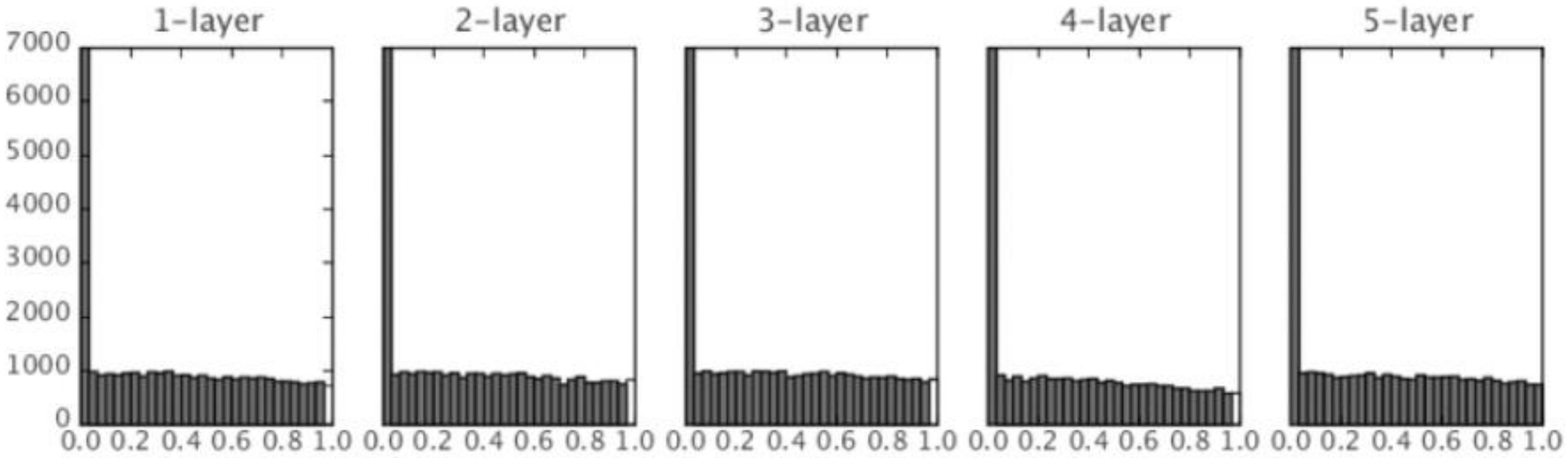


가중치의 초깃값

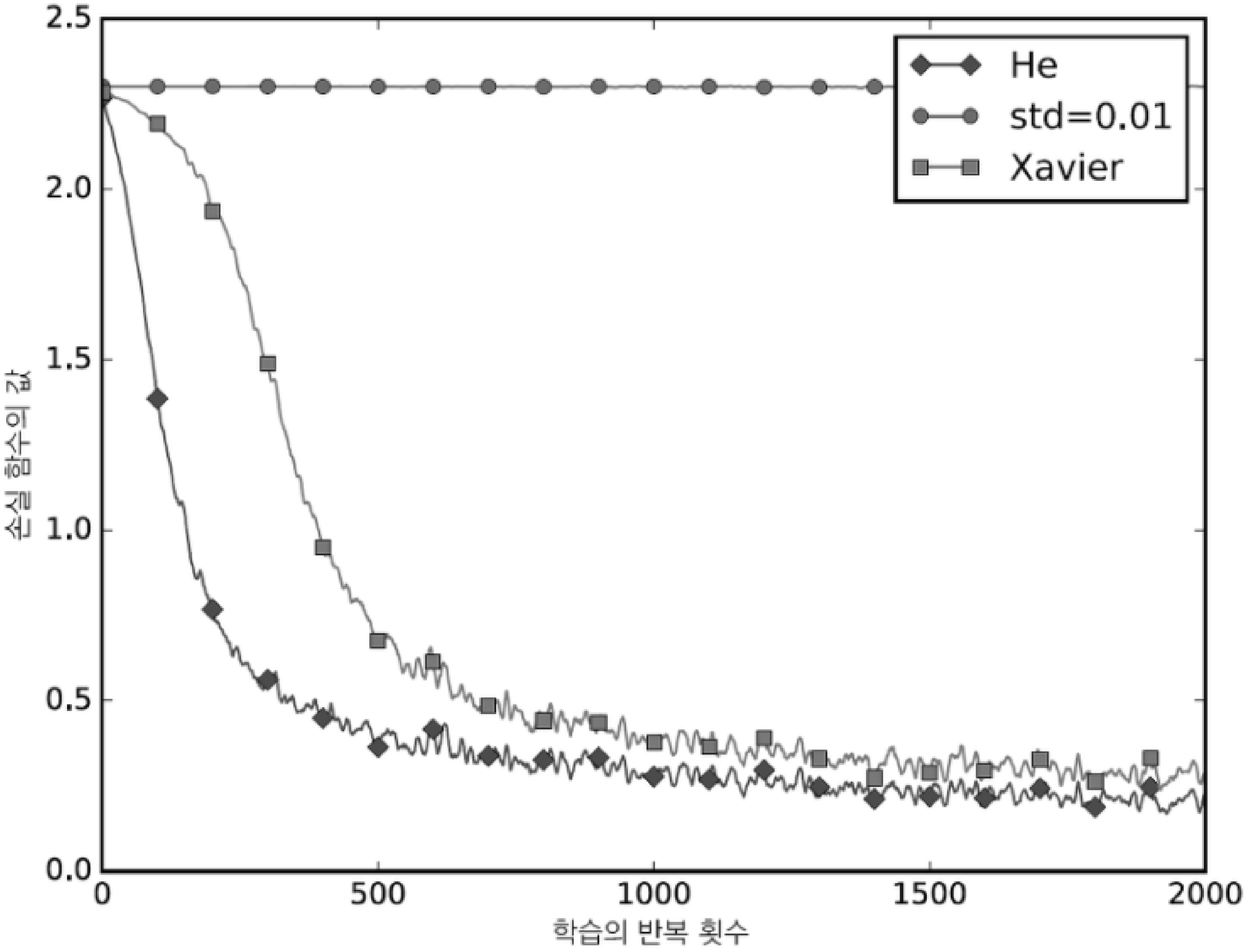
He 초깃값

- 활성화 함수가 ReLU일 때 사용

- 앞 계층의 노드 수가 n일 때 표준편차 $1/\sqrt{\frac{2}{n}}$인 가중치 분포



가중치의 초깃값



- MNIST 데이터셋
- 활성화 함수 : ReLU

3. 배치 정규화

배치 정규화

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

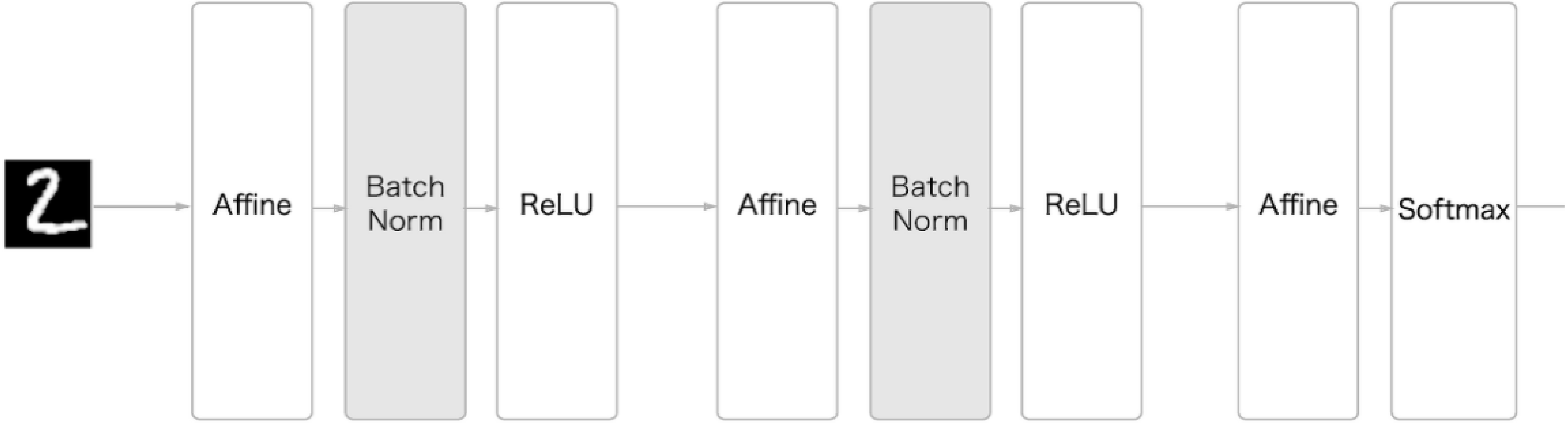
$$\sigma_B^2 := \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$x_i := \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

배치 정규화 (Batch Normalization) 란?

- 학습하는 과정을 전체적으로 안정화 시키는 방법
- 각 층이 활성화를 적당히 퍼뜨리도록 강제함

배치 정규화



배치 정규화 계층을 신경망에 삽입한 모습

배치 정규화

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

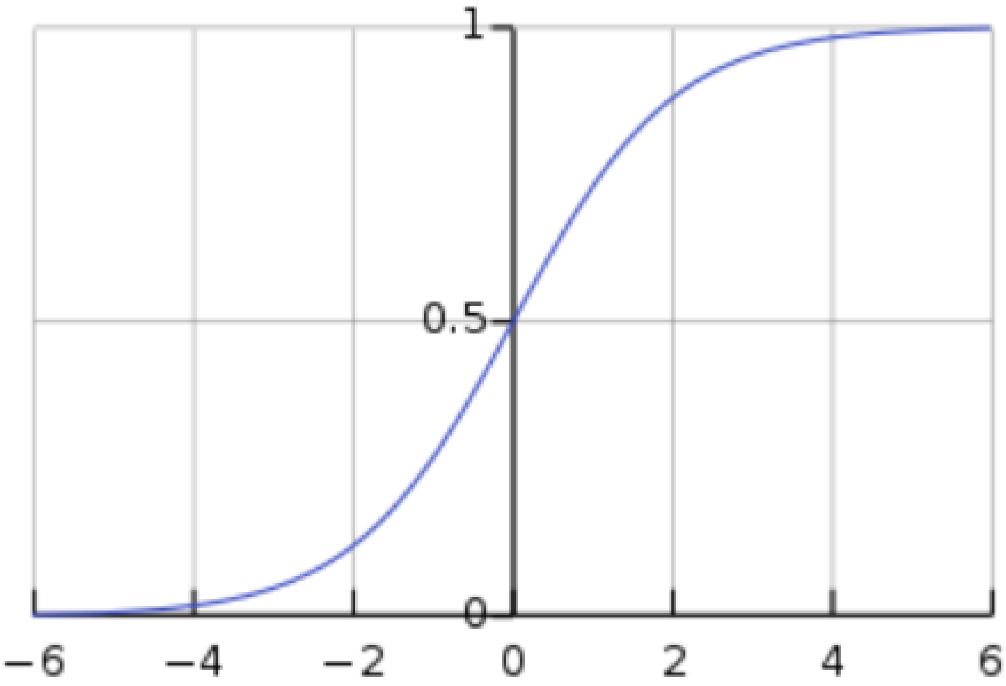
to recover the identity mapping.

감마(scale)와 베타(shift) 조정이 가능하다.

→ saturation 현상을 얼마나 일어나게 할 것인지 조절할 수 있음

→ 배치 정규화를 진행할 것인지, 안 할 것인지 결정 가능함.

배치 정규화

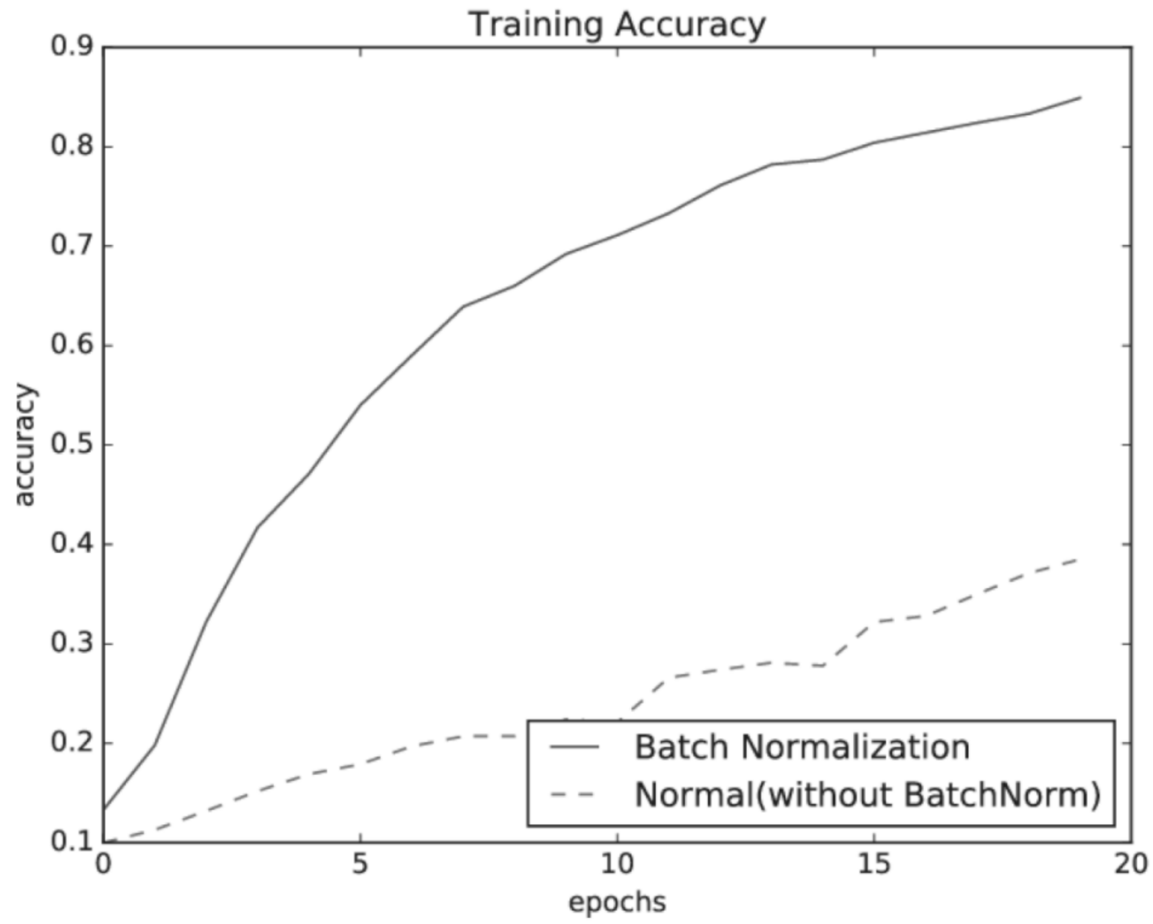


Sigmoid 함수

scale과 shift 사용 이유

- 비선형성을 잃는 함수의 비선형성을 유지
- saturation 현상의 조절

배치 정규화



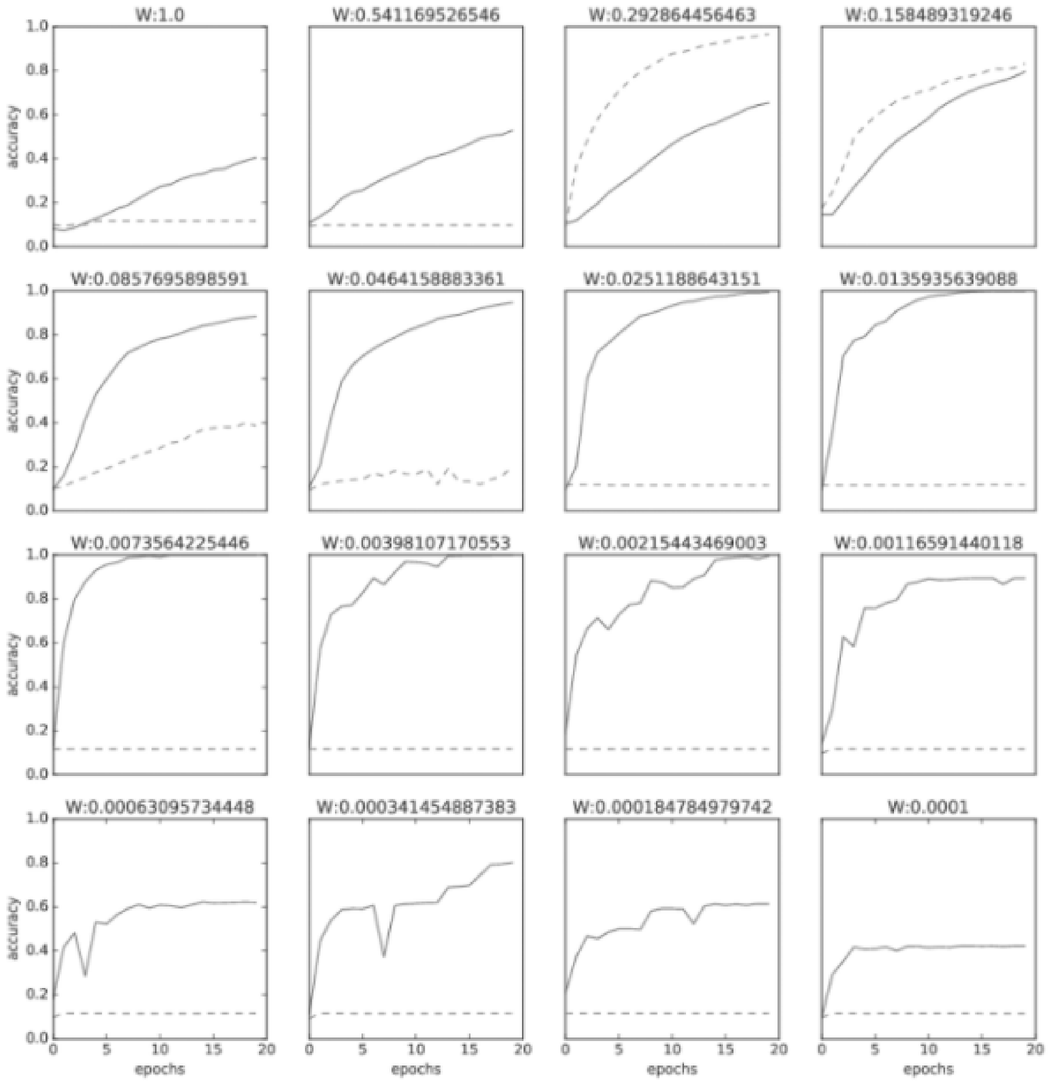
배치 정규화의 빠른 속도

→ 높은 학습률을 부여할 수 있음

→ 드롭아웃과 같은 규제를 적용하지 않아도 됨

배치 정규화

② 배치 정규의 효과



가중치 초기값의 영향 감소

→ 원래 가중치 초기값을 계속해서 조절함

배치 정규화

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

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

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

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

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

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

summary

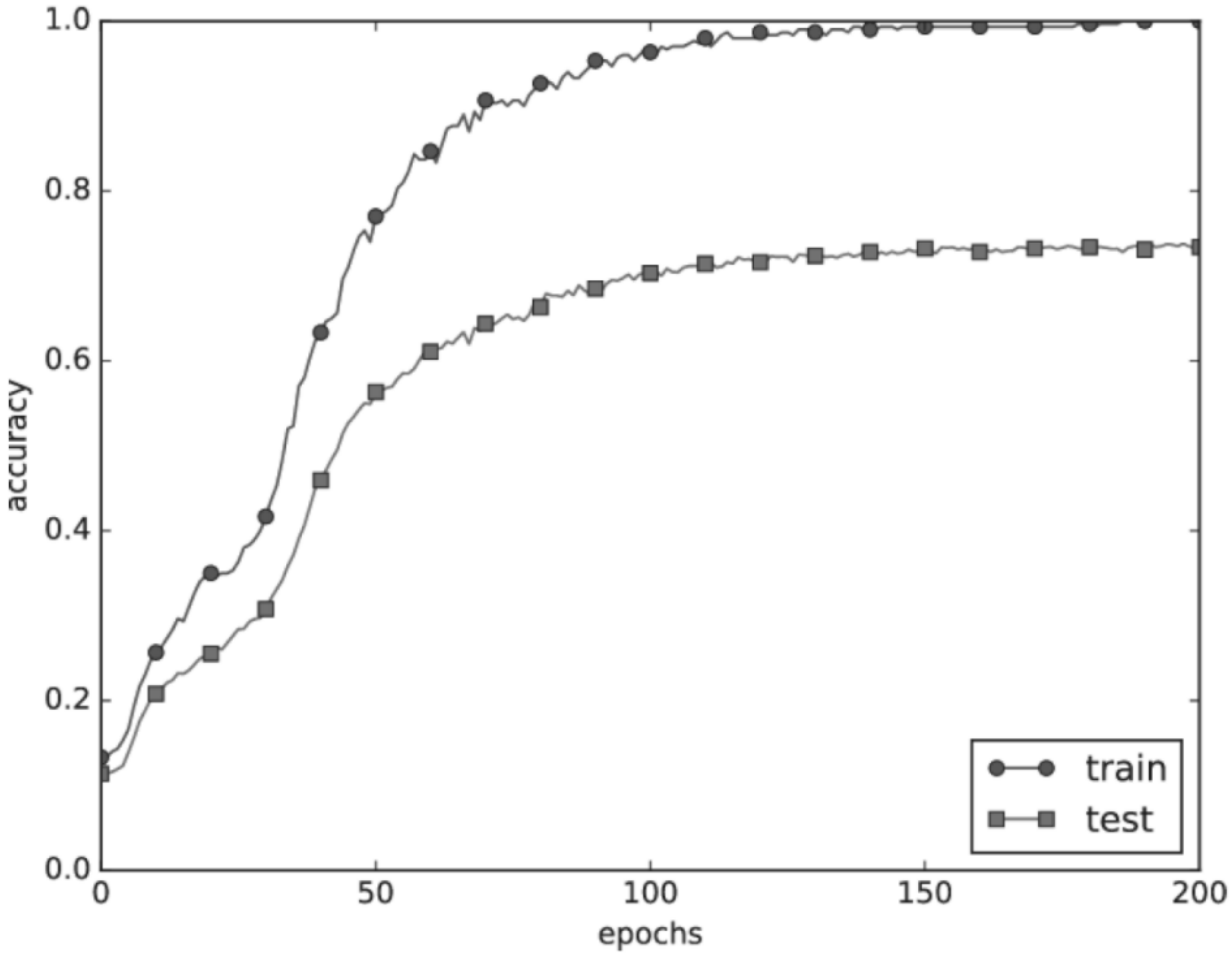
- 평균과 분산 계산
- 계산된 평균과 분산으로 정규화
- scale과 shift 지정

<장점>

- 학습 속도 개선
- 초기값에 크게 의존하지 않음
- 규제의 효과 → 오버피팅 억제
  (드롭아웃 등의 필요성 감소)

4. 바른 학습을 위해

바른 학습을 위해



오버피팅이란?

- 과적합
- 학습 데이터에 너무 과하게 학습이 됨
- 다른 데이터에 대한 정확도 감소

바른 학습을 위해

오버피팅이 일어나는 경우

- 매개변수가 많고 표현력이 높은 모델
- 훈련 데이터가 적음

→ 오버피팅 억제는 매우 중요. 아직 보지 못한 데이터를 바르게 식별하는 모델이 좋다.

바른 학습을 위해

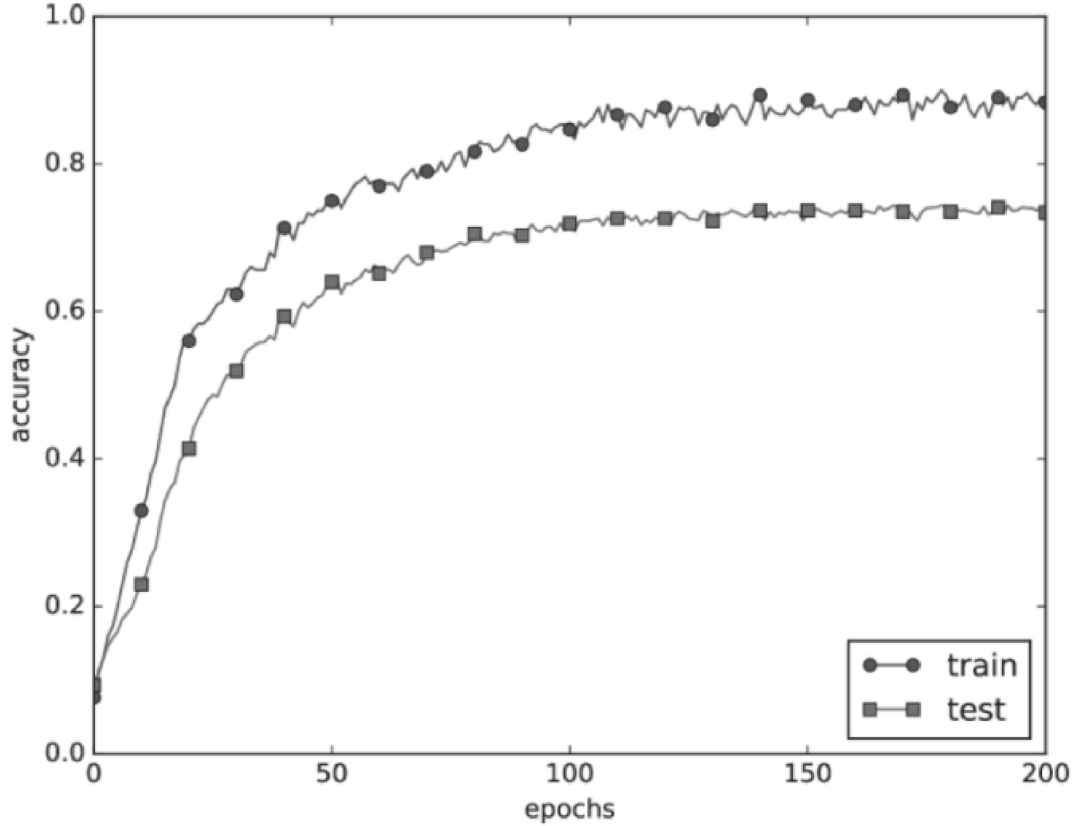
오버피팅을 억제하는 방법

1. 가중치 감소

: 큰 가중치에 대해서는 그에 상응하는

큰 페널티를 부과

→ 가중치 감소에는 규제 사용

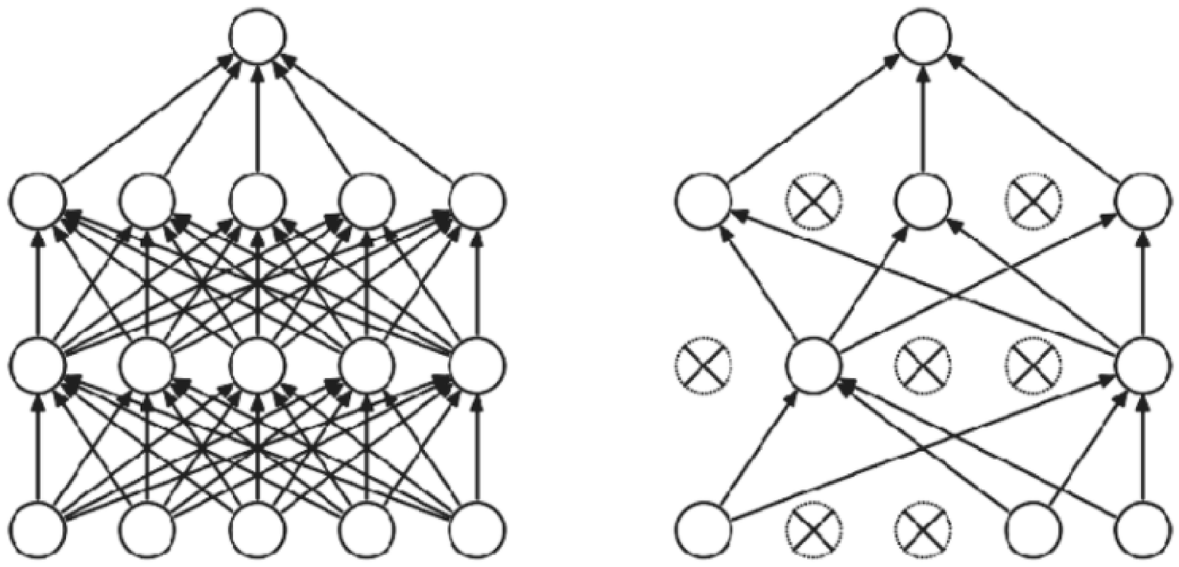


바른 학습을 위해

오버피팅을 억제하는 방법

2. 드롭아웃

: 뉴런을 임의로 삭제하면서 학습



(a) 일반 신경망 (b) 드롭아웃을 적용한 신경망

5. 적절한 하이퍼파라미터 값 찾기

적절한 하이퍼파라미터 값 찾기

- **하이퍼파라미터 :** 뉴런 수, 배치 크기, 학습률 등

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

적절한 하이퍼파라미터 값 찾기

하이퍼파라미터의 최적값이 존재하는 범위를 조금씩 줄여나감

Double check that the loss is reasonable:

```
def init_two_layer_model(input_size, hidden_size, output_size):
  # initialize a model
  model = {}
  model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
  model['b1'] = np.zeros(hidden_size)
  model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
  model['b2'] = np.zeros(output_size)
  return model
```

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 0.0)
print loss
```

disable regularization

```
2.30261216167
```

loss ~2.3.
“correct “ for
10 classes

returns the loss and the
gradient for all parameters

적절한 하이퍼파라미터 값 찾기

Lets try to train now…

Tip: Make sure that you can overfit very small portion of the training data

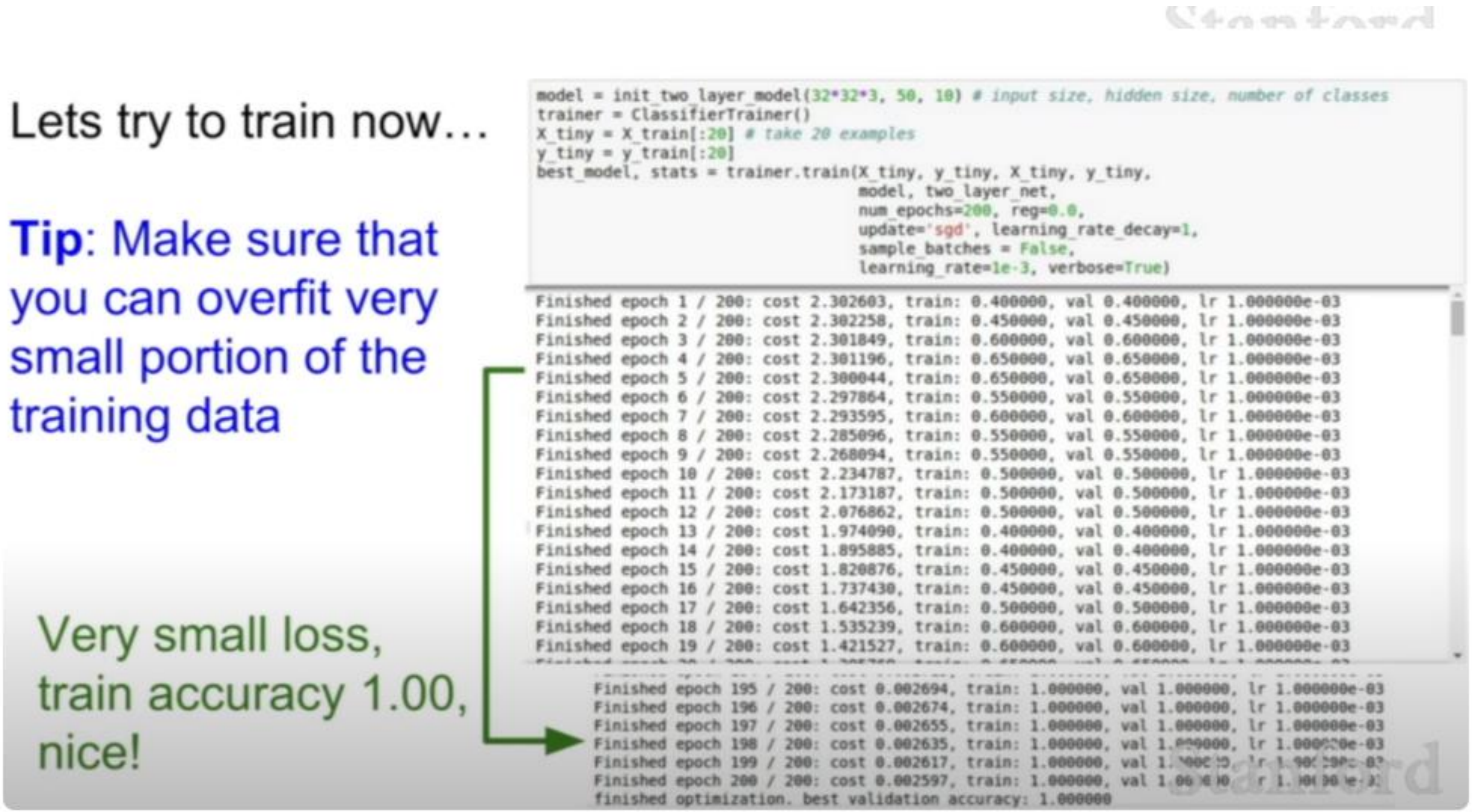
```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

The above code:

- take the first 20 examples from CIFAR-10
- turn off regularization (reg = 0.0)
- use simple vanilla ‘sgd’

적절한 하이퍼파라미터 값 찾기

② 하이퍼파라미터 최적화



적절한 하이퍼파라미터 값 찾기

Lets try to train now…

Start with small regularization and find learning rate that makes the loss go down.

loss not going down:
learning rate too low
loss exploding:
learning rate too high

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e6, verbose=True)
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:50: RuntimeWarning: divide by zero en
countered in log
  data_loss = -np.sum(np.log(probs[range(N), y])) / N
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:48: RuntimeWarning: invalid value enc
ountered in subtract
  probs = np.exp(scores - np.max(scores, axis=1, keepdims=True))
Finished epoch 1 / 10: cost nan, train: 0.091000, val 0.087000, lr 1.000000e+06
Finished epoch 2 / 10: cost nan, train: 0.095000, val 0.087000, lr 1.000000e+06
Finished epoch 3 / 10: cost nan, train: 0.100000, val 0.087000, lr 1.000000e+06
```

cost: NaN almost always means high learning rate...

Stanford

적절한 하이퍼파라미터 값 찾기

Lets try to train now…

Start with small regularization and find learning rate that makes the loss go down.

loss not going down:
learning rate too low
loss exploding:
learning rate too high

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=3e-3, verbose=True)

Finished epoch 1 / 10: cost 2.186654, train: 0.308000, val 0.306000, lr 3.000000e-03
Finished epoch 2 / 10: cost 2.176230, train: 0.330000, val 0.350000, lr 3.000000e-03
Finished epoch 3 / 10: cost 1.942257, train: 0.376000, val 0.352000, lr 3.000000e-03
Finished epoch 4 / 10: cost 1.827868, train: 0.329000, val 0.310000, lr 3.000000e-03
Finished epoch 5 / 10: cost inf, train: 0.128000, val 0.128000, lr 3.000000e-03
Finished epoch 6 / 10: cost inf, train: 0.144000, val 0.147000, lr 3.000000e-03
```

3e-3 is still too high. Cost explodes….

=> Rough range for learning rate we should be cross-validating is somewhere [1e-3 … 1e-5]

적절한 하이퍼파라미터 값 찾기

For example: run coarse search for 5 epochs

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)

      trainer = ClassifierTrainer()
      model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
      trainer = ClassifierTrainer()
      best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                   model, two_layer_net,
                                   num_epochs=5, reg=reg,
                                   update='momentum', learning_rate_decay=0.9,
                                   sample_batches = True, batch_size = 100,
                                   learning_rate=lr, verbose=False)
```

note it's best to optimize in log space!

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

nice

Stanford

적절한 하이퍼파라미터 값 찾기

Now run finer search...

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

adjust range

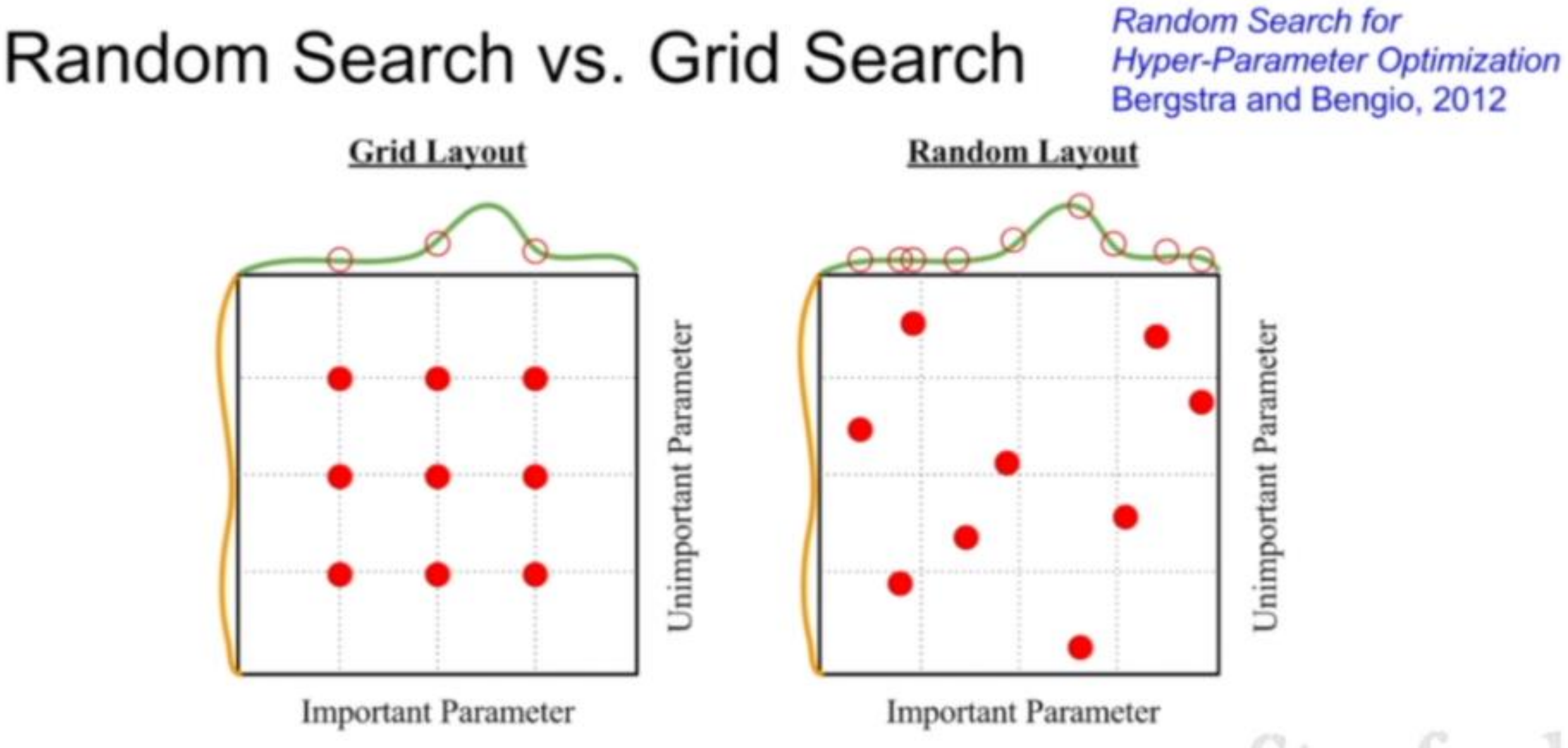
```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good for a 2-layer neural net with 50 hidden neurons.

But this best cross-validation result is worrying. Why?

적절한 하이퍼파라미터 값 찾기



The End