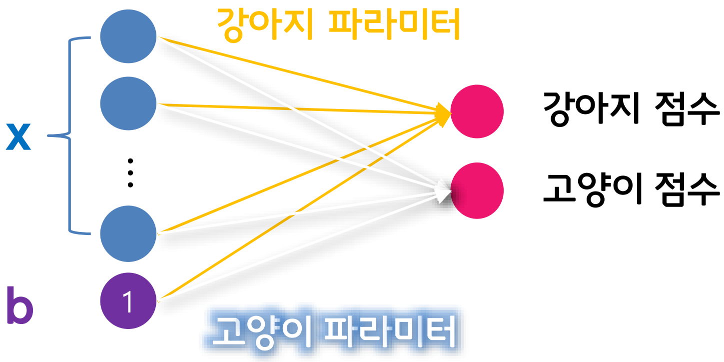
지난시간 Review

모두의연구소
박은수 Research Director

돌아보기 ...



3,072x1

$$2\text{x}1 \quad f(x, W) = Wx + b \quad 2\text{x}1$$

2x3,072

32x32x3 영상 =
3,072 픽셀

강아지 파라미터	0.2	0.1	...	0.3	0.7
고양이 파라미터	0.7	0.8	...	0.9	0.4

W

x

155
200
...
110
78

x

\+

0.1
0.2

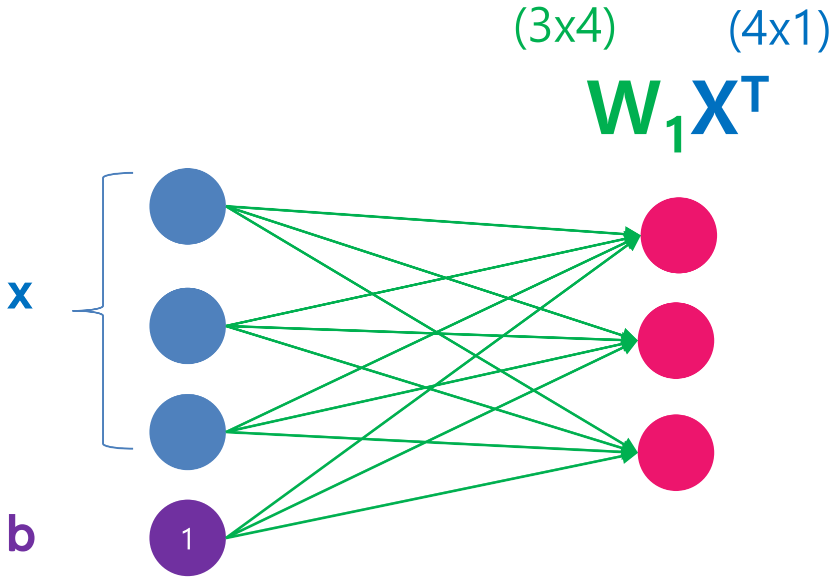
b

=

0.3	강아지 점수
0.7	고양이 점수

선형분류기

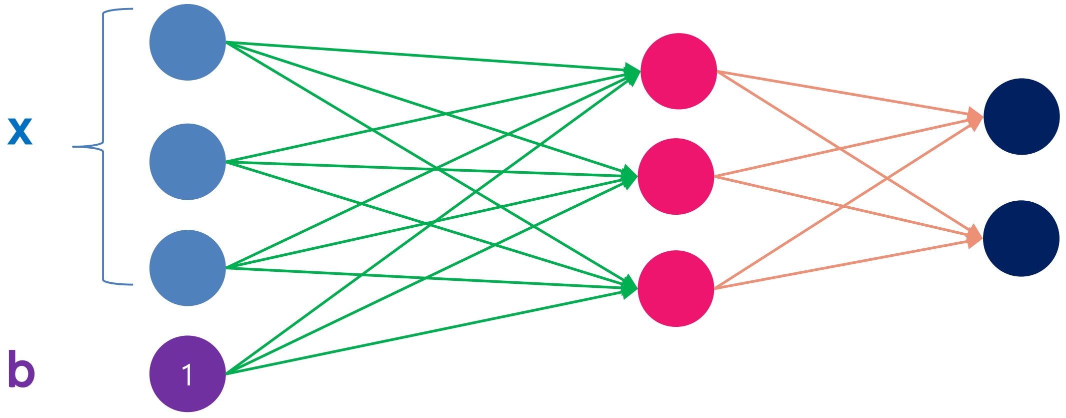
돌아보기 ..



돌아보기 ..

$$\underset{(2\text{x}3)}{W_2}\ \underset{(3\text{x}4)}{W_1}\ \underset{(4\text{x}1)}{X^T}$$

레이어추가



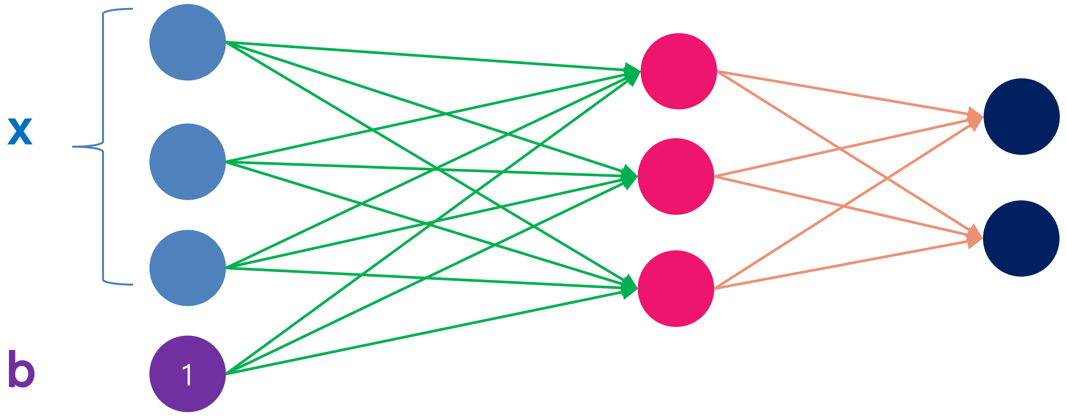
돌아보기 ..

핫 ~

$$\underset{(2\times3)}{W_2}\ \underset{(3\times4)}{W_1} = \underset{(2\times4)}{W}$$

$$\left[\underset{(2\times3)}{W_2}\ \underset{(3\times4)}{W_1}\right]\underset{(4\times1)}{X^T}$$

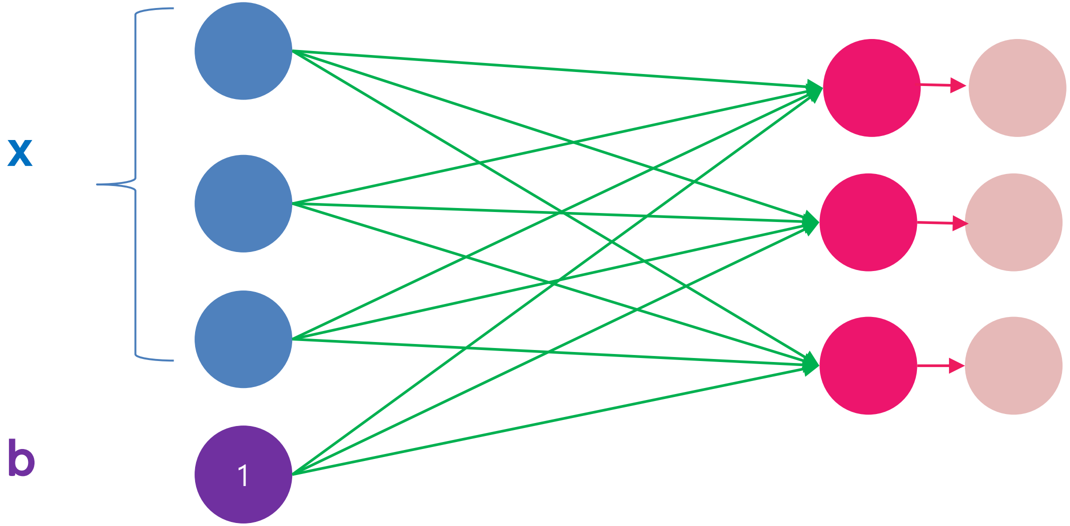
- 두개의 레이어가 하나로 표현
- 레이어를 쌓는 효과가 없어짐



비선형 연산이 필요

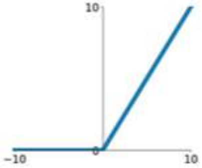
돌아보기 ..

$\max(0, W_1X^T)$



ReLU

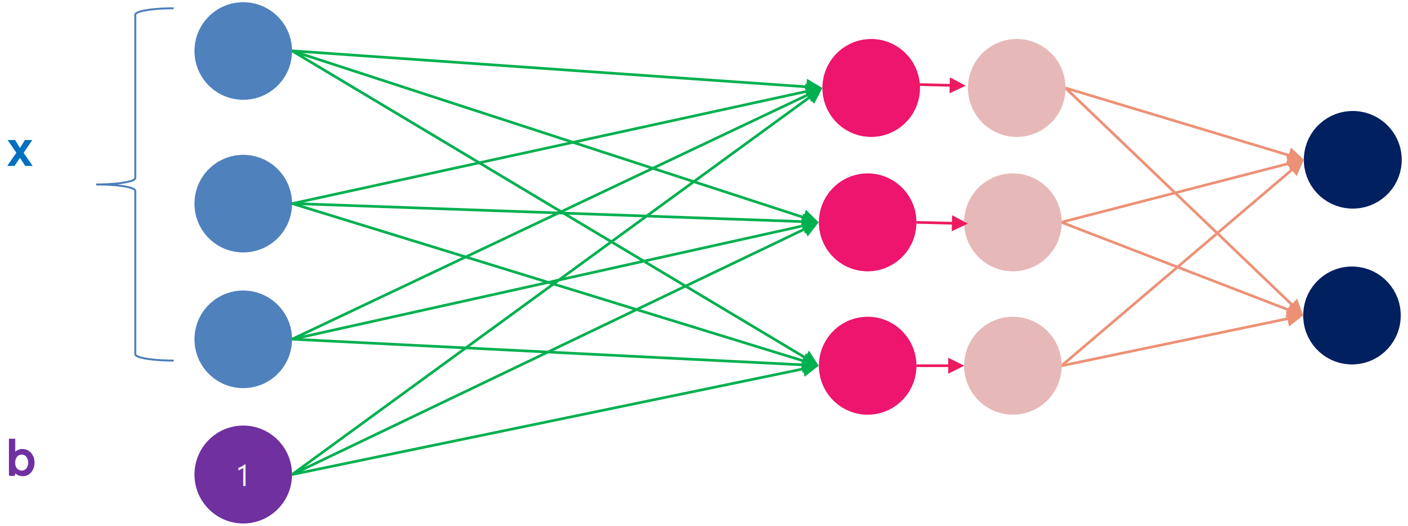
$\max(0, x)$



돌아보기 ..

레이어 추가

$$W_2 \times \max(\,0\,,\, W_1 X^T)$$

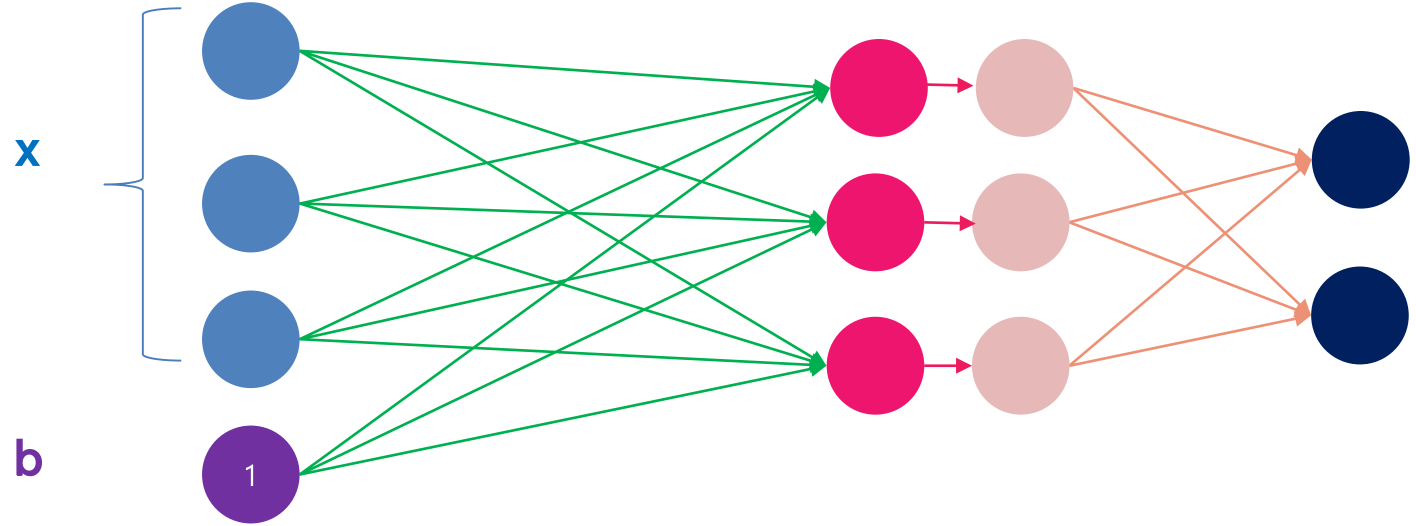


돌아보기 ..

1. Score function

레이어 추가

$$W_2 \times \max(\ 0\ ,\ W_1 X^T)$$

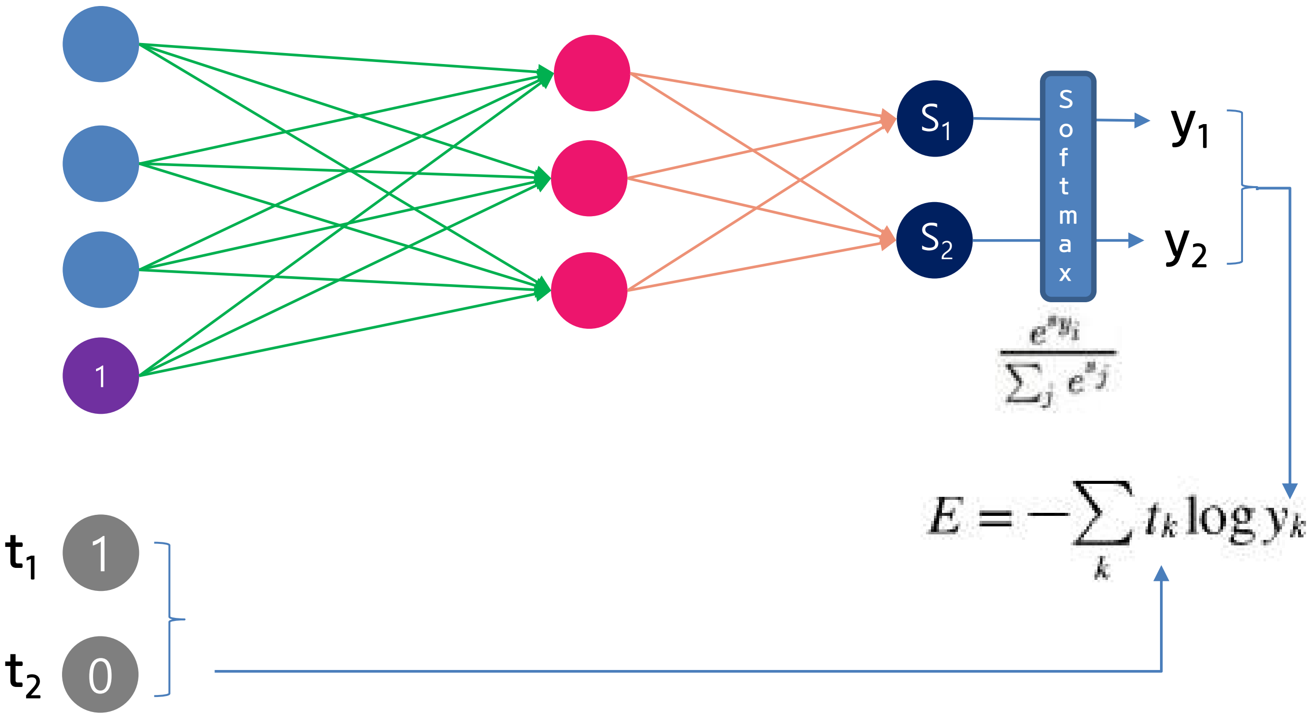


돌아보기 ..

1. Score function
2. Loss function

Loss function

모두의연구소

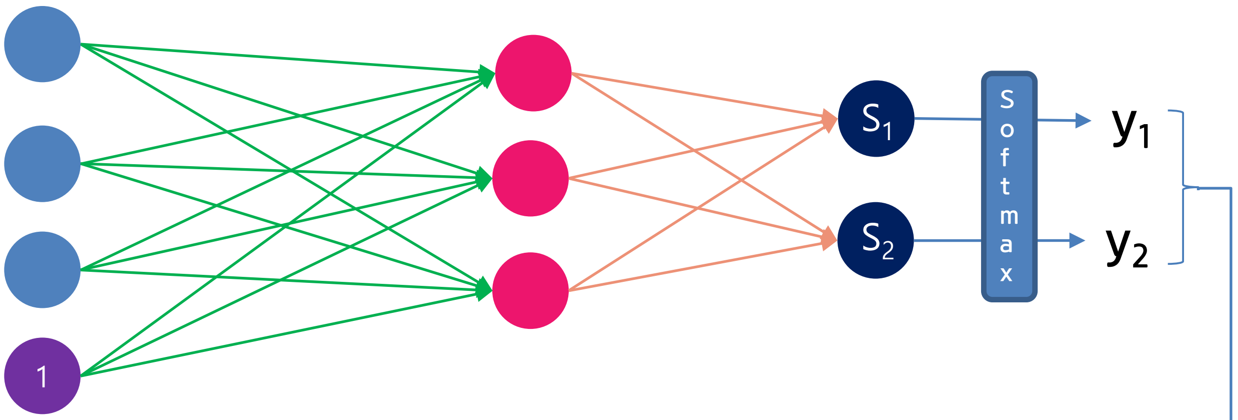


$$E = -\sum_k t_k \log y_k$$

Cross entropy
(Softmax loss)

돌아보기 ..

1. Score function
2. Loss function
3. Optimization



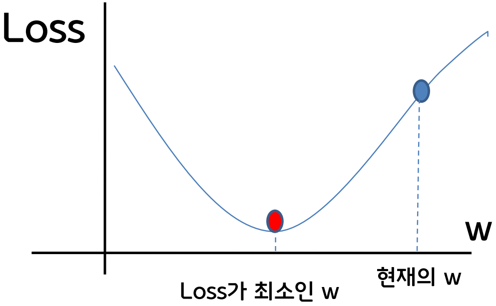
Loss를 줄이는 방법

Gradient Descent

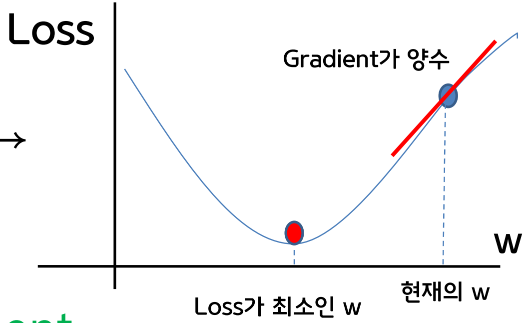
$$Loss \leftarrow E = -\sum_k t_k \log y_k$$

Cross entropy
(Softmax loss)

Gradient Descent



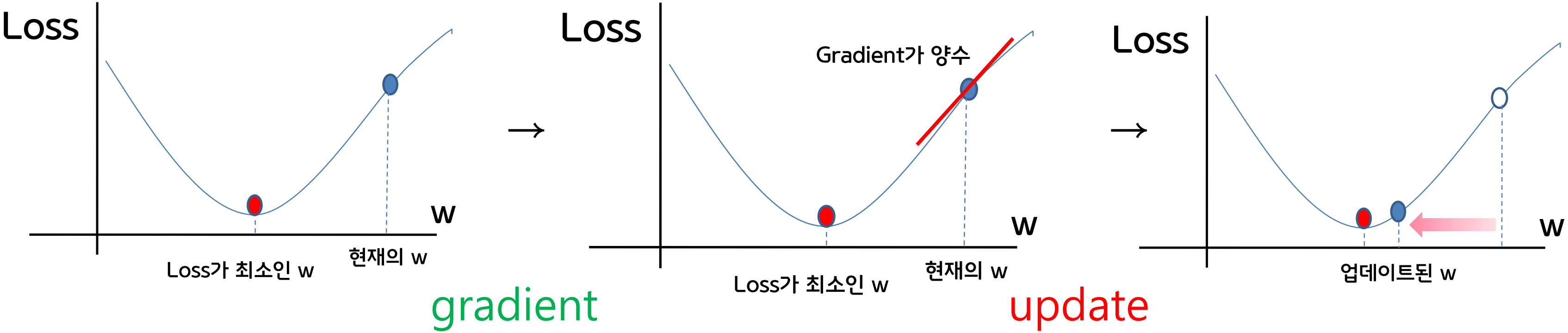
→



gradient

$$w = w - \eta \frac{\partial L}{\partial w}$$

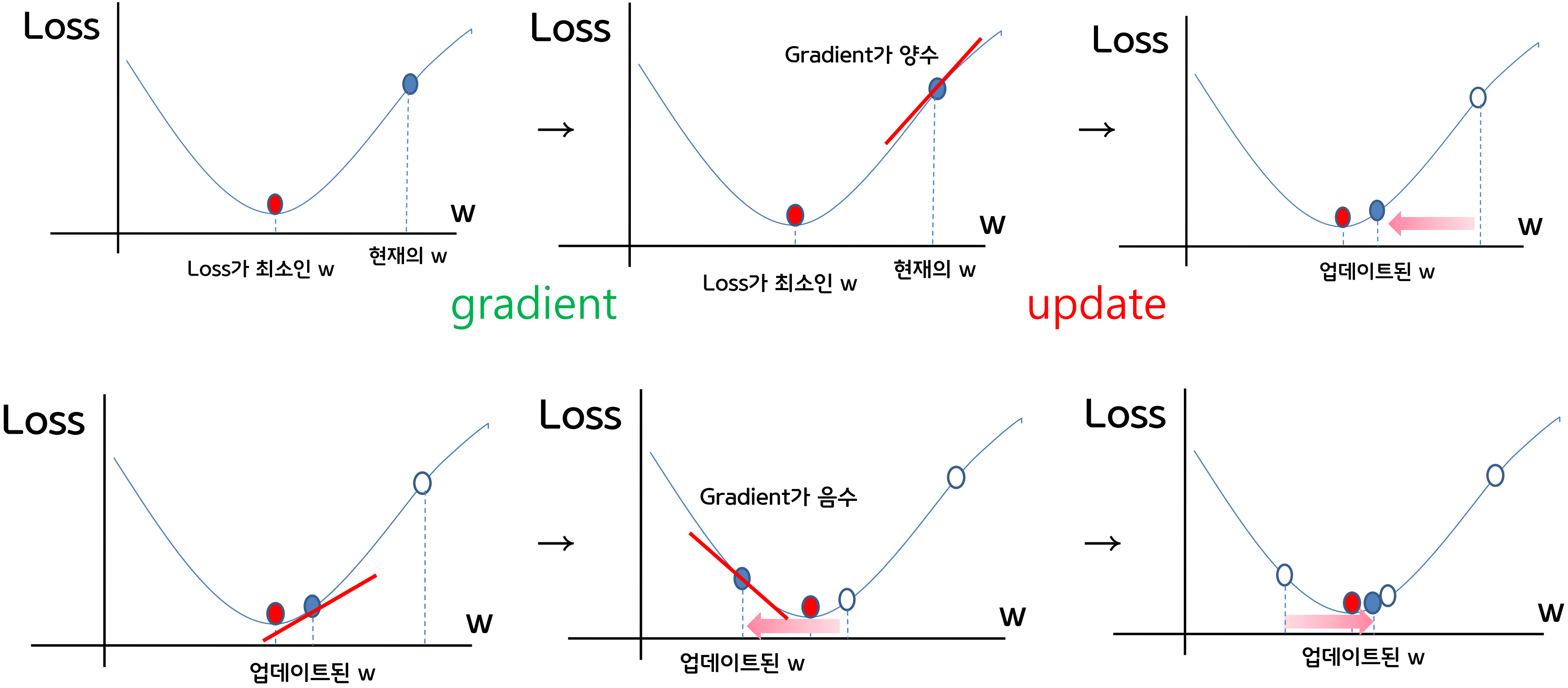
Gradient Descent



$$w = w - \eta \frac{\partial L}{\partial w}$$

Gradient Descent :

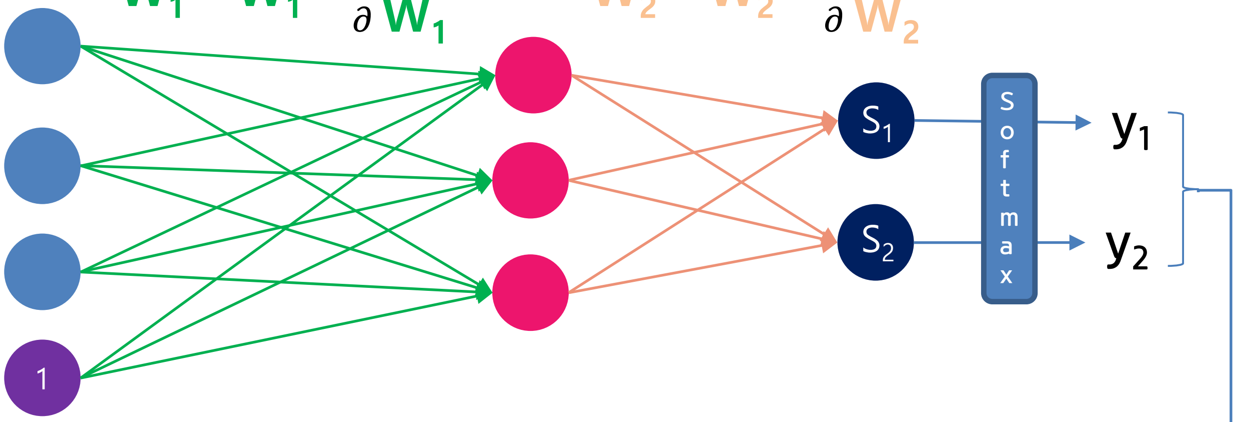
$$w = w - \eta \frac{\partial L}{\partial w}$$



1. Score function
2. Loss function
3. Optimization

돌아보기 ..

$W_1 = W_1 - \frac{\partial Loss}{\partial W_1}$ $W_2 = W_2 - \frac{\partial Loss}{\partial W_2}$



Loss를 줄이는 방법

$Loss \leftarrow E = -\sum_k t_k \log y_k$

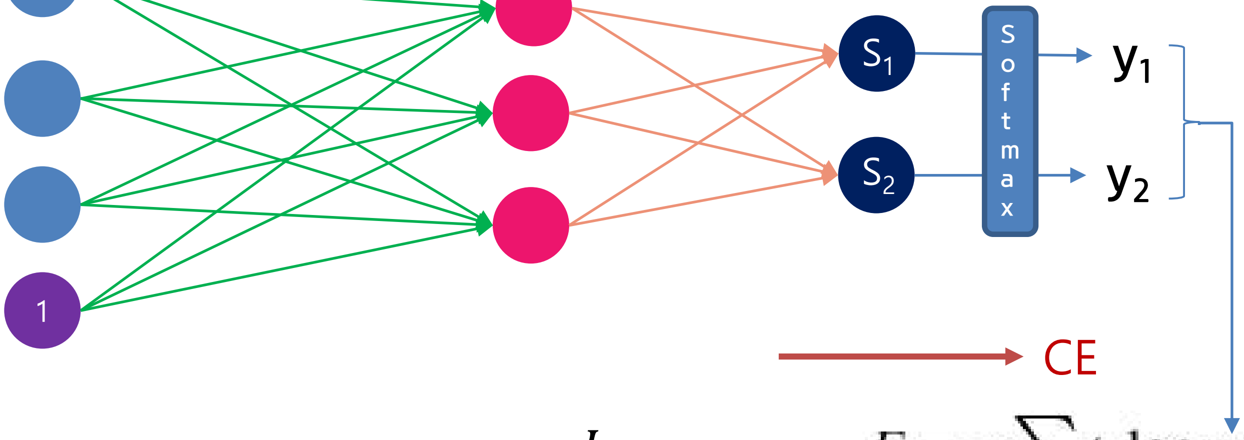
Cross entropy
(Softmax loss)

Gradient 계산방법 : Backpropagation

1. Score function
2. Loss function
3. Optimization

돌아보기 ..

$W_1 = W_1 - \frac{\partial Loss}{\partial W_1}$ $\qquad$ $W_2 = W_2 - \boxed{\frac{\partial Loss}{\partial W_2}}$



$$Loss \leftarrow E = -\sum_k t_k \log y_k$$

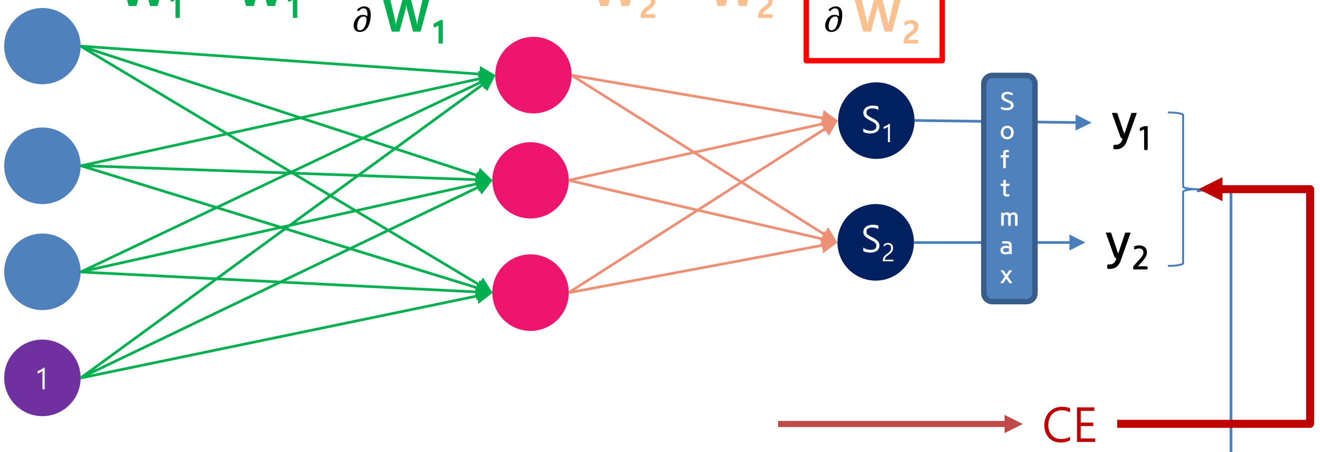
Cross entropy
(Softmax loss)

$$\frac{\partial Loss}{\partial W_2} \rightarrow \frac{\partial Loss}{\partial CE}$$

1. Score function
2. Loss function
3. Optimization

돌아보기 ..

$$W_1 = W_1 - \frac{\partial Loss}{\partial W_1} \qquad W_2 = W_2 - \boxed{\frac{\partial Loss}{\partial W_2}}$$



$$Loss \leftarrow E = -\sum_k t_k \log y_k$$

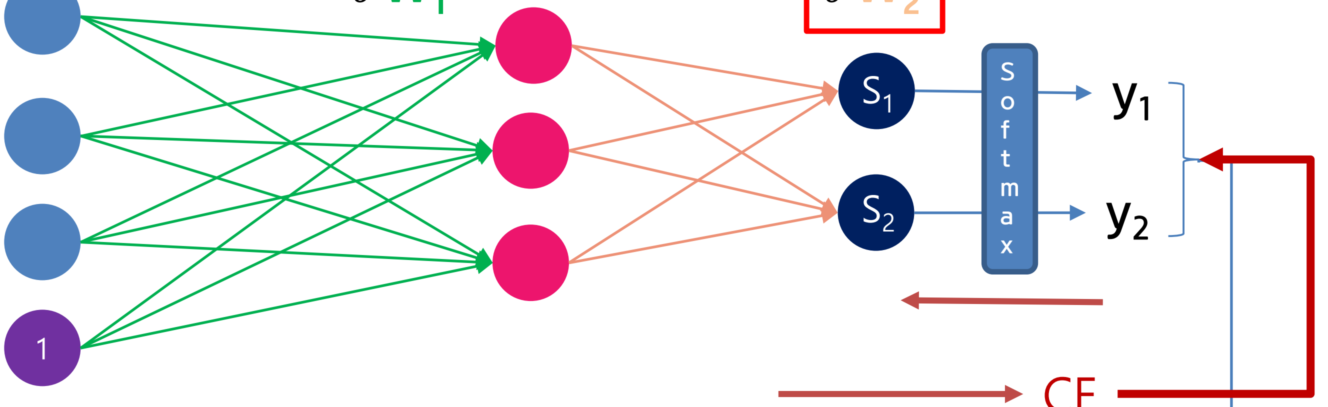
Cross entropy
(Softmax loss)

$$\frac{\partial Loss}{\partial W_2} \rightarrow \frac{\partial Loss}{\partial CE}\frac{\partial CE}{\partial y}$$

1. Score function
2. Loss function
3. Optimization

돌아보기 ..

$$W_1 = W_1 - \frac{\partial Loss}{\partial W_1} \qquad W_2 = W_2 - \boxed{\frac{\partial Loss}{\partial W_2}}$$



$$Loss \leftarrow E = -\sum_k t_k \log y_k$$

Cross entropy
(Softmax loss)

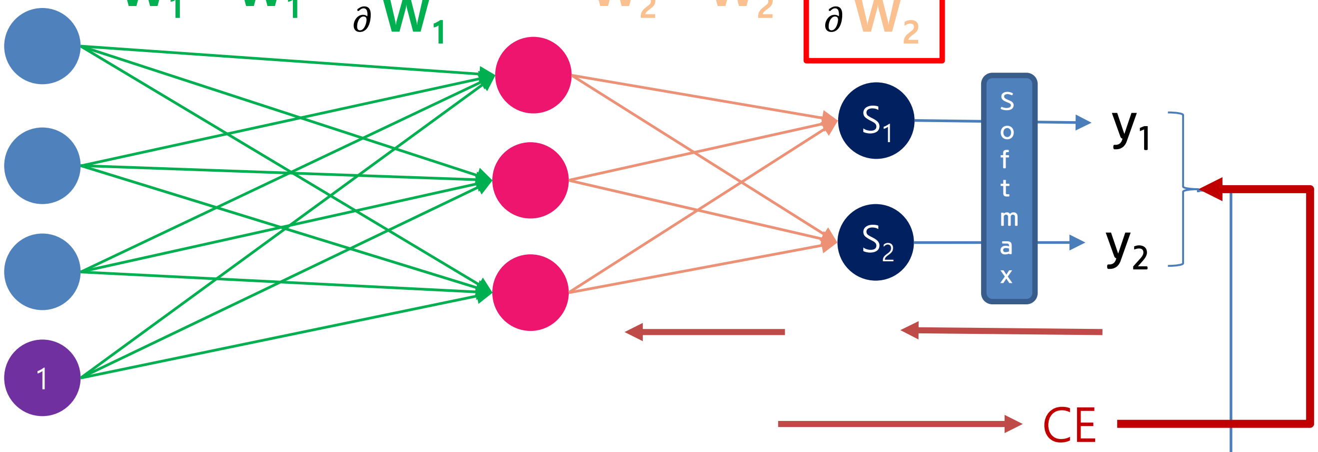
$$\frac{\partial Loss}{\partial W_2} \rightarrow \frac{\partial Loss}{\partial CE} \frac{\partial CE}{\partial y} \frac{\partial y}{\partial \,\mathbf{softmax}}$$

1. Score function
2. Loss function
3. Optimization

돌아보기 ..

모두의연구소

$$W_1 = W_1 - \frac{\partial Loss}{\partial W_1} \qquad W_2 = W_2 - \boxed{\frac{\partial Loss}{\partial W_2}}$$



$$Loss \leftarrow E = -\sum_k t_k \log y_k$$

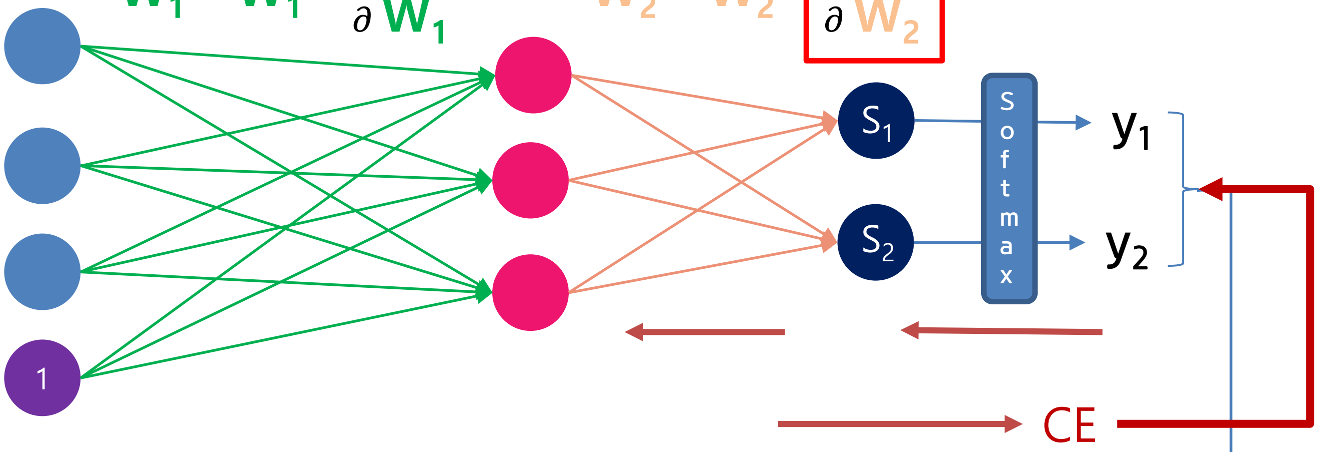
Cross entropy
(Softmax loss)

$$\frac{\partial Loss}{\partial W_2} \rightarrow \frac{\partial Loss}{\partial CE}\frac{\partial CE}{\partial y}\frac{\partial y}{\partial softmax}\frac{\partial softmax}{\partial \mathbf{s}}$$

1. Score function
2. Loss function
3. Optimization

돌아보기 ..

$$W_1 = W_1 - \frac{\partial Loss}{\partial W_1} \qquad W_2 = W_2 - \boxed{\frac{\partial Loss}{\partial W_2}}$$



$$Loss \leftarrow E = -\sum_k t_k \log y_k$$

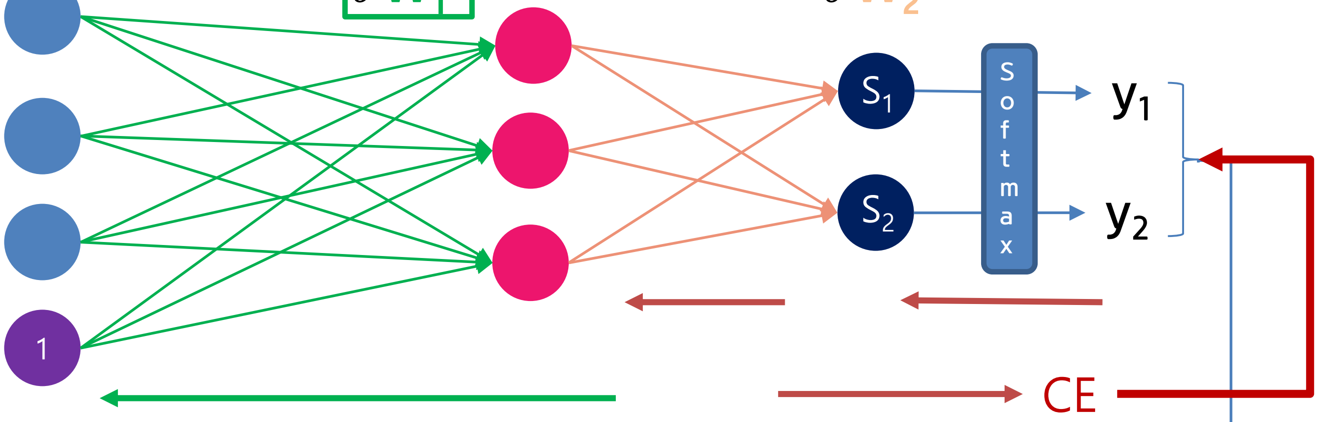
Cross entropy
(Softmax loss)

$$\frac{\partial Loss}{\partial W_2} \rightarrow \frac{\partial Loss}{\partial CE}\frac{\partial CE}{\partial y}\frac{\partial y}{\partial softmax}\frac{\partial softmax}{\partial s}\frac{\partial s}{\partial W_2}$$

1. Score function
2. Loss function
3. Optimization

돌아보기 ..

$W_1 = W_1 - \boxed{\frac{\partial Loss}{\partial W_1}}$ $\quad$ $W_2 = W_2 - \frac{\partial Loss}{\partial W_2}$



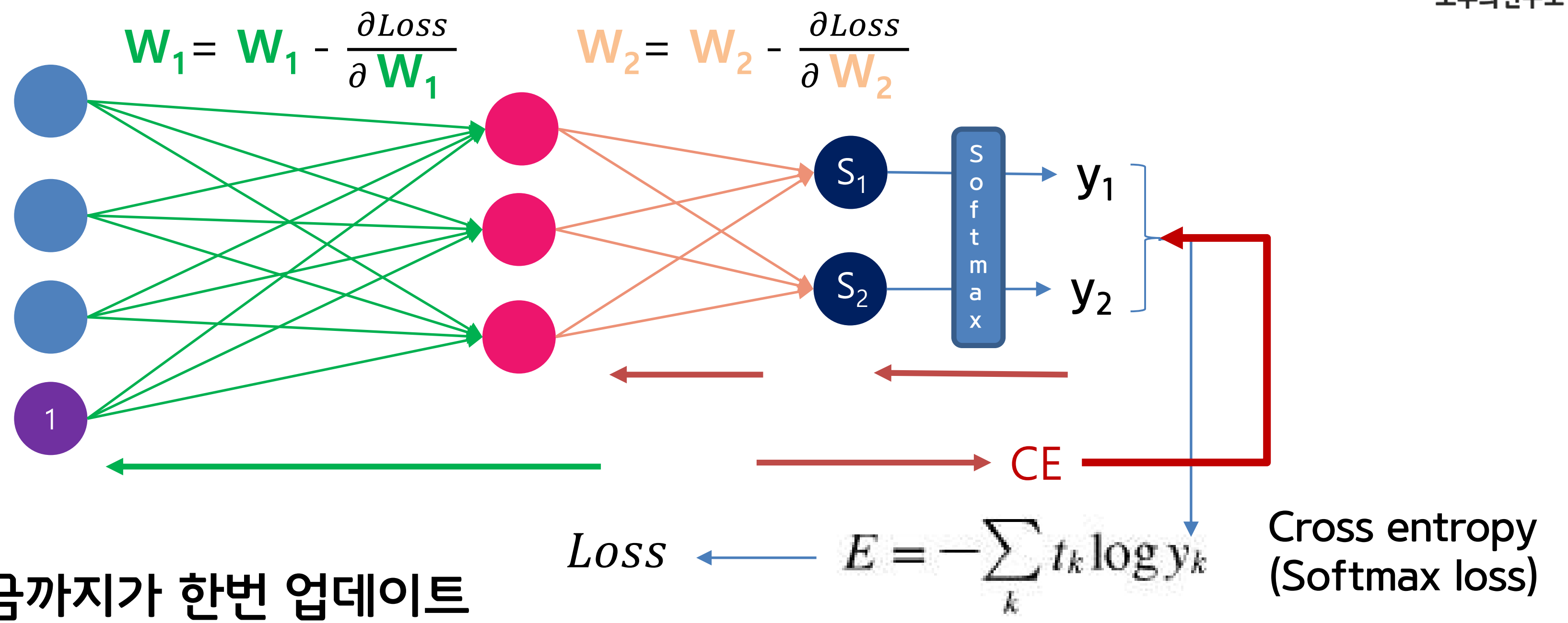
$Loss \leftarrow E = -\sum_k t_k \log y_k$

Cross entropy
(Softmax loss)

$$\frac{\partial Loss}{\partial W_1} \rightarrow \frac{\partial Loss}{\partial CE}\frac{\partial CE}{\partial y}\frac{\partial y}{\partial softmax}\frac{\partial softmax}{\partial s}\frac{\partial s}{\partial relu}\frac{\partial relu}{\partial k}\frac{\partial k}{\partial W_1}$$

1. Score function
2. Loss function
3. Optimization

돌아보기 ..



- 지금까지가 한번 업데이트
- 모든 이미지에 대해서 반복

Training Neural Networks

모두의연구소
박은수 Research Director

Gradient Descent

우리가 지금까지 Optimization 에서 초점을 둔 부분

$$w = w - \eta \frac{\partial L}{\partial w}$$

Gradient 의 계산

Gradient Descent

이제 부터는 Optimizer를 공부해 봅니다

$$w = w - \eta \frac{\partial L}{\partial w}$$

목적 : 그레디언트를 좀 더 효율적으로 변형한 뒤 업데이트

- Optimizer : 최적화를 행하는 자

매개변수 갱신

- **확률적 경사하강법 복습 (Stochastic Gradient Descent; SGD)**

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

  - /common/optimizer.py

```python
# coding: utf-8
import numpy as np

class SGD:

    """확률적 경사 하강법(Stochastic Gradient Descent)"""

    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]
```

Learning rate

매개변수 갱신

- 기본 Optimizer : SGD
- **common/optimizer.py**

```
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)
    ...
```

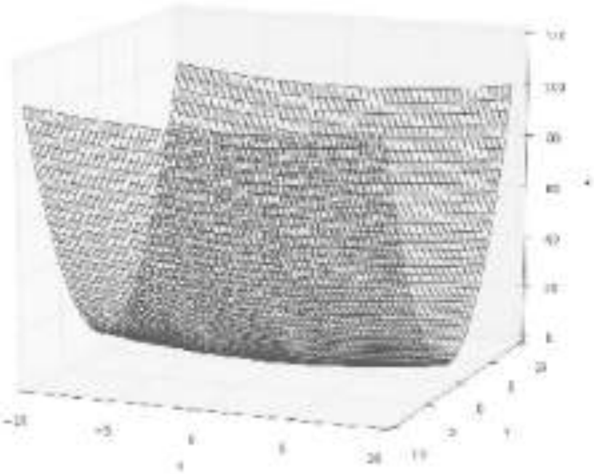
Gradient만을 변형할 것이기 때문에 Network와 별개로 존재할 수 있습니다.

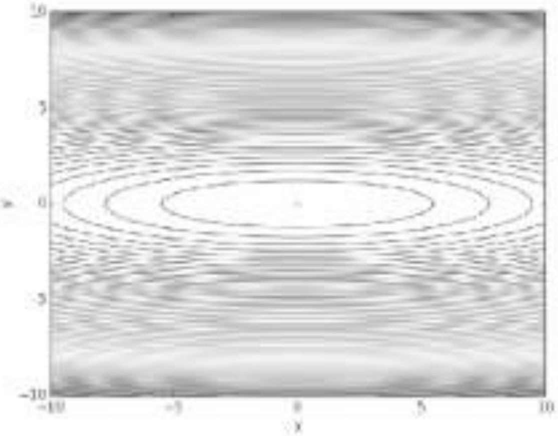
*대부분의 딥러닝 프레임워크는 이렇게 구성되어 있습니다

매개변수 갱신

- SGD의 단점

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



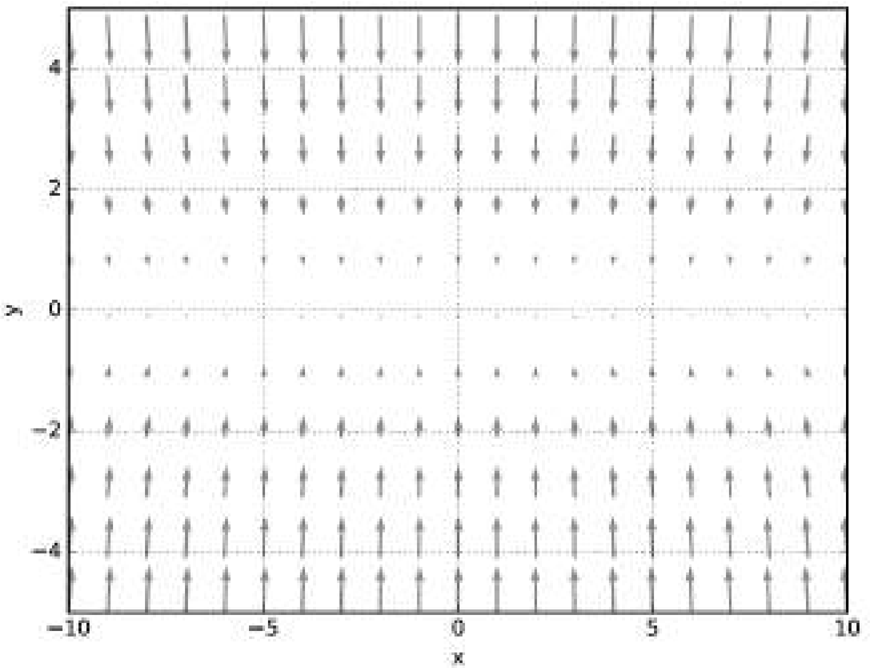


기울기를 그려보면

매개변수 갱신

- SGD의 단점

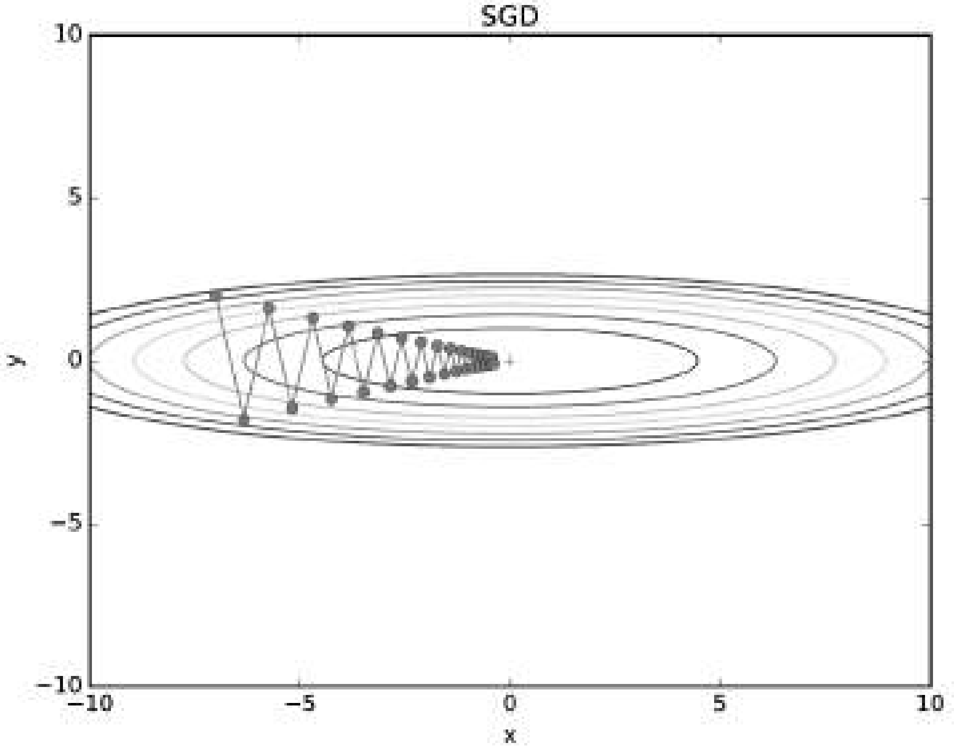
$$f(x,y) = \frac{1}{20}x^2 + y^2$$



- y축 방향은 크고 x축 방향은 작습니다
- 최소값은 (0,0)이지만 대부분 (0,0)을 가리키지 못합니다
- (x, y) = (−7, 2)에서 시작해서 SGD를 수행해 봅니다

매개변수 갱신

- SGD의 단점



- 탐색경로가 비효율적 입니다
- y축 기울기는 크고 x축 기울기는 매우 작고
- 제대로 최솟값을 가르키지 못하고 있어서 입니다

개선이 필요합니다

매개변수 갱신

- Optimizer 1 : 모멘텀 (Momentum)
  - '운동량'을 뜻하는 단어로 물리와 관계가 있습니다

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

→ 기울기 방향으로 힘을 받아 물체가 가속된다는 물리 법칙을 나타냄

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

- **W** : 갱신 할 가중치
- $\frac{\partial L}{\partial \mathbf{W}}$: 손실함수 기울기
- η : 학습율
- **v** : 물리에서의 속도(velocity)

매개변수 갱신

- **모멘텀 (Momentum)**
  - **'운동량' 을 뜻하는 단어로 물리와 관계가 있습니다**

물리에서의 지면 마찰이나 공기 저항에 해당합니다. 보통 0.9

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

기울기 방향으로 힘을 받아 물체가 가속된다는 물리 법칙을 나타냄

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

- **W** : 갱신 할 가중치
- $\frac{\partial L}{\partial \mathbf{W}}$: 손실함수 기울기
- η : 학습율
- $\mathbf{v}$ **:** 물리에서의 속도(velocity)

매개변수 갱신

- 모멘텀 (Momentum)
  - 구현

```python
class Momentum:

    """모멘텀 SGD"""

    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, 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.keys():
            self.v[key] = self.momentum*self.v[key] - self.lr*grads[key]
            params[key] += self.v[key]
```

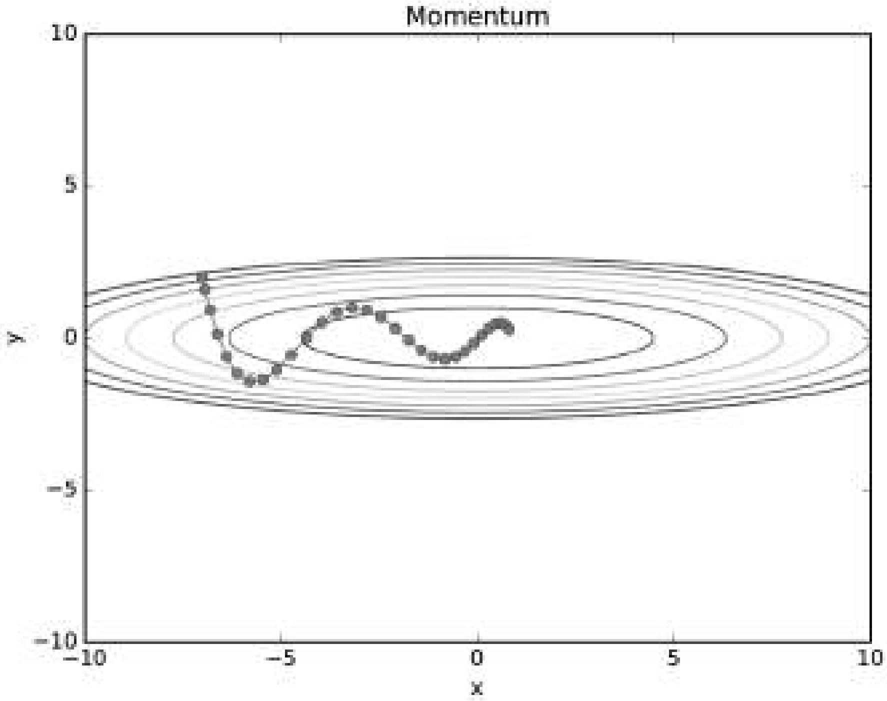
초기값 0

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

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

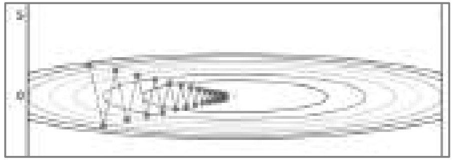
매개변수 갱신

- 모멘텀 (Momentum)



- 공이 그릇 바닥으 구르듯 움직입니다

- 전체적으로 지그재그가 SGD에 비해 덜합니다



SGD

매개변수 갱신

- 모멘텀 (Momentum)
  - 갑자기 물리 얘기 나오고 공 굴러가고 대체 뭔 소리에요?

매개변수 갱신

- 모멘텀 (Momentum)
  - 갑자기 물리 얘기 나오고 공 굴러가고 대체 뭔 소리에요?

- v의 변화를 살펴 봅시다
  - v_0는 0 부터 시작

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

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

관측

K_i로 변경
(그레디언트임)

업데이트 1) $v_1 \longleftarrow \alpha * 0 - K_o \ : -K_o$
업데이트 2) $v_2 \longleftarrow \alpha v_1 - K_1 \ : -\alpha K_o - K_1$
업데이트 3) $v_3 \longleftarrow \alpha v_2 - K_2 \ : -\alpha^2 K_o - \alpha K_1 - K_2$
업데이트 4) $v_4 \longleftarrow \alpha v_3 - K_3 \ : -\alpha^3 K_o - \alpha^2 K_1 - \alpha K_2 - K_3$

⋮

- 1) W ← W + v_1
- 2) W ← W + v_2
- 3) W ← W + v_3
- 4) W ← W + v_4

모두의연구소

매개변수 갱신

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

관측

K_i로 변경 (그레디언트임)

- **모멘텀 (Momentum)**

업데이트 1) $v_1 \leftarrow \alpha * 0 - K_o \ : -K_o$
업데이트 2) $v_2 \leftarrow \alpha v_1 - K_1 \ : -\alpha K_o - K_1$
업데이트 3) $v_3 \leftarrow \alpha v_2 - K_2 \ : -\alpha^2 K_o - \alpha K_1 - K_2$
업데이트 4) $v_4 \leftarrow \alpha v_3 - K_3 \ : -\alpha^3 K_o - \alpha^2 K_1 - \alpha K_2 - K_3$
$\vdots$

- 1) $W \leftarrow W + v_1$
- 2) $W \leftarrow W + v_2$
- 3) $W \leftarrow W + v_3$
- 4) $W \leftarrow W + v_4$

일반적인 경우처럼 α = 0.9로 하면 α^2= 0.81, α^3= 0.729

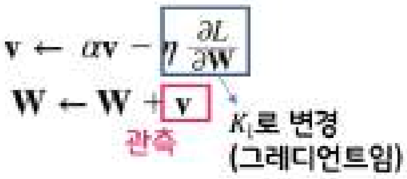
기존의 업데이트 값을 계속 반영하지만 점점 지수적으로 줄입니다

물리에서의 지면 마찰이나 공기 저항에 해당합니다. 보통 0.9

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

기울기 방향으로 힘을 받아 물체가 가속된다는 물리 법칙을 나타냄

매개변수 갱신

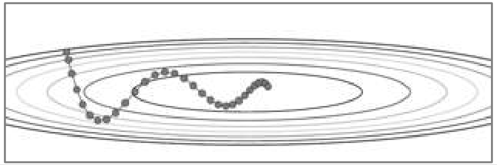


- **모멘텀 (Momentum)**

업데이트 1) $v_1 \longleftarrow \alpha * 0 - K_o \ : -K_o$
업데이트 2) $v_2 \longleftarrow \alpha v_1 - K_1 \ : -\alpha K_o - K_1$
업데이트 3) $v_3 \longleftarrow \alpha v_2 - K_2 \ : -\alpha^2 K_o - \alpha K_1 - K_2$
업데이트 4) $v_4 \longleftarrow \alpha v_3 - K_3 \ : -\alpha^3 K_o - \alpha^2 K_1 - \alpha K_2 - K_3$

- 1) $W \leftarrow W + v_1$
- 2) $W \leftarrow W + v_2$
- 3) $W \leftarrow W + v_3$
- 4) $W \leftarrow W + v_4$

- **지난 업데이트를 기억하고 이를 업데이트에 반영함과 동시에 과거의 값들의 영향력은 줄여 나갑니다**
- **업데이트 부호가 바뀌면 그 영향력이 줄어 들게 됩니다**



매개변수 갱신

- AdaGrad
  - AdaGrad는 '각각의' 매개변수에 맞게 '맞춤형'으로 매개변수를 갱신합니다
    - 적응적(adaptive)으로 학습률을 조절하면서 학습을 진행

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

기존 기울기값을 제곱하여 계속 더함

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

- $\odot$: 행렬의 원소별 곱셈을 의미함

매개변수 갱신

- AdaGrad
  - AdaGrad는 '각각의' 매개변수에 맞게 '맞춤형'으로 매개변수를 갱신합니다

```python
class AdaGrad:

    """AdaGrad"""

    def __init__(self, lr=0.01):
        self.lr = lr
        self.h = None

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

매개변수 갱신

- Adagrad

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

K_i로 변경 (Gradient임)

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

업데이트를 관측 해봅시다

업데이트 1) $\frac{1}{\sqrt{K_0^2}} K_o$

업데이트 2) $\frac{1}{\sqrt{K_1^2+K_0^2}} K_1$

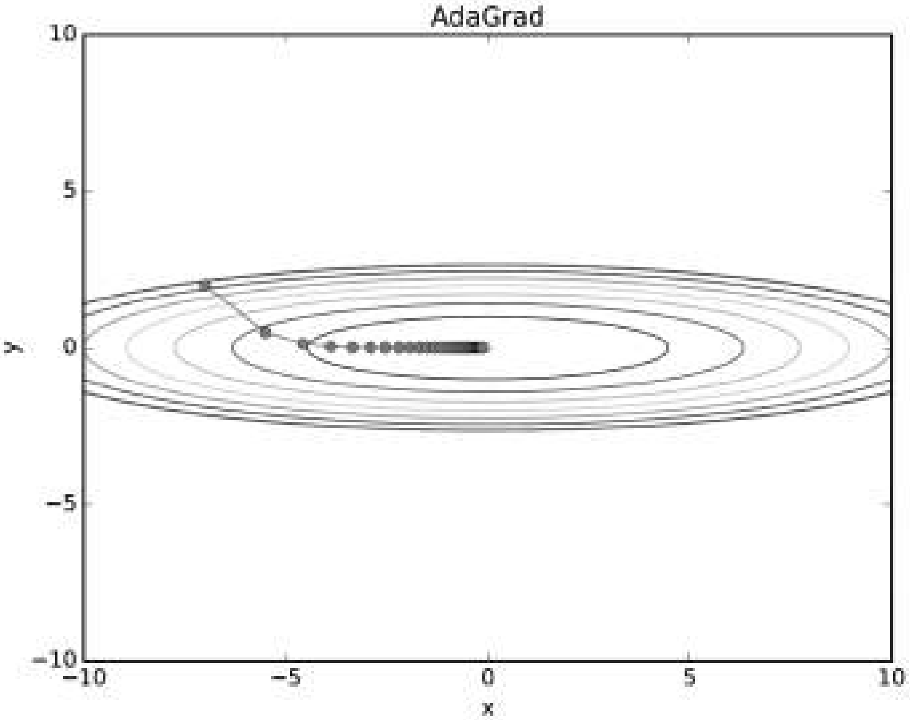
업데이트 3) $\frac{1}{\sqrt{K_3^2+K_1^2+K_0^2}} K_3$

업데이트 4) $\frac{1}{\sqrt{K_4^2+K_3^2+K_1^2+K_0^2}} K_4$

- 일변수가 아니라 실제로는 벡터로 작용해서 각 매개 변수 마다 업데이트 되는 크기가 다름을 잊지 마세요
- 업데이트를 어느 정도 안정된 값으로 하는 정규화 속성이 존재합니다
- 시간이 지날 수록 업데이트 값이 계속 줄어들어 나중에는 업데이트가 발생하지 않게 됩니다
  - 해결책 -> RMSprop

매개변수 갱신

- Adagrad



- 지그재그가 줄어들고 어느 정도 안정적으로 최솟값을 향해 갑니다

매개변수 갱신

모두의연구소

- RMSProp

RMSProp update

[Tieleman and Hinton, 2012]

```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

```
# RMSProp
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 30 25 Jan 2016

매개변수 갱신

- RMSProp

rmsprop: A mini-batch version of rprop

- rprop is equivalent to using the gradient but also dividing by the size of the gradient.
  - The problem with mini-batch rprop is that we divide by a different number for each mini-batch. So why not force the number we divide by to be very similar for adjacent mini-batches?
- rmsprop: Keep a moving average of the squared gradient for each weight

$$MeanSquare(w,t) = 0.9\ MeanSquare(w,\ t-1) + 0.1\left(\partial E/\partial w\,(t)\right)^2$$

- Dividing the gradient by $\sqrt{MeanSquare(w,\ t)}$ makes the learning work much better (Tijmen Tieleman, unpublished).

Introduced in a slide in Geoff Hinton's Coursera class, lecture 6

Cited by several papers as:

[52] T. Tieleman and G. E. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude., 2012.

매개변수 갱신

- RMSProp

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \boxed{\frac{\partial L}{\partial \mathbf{W}}}\right) \rightarrow \mathbf{K}_i\text{로 변경}$$

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

업데이트 1) $\mathbf{h}_1 = (1-\alpha)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = \alpha\mathbf{h}_1 + (1-\alpha)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = \alpha\mathbf{h}_2 + (1-\alpha)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = \alpha\mathbf{h}_3 + (1-\alpha)\mathbf{K}_4^2$

매개변수 갱신

- RMSProp

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \boxed{\frac{\partial L}{\partial \mathbf{W}}}\right) \rightarrow \mathbf{K}_i\text{로 변경}$$

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

a 가 1보다 작다면(보통 0.9) : 모멘텀과 같이 기존값들을 반영하면서 그 영향력을 지수적으로 줄인다

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = \alpha\mathbf{h}_1 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = \alpha\mathbf{h}_2 + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = \alpha\mathbf{h}_3 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- RMSProp

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \boxed{\frac{\partial L}{\partial \mathbf{W}}}\right) \longrightarrow \mathbf{K}_i\text{로 변경}$$

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

α가 1보다 작다면(보통 0.9) : 모멘텀과 같이 기존값들을 반영하면서 그 영향력을 지수적으로 줄인다

업데이트 1) $\mathbf{h}_1 = (1-\alpha)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = \alpha\mathbf{h}_1 + (1-\alpha)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = \alpha\mathbf{h}_2 + (1-\alpha)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = \alpha\mathbf{h}_3 + (1-\alpha)\mathbf{K}_4^2$

기존 그레디언트 값들의 누적하면서 영향력을 지속적으로 줄임

매개변수 갱신

- RMSProp

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \boxed{\frac{\partial L}{\partial \mathbf{W}}}\right) \rightarrow \mathbf{K}_i\text{로 변경}$$

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

a가 1보다 작다면(보통 0.9) : 모멘텀과 같이 기존값들을 반영하면서 그 영향력을 지수적으로 줄인다

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = \alpha\mathbf{h}_2 + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = \alpha\mathbf{h}_3 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- RMSProp

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \boxed{\frac{\partial L}{\partial \mathbf{W}}}\right) \longrightarrow \mathbf{K}_i\text{로 변경}$$

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

a가 1보다 작다면(보통 0.9) : 모멘텀과 같이 기존값들을 반영하면서 그 영향력을 지수적으로 줄인다

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = \alpha\mathbf{h}_2 + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = \alpha\mathbf{h}_3 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- RMSProp

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \boxed{\frac{\partial L}{\partial \mathbf{W}}}\right) \rightarrow \mathbf{K}_i\text{로 변경}$$

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

a가 1보다 작다면(보통 0.9) : 모멘텀과 같이 기존값들을 반영하면서 그 영향력을 지수적으로 줄인다

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = a(a\,(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2) + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = \alpha\mathbf{h}_3 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- RMSProp

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \boxed{\frac{\partial L}{\partial \mathbf{W}}}\right) \longrightarrow \mathbf{K}_i\text{로 변경}$$

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

a가 1보다 작다면(보통 0.9) : 모멘텀과 같이 기존값들을 반영하면서 그 영향력을 지수적으로 줄인다

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$

업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$

업데이트 3) $\mathbf{h}_3 = a^2(1-a)\mathbf{K}_1^2 + a(1-a)\mathbf{K}_2^2 + (1-a)\mathbf{K}_3^2$

업데이트 4) $\mathbf{h}_4 = \alpha\mathbf{h}_3 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- RMSProp

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \boxed{\frac{\partial L}{\partial \mathbf{W}}}\right) \rightarrow \mathbf{K}_i\text{로 변경}$$

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

a가 1보다 작다면(보통 0.9) : 모멘텀과 같이 기존값들을 반영하면서 그 영향력을 지수적으로 줄인다

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$

업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$

업데이트 3) $\mathbf{h}_3 = a^2(1-a)\mathbf{K}_1^2 + a(1-a)\mathbf{K}_2^2 + (1-a)\mathbf{K}_3^2$

업데이트 4) $\mathbf{h}_4 = a(a^2(1-a)\mathbf{K}_1^2 + a(1-a)\mathbf{K}_2^2 + (1-a)\mathbf{K}_3^2) + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- RMSProp

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \boxed{\frac{\partial L}{\partial \mathbf{W}}}\right) \longrightarrow \mathbf{K}_i\text{로 변경}$$

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

a가 1보다 작다면(보통 0.9) : 모멘텀과 같이 기존값들을 반영하면서 그 영향력을 지수적으로 줄인다

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = a^2(1-a)\mathbf{K}_1^2 + a(1-a)\mathbf{K}_2^2 + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = a^3(1-a)\mathbf{K}_1^2 + a^2(1-a)\mathbf{K}_2^2 + a(1-a)\mathbf{K}_3^2 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- **Momentum**

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

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

업데이트 1) $\mathbf{v}_1 \leftarrow \alpha * 0 - K_0 : -K_0$
업데이트 2) $\mathbf{v}_2 \leftarrow \alpha\mathbf{v}_1 - K_1 : -\alpha K_0 - K_1$
업데이트 3) $\mathbf{v}_3 \leftarrow \alpha\mathbf{v}_2 - K_2 : -\alpha^2 K_0 - \alpha K_1 - K_2$
업데이트 4) $\mathbf{v}_4 \leftarrow \alpha\mathbf{v}_3 - K_3 : -\alpha^3 K_0 - \alpha^2 K_1 - \alpha K_2 - K_3$

- 1) $\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}_1$
- 2) $\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}_2$
- 3) $\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}_3$
- 4) $\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}_4$

- **Adagrad**

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

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

업데이트 1) $\frac{1}{\sqrt{K_0^2}} K_o$
업데이트 2) $\frac{1}{\sqrt{K_1^2+K_0^2}} K_1$
업데이트 3) $\frac{1}{\sqrt{K_3^2+K_1^2+K_0^2}} K_3$
업데이트 4) $\frac{1}{\sqrt{K_4^2+K_3^2+K_1^2+K_0^2}} K_4$

- **RMSprop**

$$\mathbf{h} \longleftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}}\right)$$

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

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = a^2(1-a)\mathbf{K}_1^2 + a(1-a)\mathbf{K}_2^2 + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = a^3(1-a)\mathbf{K}_1^2 + a^2(1-a)\mathbf{K}_2^2 + a(1-a)\mathbf{K}_3^2 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- **Momentum**

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

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

업데이트 1) $v_1 \leftarrow \alpha * 0 - K_0 : -K_0$
업데이트 2) $v_2 \leftarrow \alpha v_1 - K_1 : -\alpha K_0 - K_1$
업데이트 3) $v_3 \leftarrow \alpha v_2 - K_2 : -\alpha^2 K_0 - \alpha K_1 - K_2$
업데이트 4) $v_4 \leftarrow \alpha v_3 - K_3 : -\alpha^3 K_0 - \alpha^2 K_1 - \alpha K_2 - K_3$

- 1) $\mathbf{W} \leftarrow \mathbf{W} + v_1$
- 2) $\mathbf{W} \leftarrow \mathbf{W} + v_2$
- 3) $\mathbf{W} \leftarrow \mathbf{W} + v_3$
- 4) $\mathbf{W} \leftarrow \mathbf{W} + v_4$

- **Adagrad**

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

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

업데이트 1) $\frac{1}{\sqrt{\boxed{K_0^2}}} K_o$
업데이트 2) $\frac{1}{\sqrt{\boxed{K_1^2+K_0^2}}} K_1$
업데이트 3) $\frac{1}{\sqrt{\boxed{K_3^2+K_1^2+K_0^2}}} K_3$
업데이트 4) $\frac{1}{\sqrt{\boxed{K_4^2+K_3^2+K_1^2+K_0^2}}} K_4$

- **RMSprop**

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}}\right)$$

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

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = a^2(1-a)\mathbf{K}_1^2 + a(1-a)\mathbf{K}_2^2 + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = a^3(1-a)\mathbf{K}_1^2 + a^2(1-a)\mathbf{K}_2^2 + a(1-a)\mathbf{K}_3^2 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

모두의연구소

- Momentum

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

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

업데이트 1) $v_1 \leftarrow \alpha * 0 - K_0 : -K_0$
업데이트 2) $v_2 \leftarrow \alpha v_1 - K_1 : -\alpha K_0 - K_1$
업데이트 3) $v_3 \leftarrow \alpha v_2 - K_2 : -\alpha^2 K_0 - \alpha K_1 - K_2$
업데이트 4) $v_4 \leftarrow \alpha v_3 - K_3 : -\alpha^3 K_0 - \alpha^2 K_1 - \alpha K_2 - K_3$

- 1) $W \leftarrow W + v_1$
- 2) $W \leftarrow W + v_2$
- 3) $W \leftarrow W + v_3$
- 4) $W \leftarrow W + v_4$

- Adagrad

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

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

업데이트 1) $\frac{1}{\sqrt{\boxed{K_0^2}}}K_o$
업데이트 2) $\frac{1}{\sqrt{\boxed{K_1^2+K_0^2}}}K_1$
업데이트 3) $\frac{1}{\sqrt{\boxed{K_3^2+K_1^2+K_0^2}}}K_3$
업데이트 4) $\frac{1}{\sqrt{\boxed{K_4^2+K_3^2+K_1^2+K_0^2}}}K_4$

Gradient Normalization 스러운 방법

- RMSprop

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)(\frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}})$$

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

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = a^2(1-a)\mathbf{K}_1^2 + a(1-a)\mathbf{K}_2^2 + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = a^3(1-a)\mathbf{K}_1^2 + a^2(1-a)\mathbf{K}_2^2 + a(1-a)\mathbf{K}_3^2 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- Momentum

Gradient 이동누적 스러운 방법

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

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

업데이트 1) $v_1 \leftarrow \alpha * 0 - K_o : -K_o$
업데이트 2) $v_2 \leftarrow \alpha v_1 - K_1 : -\alpha K_o - K_1$
업데이트 3) $v_3 \leftarrow \alpha v_2 - K_2 : -\alpha^2 K_o - \alpha K_1 - K_2$
업데이트 4) $v_4 \leftarrow \alpha v_3 - K_3 : -\alpha^3 K_o - \alpha^2 K_1 - \alpha K_2 - K_3$

- 1) $W \leftarrow W + v_1$
- 2) $W \leftarrow W + v_2$
- 3) $W \leftarrow W + v_3$
- 4) $W \leftarrow W + v_4$

- Adagrad

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

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

업데이트 1) $\frac{1}{\sqrt{\boxed{K_0^2}}}K_o$
업데이트 2) $\frac{1}{\sqrt{\boxed{K_1^2+K_0^2}}}K_1$
업데이트 3) $\frac{1}{\sqrt{\boxed{K_3^2+K_1^2+K_0^2}}}K_3$
업데이트 4) $\frac{1}{\sqrt{\boxed{K_4^2+K_3^2+K_1^2+K_0^2}}}K_4$

Gradient Normalization 스러운 방법

- RMSprop

$$\mathbf{h} \leftarrow \alpha\mathbf{h} + (1-\alpha)(\frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}})$$

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

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = a^2(1-a)\mathbf{K}_1^2 + a(1-a)\mathbf{K}_2^2 + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = a^3(1-a)\mathbf{K}_1^2 + a^2(1-a)\mathbf{K}_2^2 + a(1-a)\mathbf{K}_3^2 + (1-a)\mathbf{K}_4^2$

매개변수 갱신

- **Momentum**

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

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

업데이트 1) $v_1 \leftarrow \alpha * 0 - K_0 : -K_0$
업데이트 2) $v_2 \leftarrow \alpha v_1 - K_1 : -\alpha K_0 - K_1$
업데이트 3) $v_3 \leftarrow \alpha v_2 - K_2 : -\alpha^2 K_0 - \alpha K_1 - K_2$
업데이트 4) $v_4 \leftarrow \alpha v_3 - K_3 : -\alpha^3 K_0 - \alpha^2 K_1 - \alpha K_2 - K_3$

- 1) $W \leftarrow W + v_1$
- 2) $W \leftarrow W + v_2$
- 3) $W \leftarrow W + v_3$
- 4) $W \leftarrow W + v_4$

- **Adagrad**

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

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

업데이트 1) $\frac{1}{\sqrt{\boxed{K_0^2}}} K_o$

업데이트 2) $\frac{1}{\sqrt{\boxed{K_1^2+K_0^2}}} K_1$

업데이트 3) $\frac{1}{\sqrt{\boxed{K_3^2+K_1^2+K_0^2}}} K_3$

업데이트 4) $\frac{1}{\sqrt{\boxed{K_4^2+K_3^2+K_1^2+K_0^2}}} K_4$

두 방법의 같이쓰자
Adam

- **RMSprop**

$$\mathbf{h} \leftarrow \alpha \mathbf{h} + (1-\alpha)\left(\frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}}\right)$$

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

업데이트 1) $\mathbf{h}_1 = (1-a)\mathbf{K}_1^2$
업데이트 2) $\mathbf{h}_2 = a(1-a)\mathbf{K}_1^2 + (1-a)\mathbf{K}_2^2$
업데이트 3) $\mathbf{h}_3 = a^2(1-a)\mathbf{K}_1^2 + a(1-a)\mathbf{K}_2^2 + (1-a)\mathbf{K}_3^2$
업데이트 4) $\mathbf{h}_4 = a^3(1-a)\mathbf{K}_1^2 + a^2(1-a)\mathbf{K}_2^2 + a(1-a)\mathbf{K}_3^2 + (1-a)\mathbf{K}_4^2$

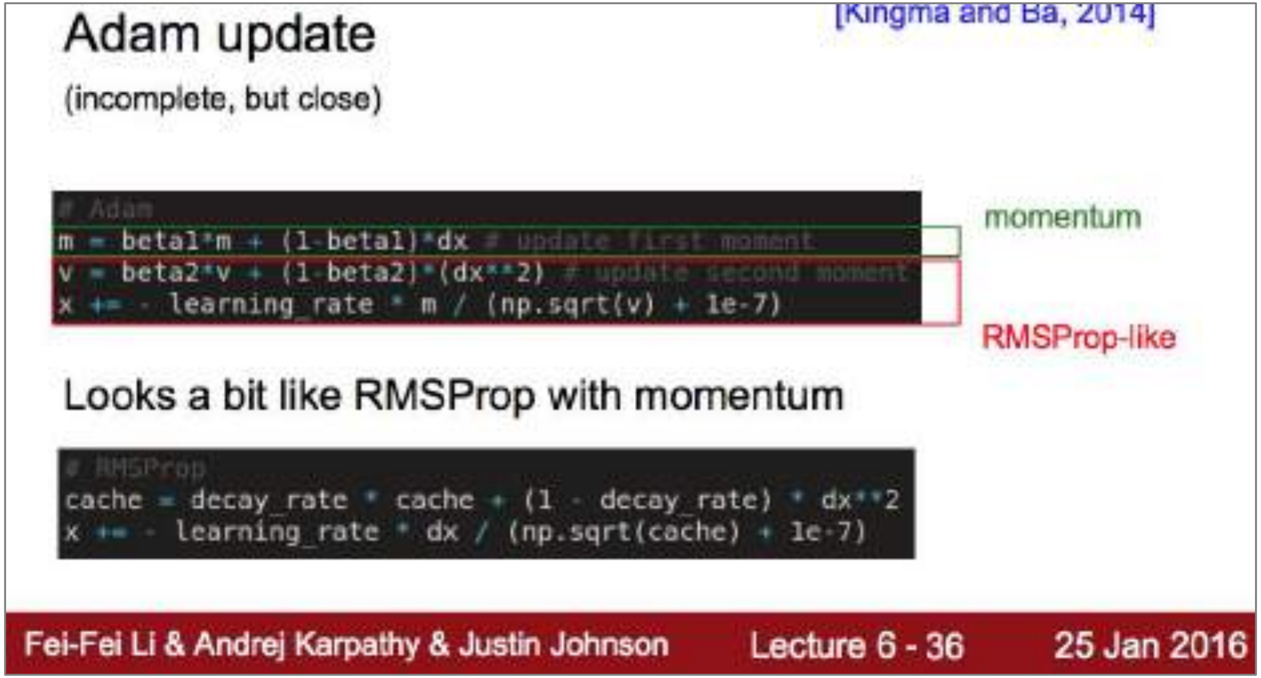
매개변수 갱신

- Adam
  - RMSProp + 모멘텀

```python
class Adam:

    """Adam (http://arxiv.org/abs/1412.6980v8)"""

    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t  = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.iter)

        for key in params.keys():
            #self.m[key] = self.beta1*self.m[key] + (1-self.beta1)*grads[key]
            #self.v[key] = self.beta2*self.v[key] + (1-self.beta2)*(grads[key]**2)
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])
```

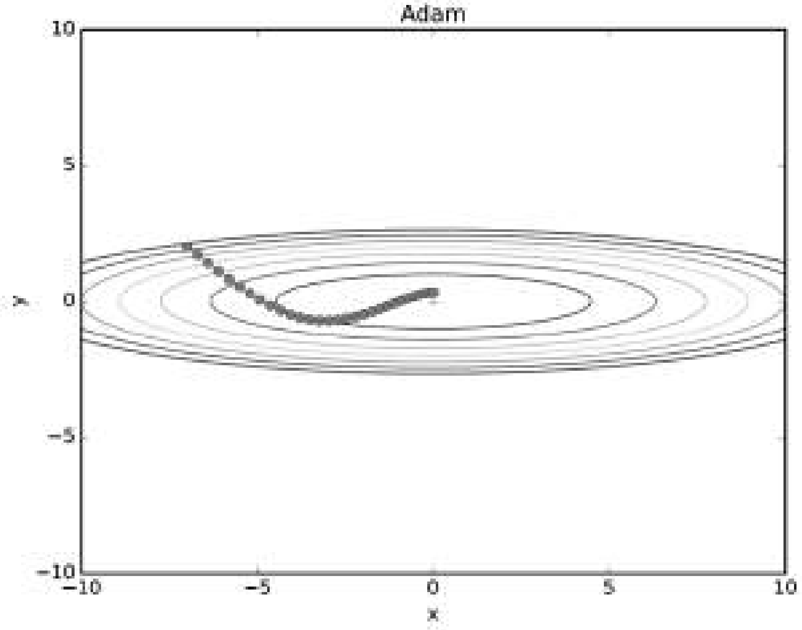
매개변수 갱신

- Adam
  - RMSProp + 모멘텀



매개변수 갱신

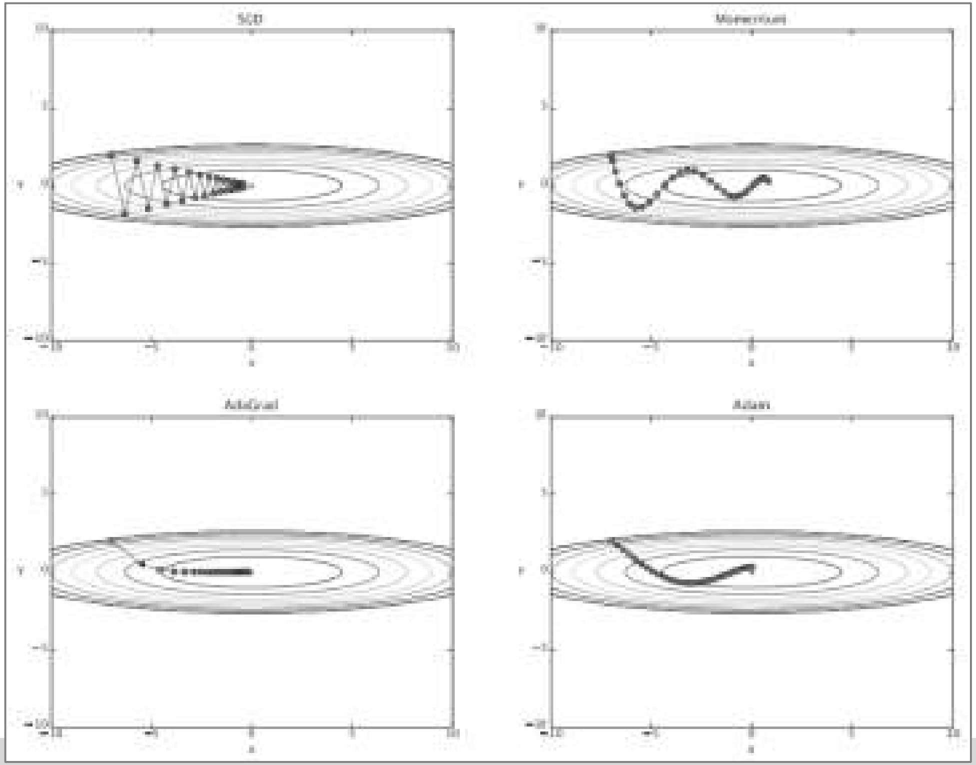
- Adam
  - RMSProp + 모멘텀



- 모멘텀처럼 그릇 바닥을 구르듯 움직임
- 모멘텀 보다 좌우 흔들림이 적음

매개변수 갱신

- **최적화 기법 비교**

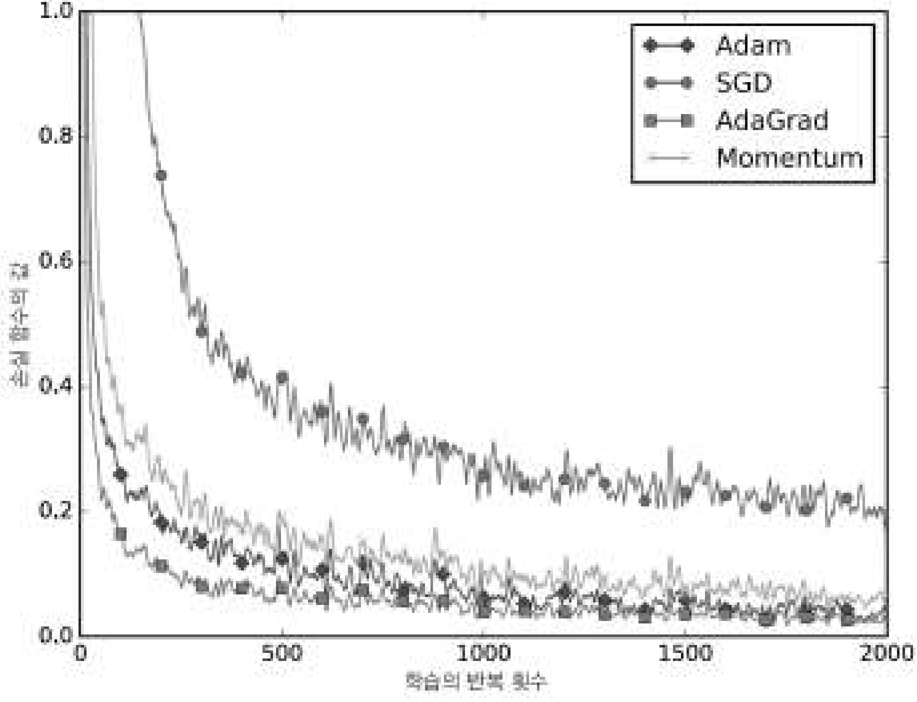


모든 문제에서 항상 뛰어난 기법은 아직 없습니다. 각자의 장단이 있습니다.

요즘 Adam을 많이 사용하는 편입니다.

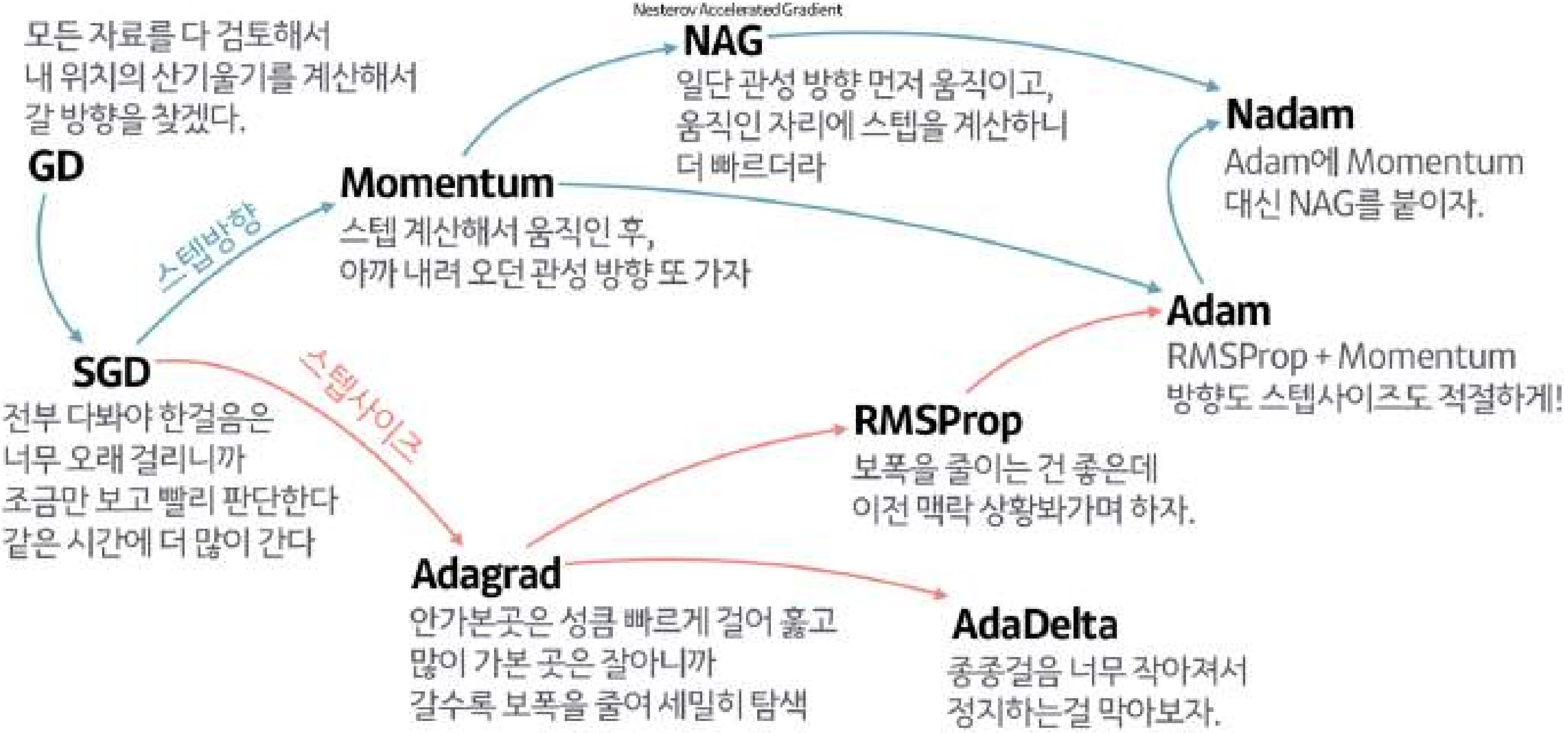
매개변수 갱신

- 최적화 기법 비교 (mnist에서 비교)



- 2_day/optimizer_compare_mnist.py

산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



참고 : "자습해도 모르겠던 딥러닝, 머리속에 인스톨 시켜드립니다", 하용호

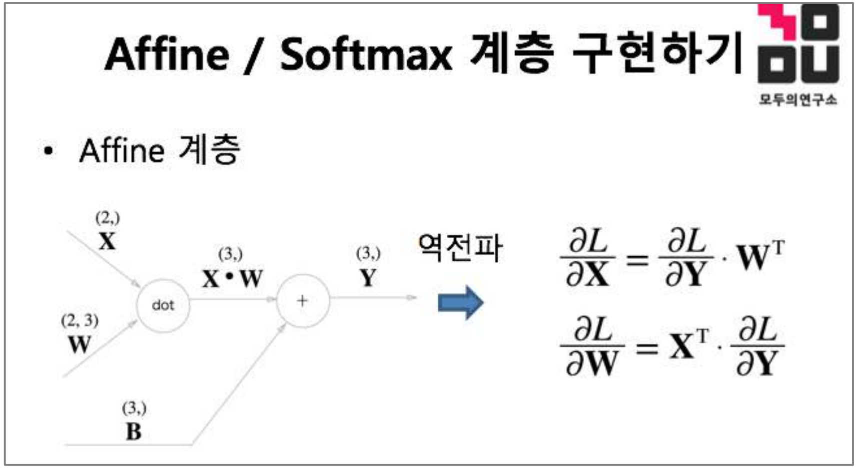
가중치의 초깃값

- 가중치 값을 작게하면 - 오버피팅이 덜 발생한다
- 가중치 값을 0으로 하면 ?

가중치의 초깃값

- 가중치 값을 작게하면 – 오버피팅이 덜 발생한다

- 가중치 값을 0으로 하면 ?
  - 학습이 발생하지 않습니다
  - 초깃값을 무작위로 설정해야 합니다



가중치의 초깃값

모두의연구소

- 은닉층의 활성화 값 분포
  - 초깃값의 변화에 따라 은닉층의 활성화 값 분포가 어떻게 변하는지 확인해 보고자 합니다
  - 실험내용
    - 5개의 층, 각 층의 뉴런은 100개
    - 입력 데이터로 1000개의 데이터를 무작위로 생성
    - 활성화 함수로 시그모이드
    - 각 층의 활성화 함수 값을 activations변수에 저장

가중치의 초깃값

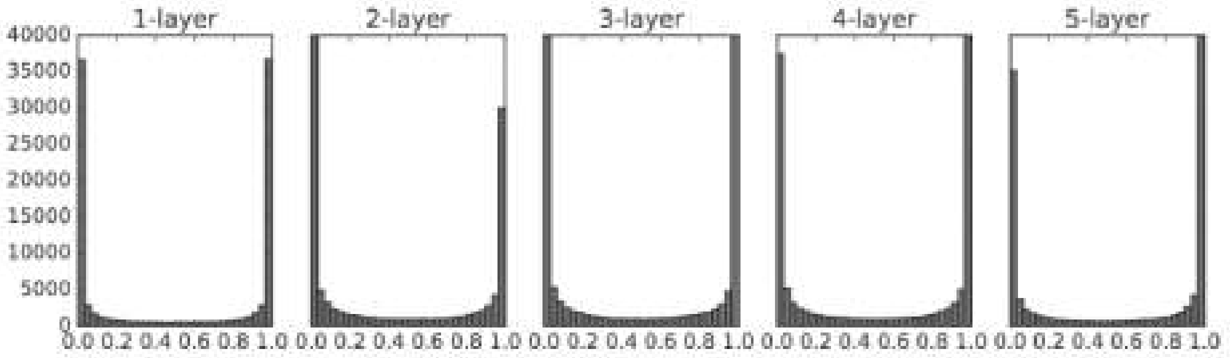
- 은닉층의 활성화 값 분포
  - 2_day/weight_init_activation_hitogram.py
  - 표준편차를 다르게 하여 실험

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

    # 초깃값을 다양하게 바꿔가며 실험해보자!
    w = np.random.randn(node_num, node_num) * 1
    # w = np.random.randn(node_num, node_num) * 0.01
    # w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    # w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)


    a = np.dot(x, w)


    # 활성화 함수도 바꿔가며 실험해보자!
    z = sigmoid(a)
    # z = ReLU(a)
    # z = tanh(a)

    activations[i] = z

# 히스토그램 그리기
for i, a in activations.items():
    plt.subplot(1, len(activations), i+1)
    plt.title(str(i+1) + "-layer")
    if i != 0: plt.yticks([], [])
    # plt.xlim(0.1, 1)
    # plt.ylim(0, 7000)
    plt.hist(a.flatten(), 30, range=(0,1))
plt.show()
```

normal gausean * 1

가중치의 초깃값

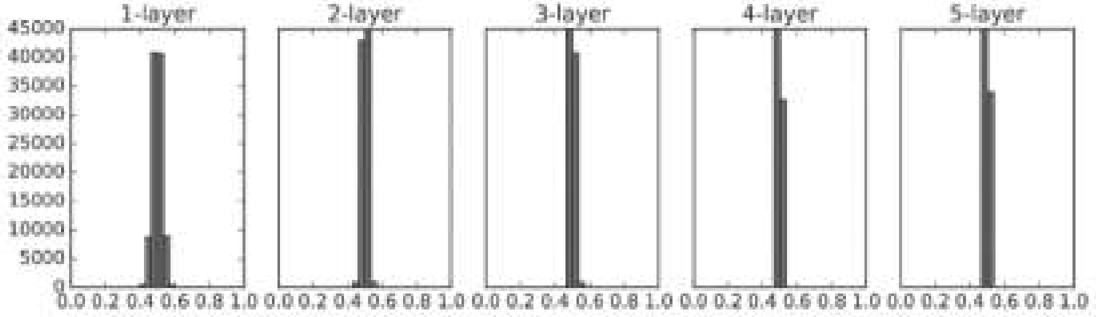
- 은닉층의 활성화 값 분포
  - 가중치를 표준편차가 1인 정규분포로 초기화 할때의 각 층의 활성화값 분포



- 값이 0과 1에 치우져 분포되어 있습니다
- 이 경우 시그모이드의 미분은 0에 가까워집니다
- 역전파 시 점점 그 값이 사라 집니다 (gradient vanishing)
- 층을 깊게 하면 더 심각해 질 것임

가중치의 초깃값

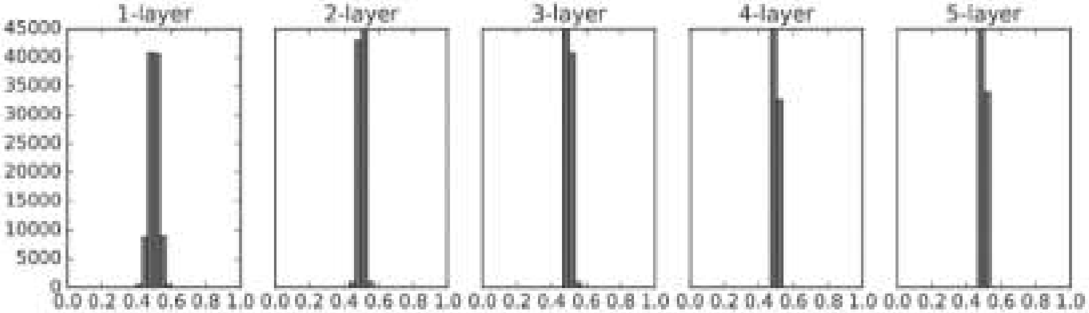
- 은닉층의 활성화 값 분포
  - 가중치를 표준편차가 0.01인 정규분포로 초기화 할때의 각 층의 활성화값 분포



  - 0.5 부근에 집중 됨. 기울기 소실이 발생하지 않음
  - 활성화 값이 치우쳐 있다는 것은 표현력 관점에서 문제가 있는 것
    - 다수의 뉴런이 거의 같은 값을 출력하니 뉴런을 여러 개 둔 의미가 없어짐. 100개가 거의 같은 값을 출력하니 1개짜리와 별반 다를바 없음
    - **활성화 값이 치우치면 표현력이 제한되어 있는 것임**

가중치의 초깃값

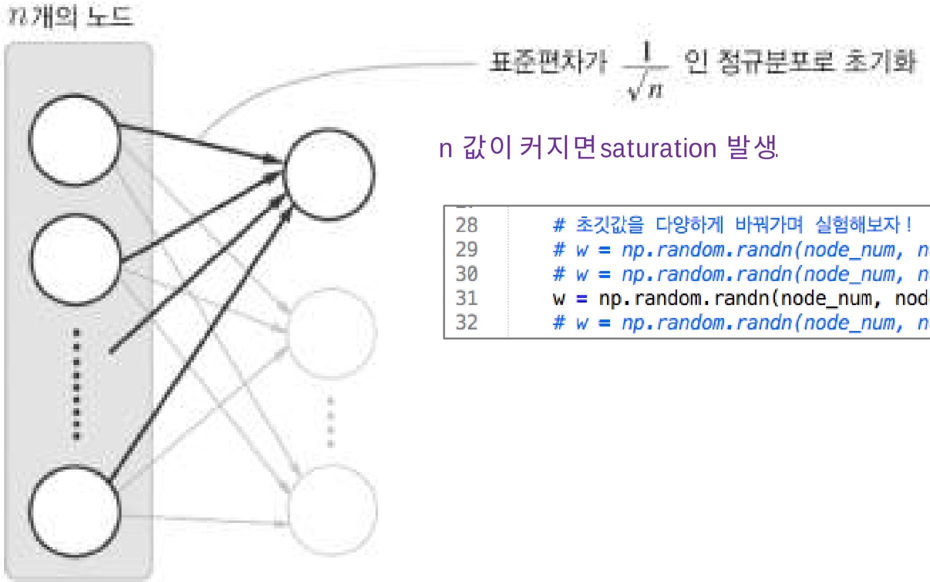
- 은닉층의 활성화 값 분포
  - 가중치를 표준편차가 0.01인 정규분포로 초기화 할때의 각 층의 활성화값 분포



- **WARNING.** 각 층의 활성화 값은 적당히 고루 분포되어야 합니다. 층과 층 사이에 적당하게 다양한 데이터가 흐르게 해야 신경망 학습이 효율적으로 이뤄지기 때문입니다. 반대로 치우친 데이터가 흐르면 기울기 소실이나 표현력 제한 문제에 빠져 학습이 잘 이뤄지지 않는 경우가 생깁니다.

가중치의 초깃값

- 은닉층의 활성화 값 분포
  - Xavier 초깃값
    - 초깃값의 표준편차가 $\frac{1}{\sqrt{n}}$인 분포를 사용

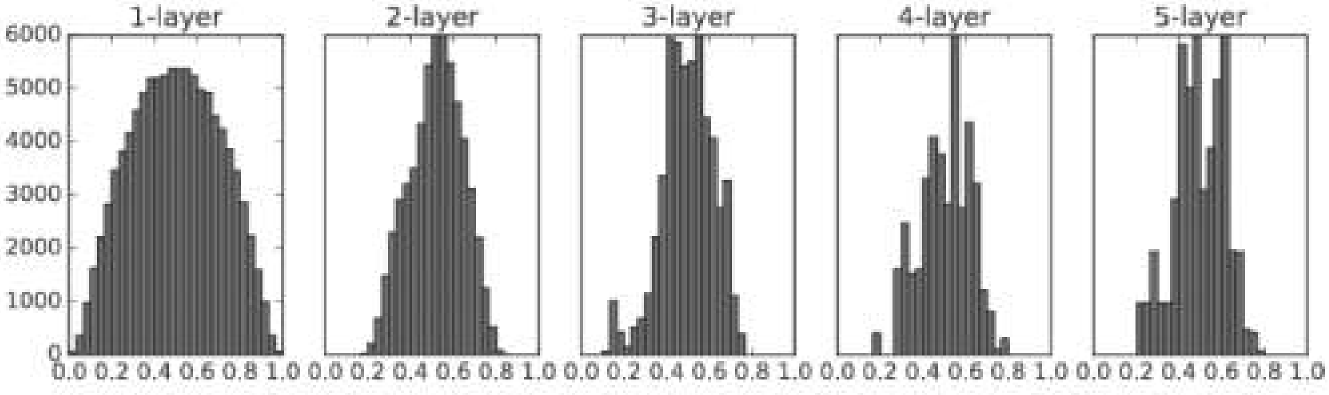


n 값이 커지면saturation 발생

```python
    # 초깃값을 다양하게 바꿔가며 실험해보자！
    # w = np.random.randn(node_num, node_num) * 1
    # w = np.random.randn(node_num, node_num) * 0.01
    w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    # w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)
```

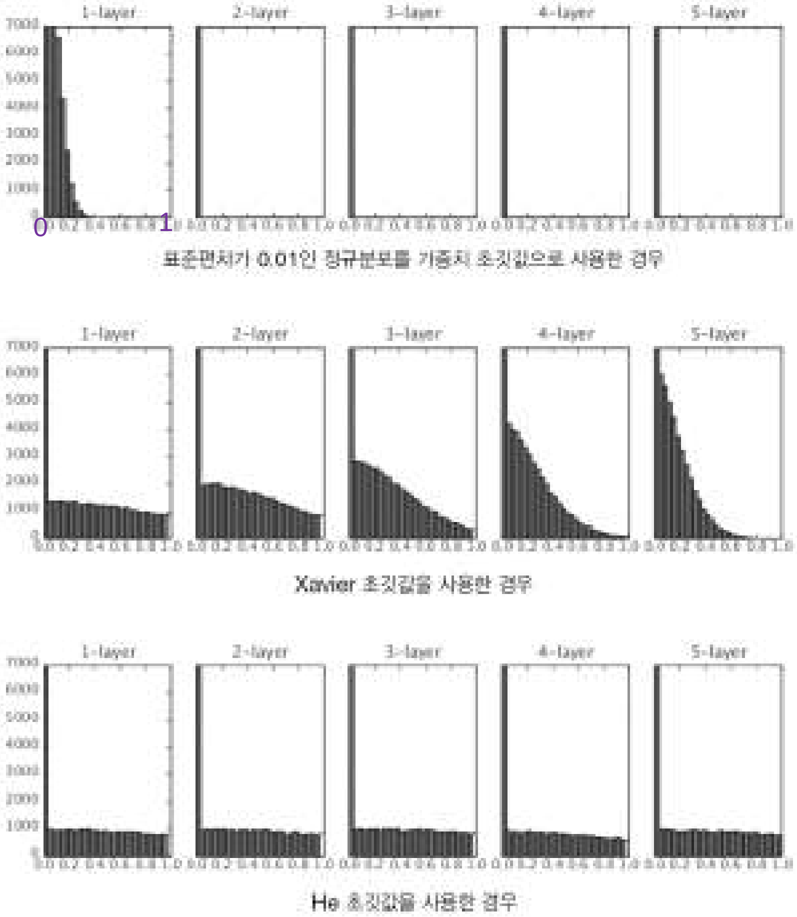
가중치의 초깃값

- 은닉층의 활성화 값 분포
  - Xavier 초깃값
    - 초깃값의 표준편차가 $\frac{1}{\sqrt{n}}$인 분포를 사용



가중치의 초깃값

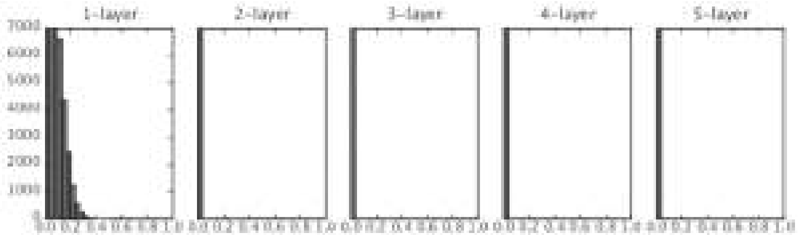
- **시그모이드 대신 ReLU를 사용한다면**
  - **He 초깃값 (Kaming He) : 표준편차가 $\frac{2}{\sqrt{n}}$인 분포 사용**



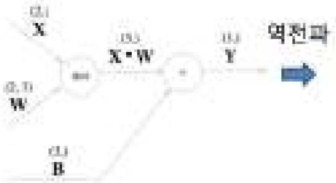
가중치의 초깃값 (ReLU)

모두의연구소

- 신경망에 작은 데이터가 흐른다는 것 -> 역전파 시 가중치의 기울기 역시 작아짐을 의미



표준편차가 0.01인 정규분포를 가중치 초깃값으로 사용한 경우

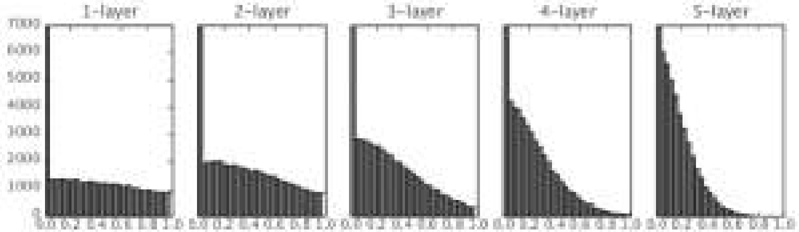


$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^{T}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{T} \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

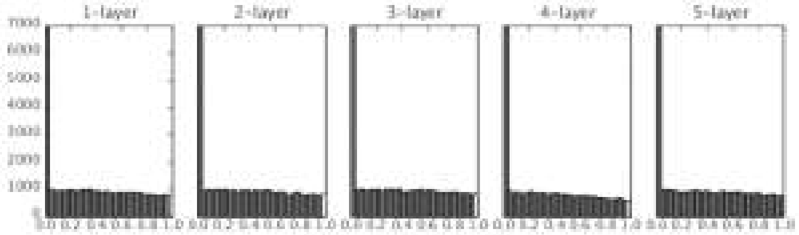
0

- 깊어질 수록 0에 쏠림 증가



Xavier 초깃값을 사용한 경우

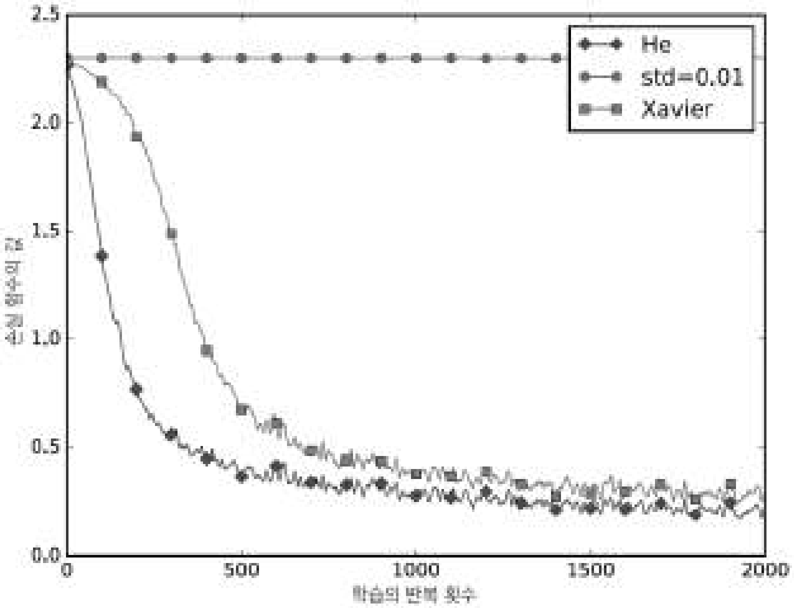
- 쓸만함



He 초깃값을 사용한 경우

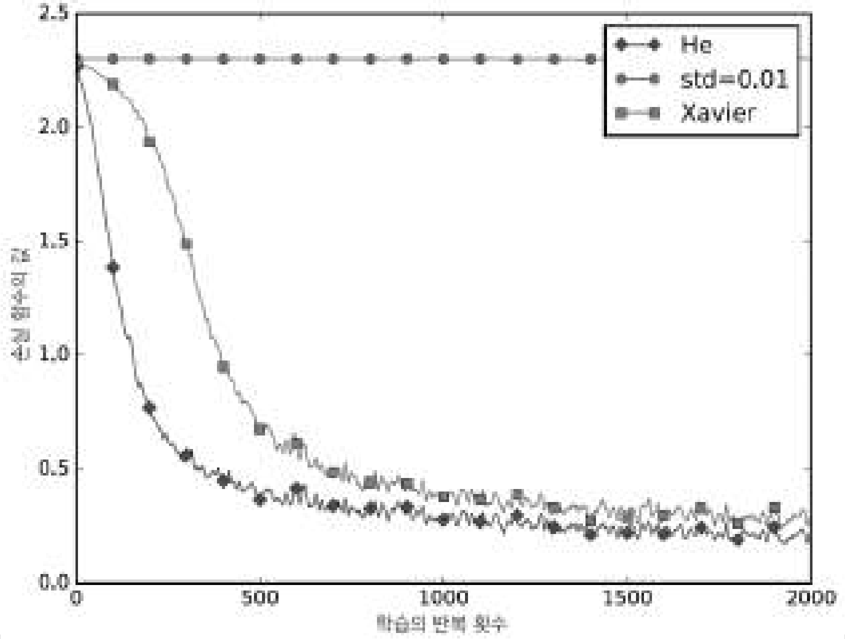
가중치의 초깃값

- MNIST 데이터셋으로 본 가중치 초깃값 비교
  - 5개층, 각 뉴런 100, 활성화 함수 ReLU
- 2_day/weight_init_compare.py



가중치의 초깃값

- MNIST 데이터셋으로 본 가중치 초깃값 비교
    - 5개층, 각 뉴런 100, 활성화 함수 ReLU



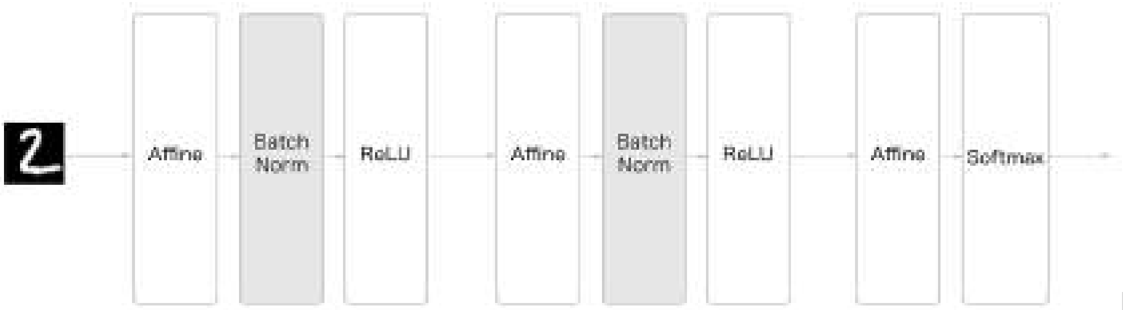
초깃값이 매우 중요하군요.
그렇지만 불편하네요~

배치 정규화

배치 정규화 (Batch Normalization)

- 2015년 등장
- 주목 받는 이유
  - 학습을 빨리 진행할 수 있다 (학습 속도 개선)
  - 초깃값에 크게 의존하지 않는다 (아픈 초깃값 선택 장애여 안녕)
  - 오버피팅을 억제한다 (드롭 아웃 등의 필요성 감소)

- 배치 정규화의 역할 : 각 층에서의 활성화값이 적당히 분포되도록 조정 (배치 정규화 계층을 삽입함)



배치 정규화 (Batch Normalization)

- 학습 시 미니배치를 단위로 데이터 분포가 평균이 0, 분산이 1이되도록 정규화 수행

1) 배치 단위 정규화

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

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

평균0, 분산1로 정규화 $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$

학습 시 미니배치의 평균과 분산은 계속 저장되어 전체 데이터에 대한 평균과 분산으로 수렴하고 이는 테스트 시 사용됩니다

배치 정규화 (Batch Normalization)

- 학습 시 미니배치를 단위로 데이터 분포가 평균이 0, 분산이 1이되도록 정규화 수행

2) 고유한 확대(scale) 이동(shift) 변환

$$y_i \leftarrow \underset{\text{확대}}{\gamma}\hat{x}_i + \underset{\text{이동}}{\beta}$$

- 초기 값 $\gamma = 1,\ \beta = 0$
- 초기 값 $\gamma,\ \beta$는 학습되는 파라미터 입니다

배치 정규화 (Batch Normalization)

모두의연구소

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want unit gaussian activations? just make them so."

consider a batch of activations at some layer.
To make each dimension unit gaussian, apply:

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

this is a vanilla
differentiable function...

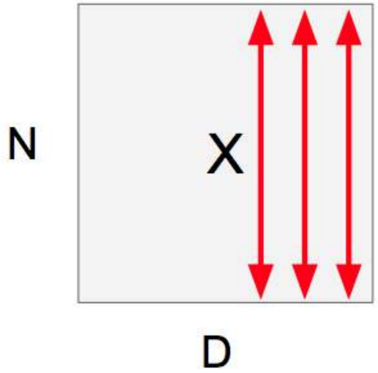
Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 54 April 20, 2017

배치 정규화 (Batch Normalization)

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want unit gaussian activations?
just make them so."



1. compute the empirical mean and variance independently for each dimension.

2. Normalize

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

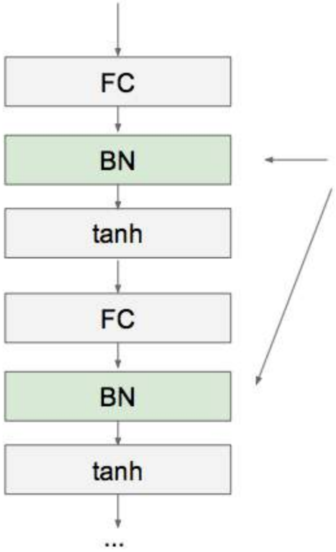
Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 55 April 20, 2017

배치 정규화 (Batch Normalization)

모두의연구소

Batch Normalization

[Ioffe and Szegedy, 2015]



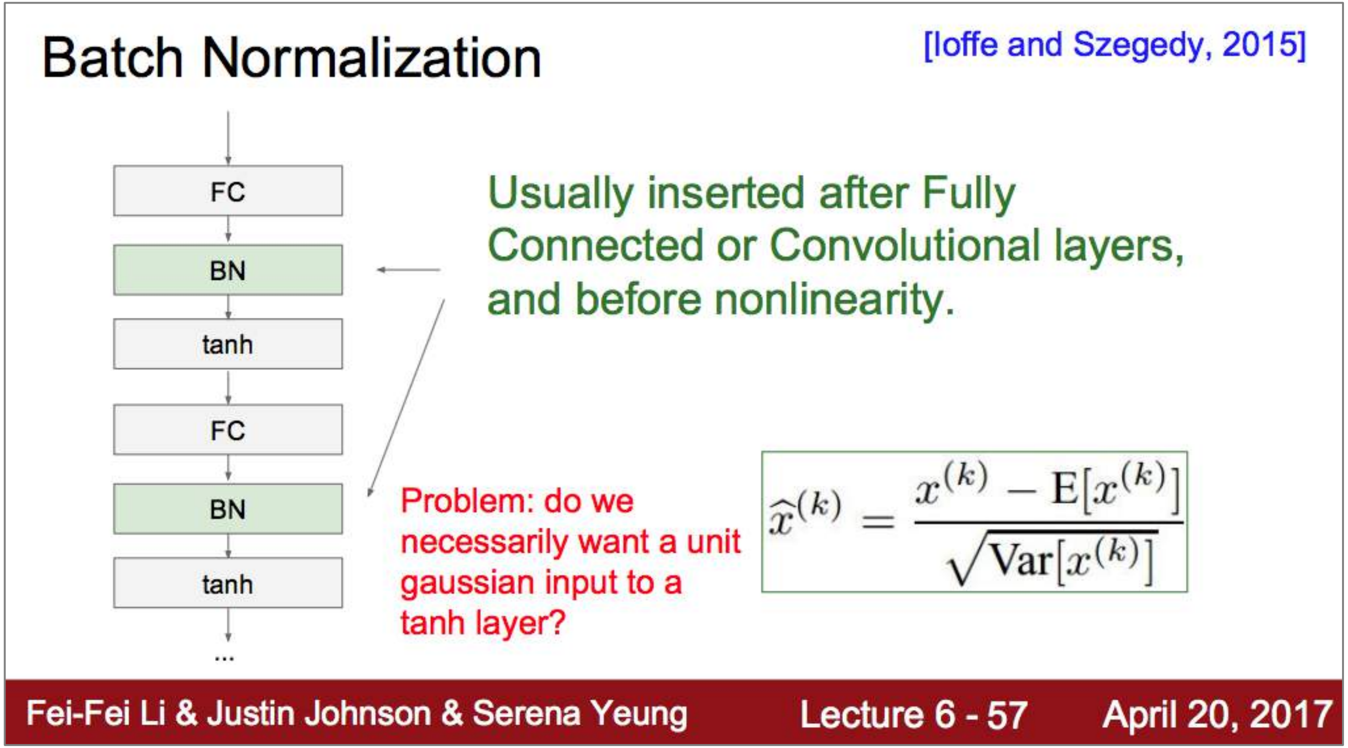
Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

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

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 56 April 20, 2017

배치 정규화 (Batch Normalization)

모두의연구소



배치 정규화 (Batch Normalization)

Batch Normalization

[Ioffe and Szegedy, 2015]

Normalize:

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

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

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

Note, the network can learn:

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

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

to recover the identity mapping.

Fei-Fei Li & Justin Johnson & Serena Yeung | Lecture 6 - 58 | April 20, 2017

배치 정규화 (Batch Normalization)

Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

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

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

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

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

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 59 April 20, 2017

배치 정규화 (Batch Normalization)

모두의연구소

Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

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

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

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

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

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

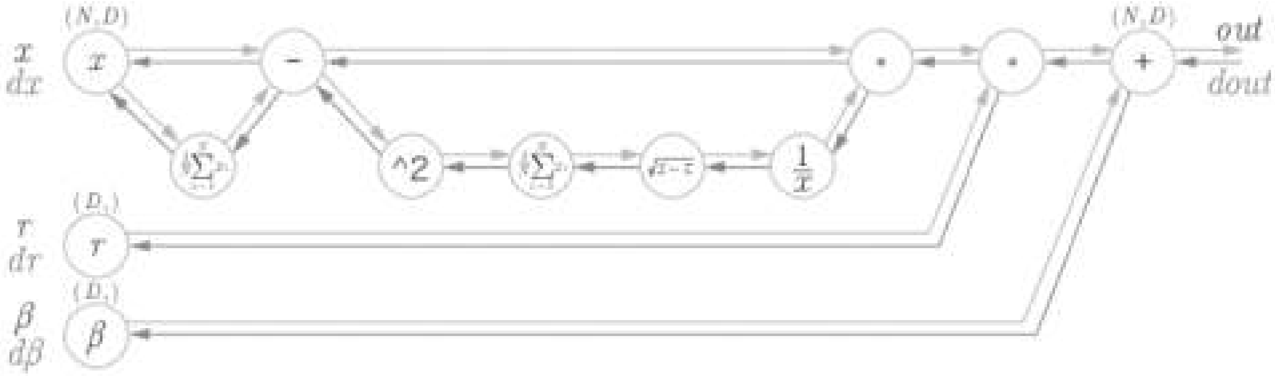
(e.g. can be estimated during training with running averages)

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 6 - 60   April 20, 2017

배치 정규화 (Batch Normalization)

- 학습 시 미니배치를 단위로 데이터 분포가 평균이 0, 분산이 1이되도록 정규화 수행

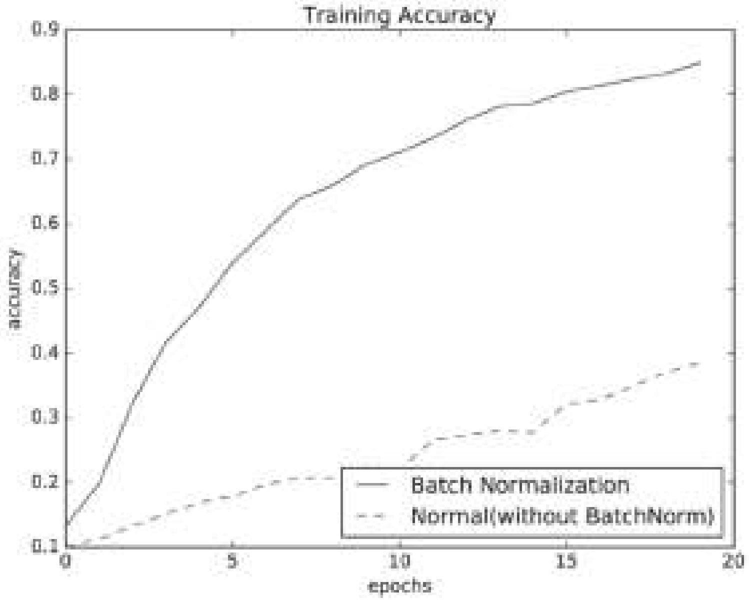
배치 정규화 계산 그래프



배치 정규화 (Batch Normalization)

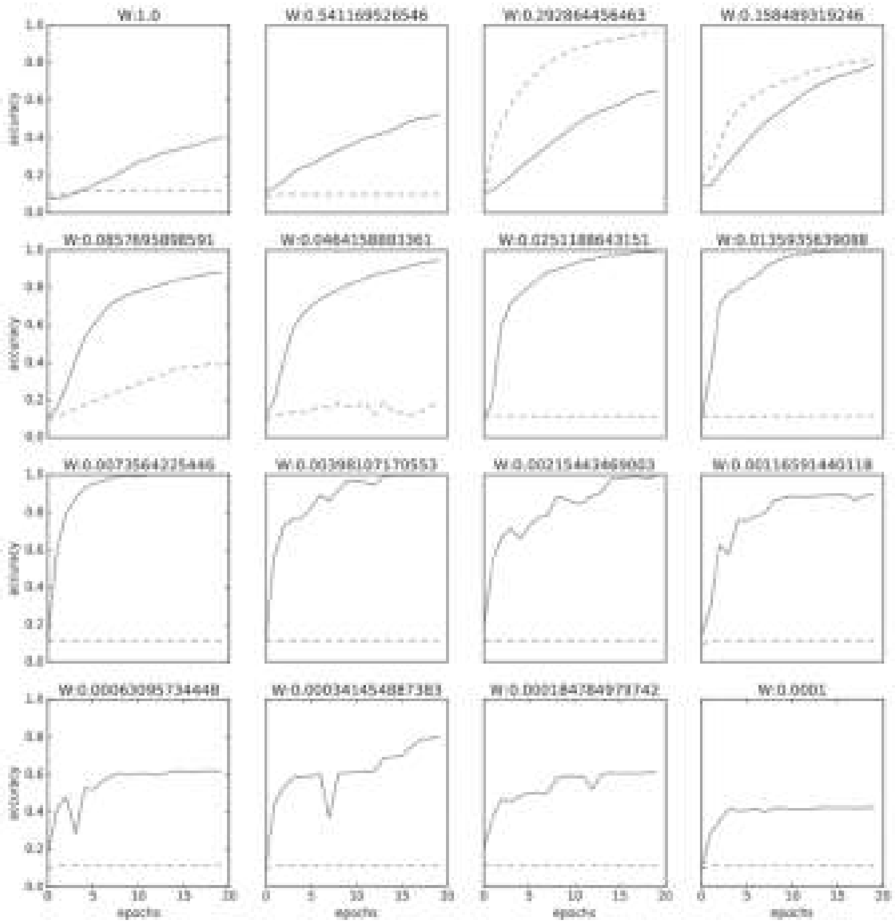
- 학습 시 미니배치를 단위로 데이터 분포가 평균이 0, 분산이 1이되도록 정규화 수행

배치 정규화 효과 : 배치 정규화가 학습 속도를 높인다



- 2_day/batch_norm_test.py

배치 정규화 (Batch Normalization)



- 실선 : Batch norm
- 점선 : 사용 안함

바른 학습을 위해

- 오버피팅 (Over fitting) : 훈련데이터에만 지나치게 적응되어 그 외의 데이터에는 제대로 대응 못함
- 주로 언제 발생하나요?

바른 학습을 위해

모두의연구소

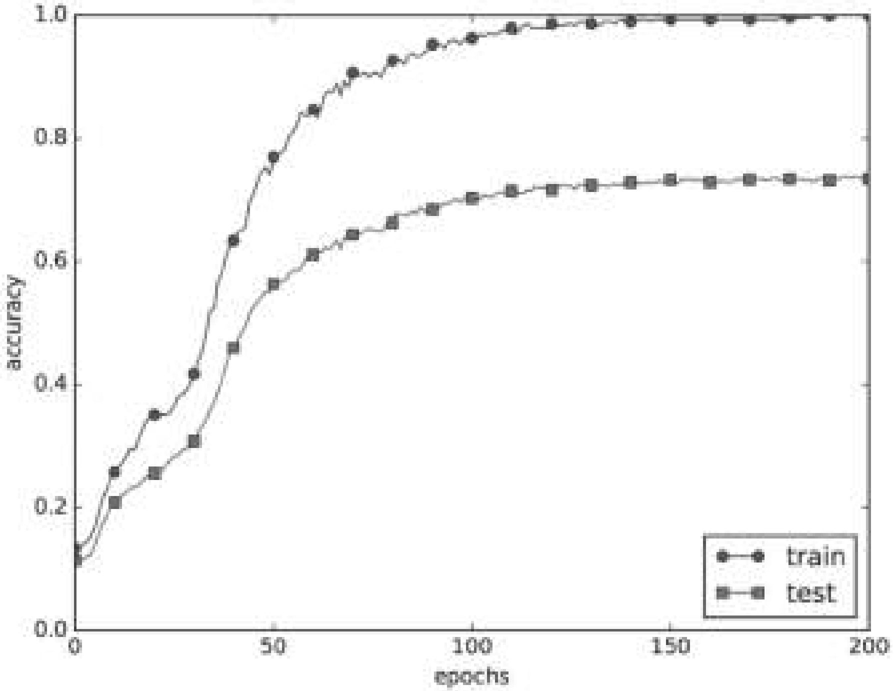
- 오버피팅 (Over fitting) : 훈련데이터에만 지나치게 적응되어 그 외의 데이터에는 제대로 대응 못함

- 주로 언제 발생하나요?
  - 매개변수가 많고 표현력이 높은 모델
  - 훈련 데이터가 적음

- 일부러 오버피팅을 발생 시키는 실험을 해봅니다
  - 데이터 : mnist데이터에서 300개만 사용
  - 각 층의 뉴런이 100개인 7층 네트워크 구성
  - ReLU적용

바른 학습을 위해

- 오버피팅 (Over fitting) : 훈련데이터에만 지나치게 적응되어 그 외의 데이터에는 제대로 대응 못함

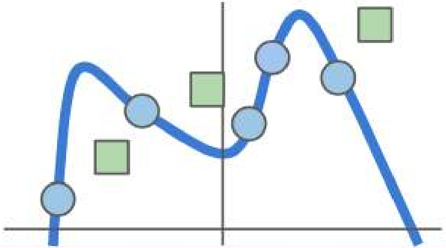


바른 학습을 위해

- 오버피팅 억제 : **가중치 감소** (weight decay)

$$L(W) = \frac{1}{N}\underbrace{\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}$$

Data loss: Model predictions
should match training data



Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 30 April 11, 2017

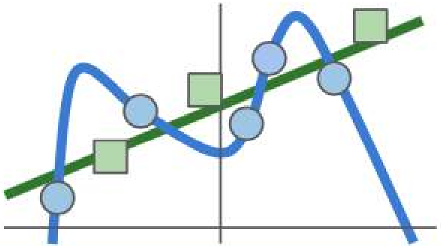
바른 학습을 위해

- 오버피팅 억제 : **가중치 감소** (weight decay)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data



Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 3 - 32   April 11, 2017

바른 학습을 위해

- 오버피팅 억제 : **가중치 감소** (weight decay)

Regularization

λ = regularization strength (hyperparameter)

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Max norm regularization (might see later)

Dropout (will see later)

Fancier: Batch normalization, stochastic depth

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 34 April 11, 2017

바른 학습을 위해

- 오버피팅 억제 : **가중치 감소** (weight decay)

L2 Regularization (Weight Decay)

$$x = [1, 1, 1, 1]$$

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$w_1 = [1, 0, 0, 0]$$
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 35 April 11, 2017

바른 학습을 위해

- 오버피팅 억제 : **가중치 감소** (weight decay)

L2 Regularization (Weight Decay)

$$x = [1, 1, 1, 1]$$

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

(If you are a Bayesian: L2 regularization also corresponds MAP inference using a Gaussian prior on W)

$$w_1^T x = w_2^T x = 1$$

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 36 April 11, 2017

바른 학습을 위해

- **오버피팅 억제 : 가중치 감소** (weight decay)
- Weight decay 구현 : common/multi_layer_net.py

```python
    def loss(self, x, t):
        """손실 함수를 구한다.

        Parameters
        ----------
        x : 입력 데이터
        t : 정답 레이블

        Returns
        -------
        손실 함수의 값
        """
        y = self.predict(x)

        weight_decay = 0
        for idx in range(1, self.hidden_layer_num + 2):
            W = self.params['W' + str(idx)]
            weight_decay += 0.5 * self.weight_decay_lambda * np.sum(W ** 2)

        return self.last_layer.forward(y, t) + weight_decay
```

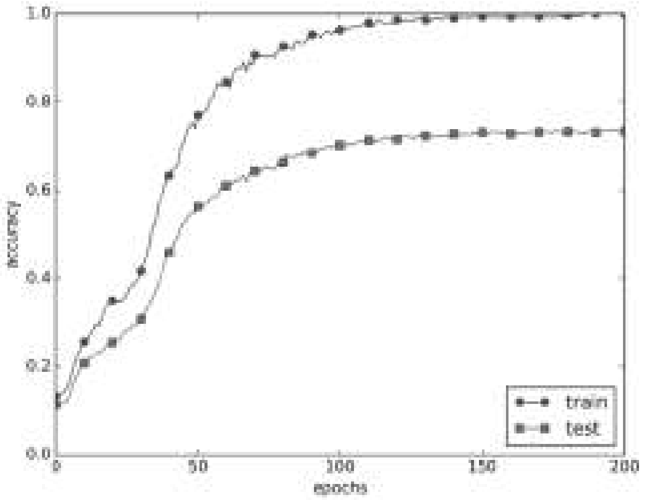
바른 학습을 위해

- Weight decay 구현 : common/multi_layer_net.py

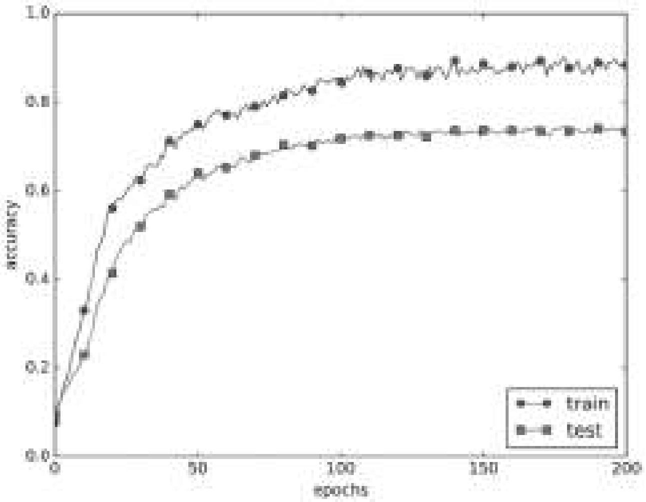
```python
    def gradient(self, x, t):
        """기울기를 구한다(오차역전파법).

        Parameters
        ----------
        x : 입력 데이터
        t : 정답 레이블

        Returns
        -------
        각 층의 기울기를 담은 딕셔너리(dictionary) 변수
            grads['W1'], grads['W2'], ... 각 층의 가중치
            grads['b1'], grads['b2'], ... 각 층의 편향
        """
        # forward
        self.loss(x, t)

        # backward
        dout = 1
        dout = self.last_layer.backward(dout)

        layers = list(self.layers.values())
        layers.reverse()
        for layer in layers:
            dout = layer.backward(dout)

        # 결과 저장
        grads = {}
        for idx in range(1, self.hidden_layer_num+2):
            grads['W' + str(idx)] = self.layers['Affine' + str(idx)].dW + \
             self.weight_decay_lambda * self.layers['Affine' + str(idx)].W
            grads['b' + str(idx)] = self.layers['Affine' + str(idx)].db

        return grads
```

바른 학습을 위해

- 오버피팅 억제 : **가중치 감소** (weight decay)



Weight decay 미 적용



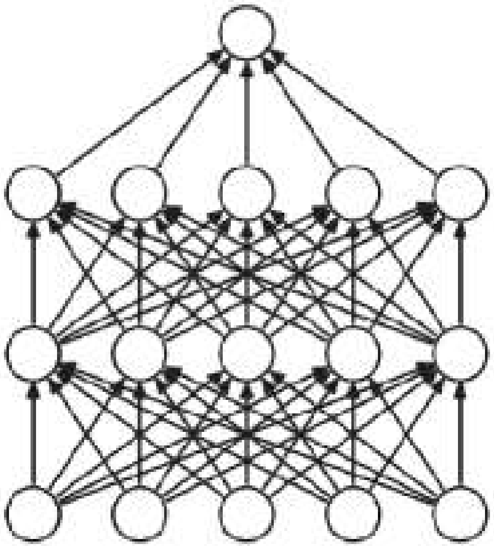
Weight decay 적용

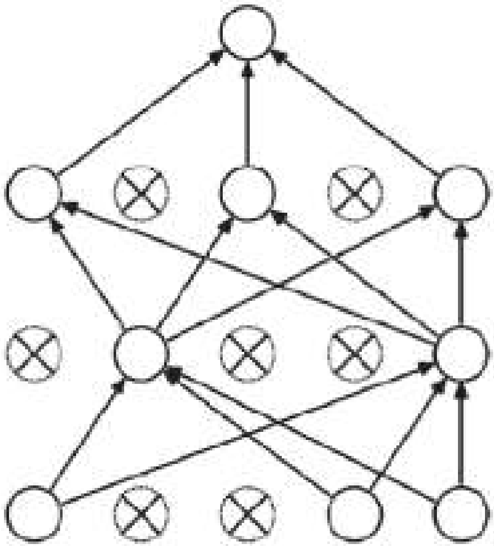
바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)
  - 훈련때 은닉층의 뉴런을 무작위로 골라 삭제하면서 학습하는 방법
  - 방법론
    - 1) 훈련때 데이터를 흘릴 때마다 삭제할 뉴런을 무작위로 선택하고
    - 2) 시험때는 모든 뉴런에 신호를 전달
      - 단, 시험 때는 각 뉴런의 출력에 훈련 때 삭제한 비율을 곱하여 출력

바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)


(a) 일반 신경망

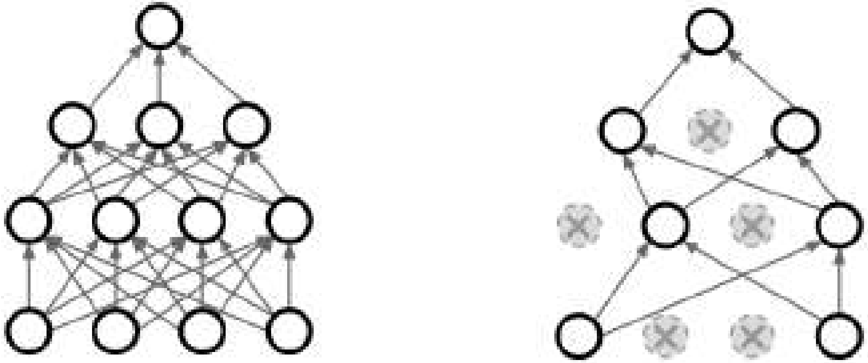

(b) 드롭아웃을 적용한 신경망

바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)

Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 60 April 25, 2017

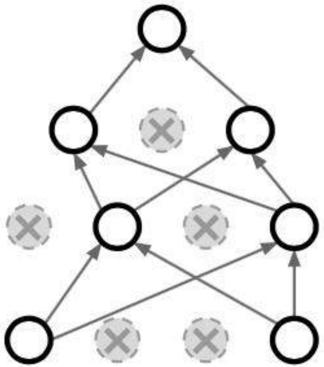
바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)

Regularization: Dropout

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

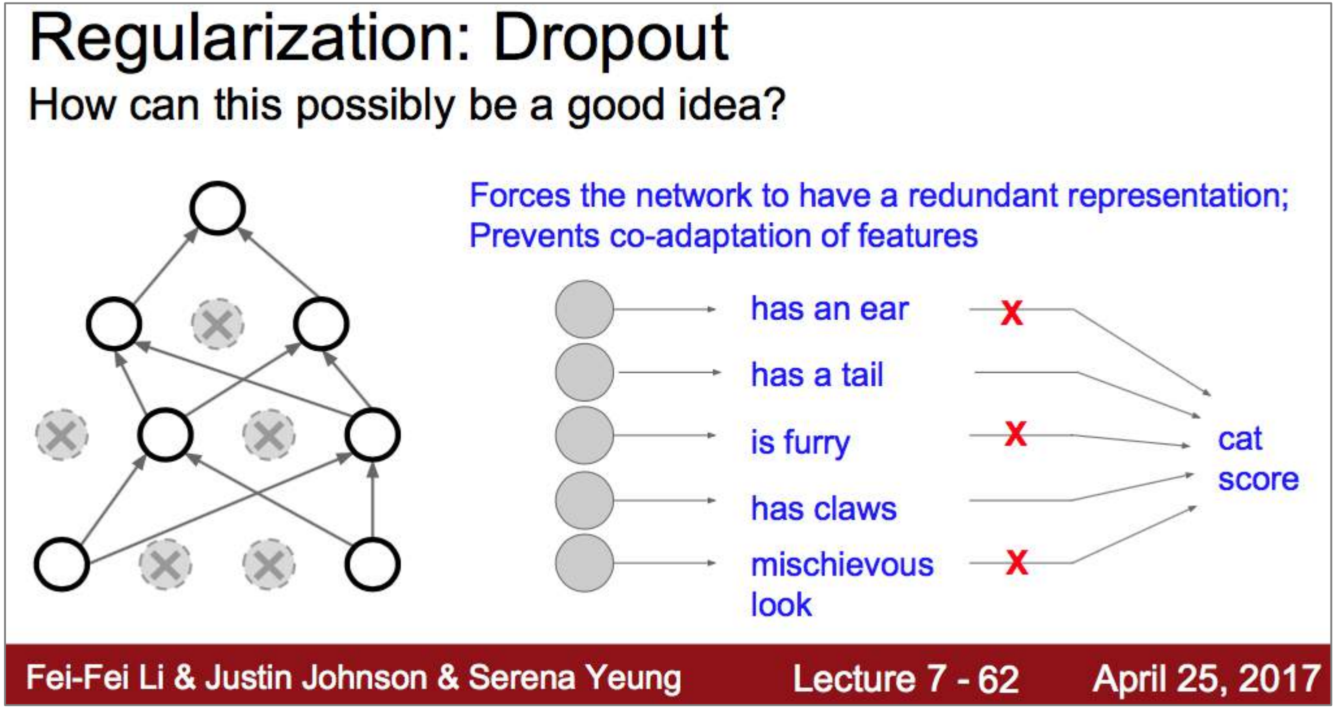
Example forward pass with a 3-layer network using dropout



Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 7 - 61   April 25, 2017

바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)

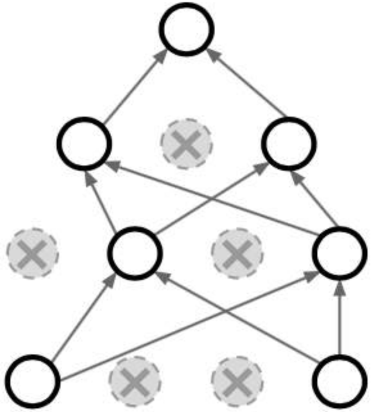


바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)

Regularization: Dropout

How can this possibly be a good idea?



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
Only ~ 10^{82} atoms in the universe...

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 7 - 63   April 25, 2017

바른 학습을 위해

모두의연구소

- **오버피팅 억제 : 드롭아웃** (Dropout)

Dropout: Test time

Dropout makes our output random!

Output (label) · Input (image) · Random mask

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

Want to "average out" the randomness at test-time

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z)f(x,z)dz$$

But this integral seems hard …

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 7 - 64   April 25, 2017

바른 학습을 위해

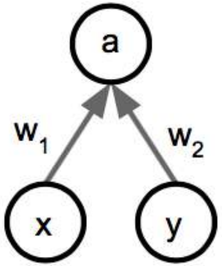
- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\left[f(x, z)\right] = \int p(z) f(x, z) dz$$

Consider a single neuron.



바른 학습을 위해

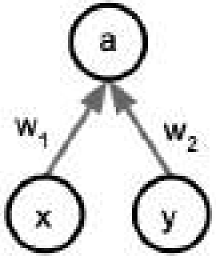
- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z[f(x, z)] = \int p(z) f(x, z) dz$$

Consider a single neuron.



At test time we have: $E[a] = w_1 x + w_2 y$

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 66 April 25, 2017

바른 학습을 위해

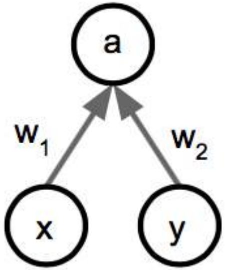
- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\left[f(x,z)\right] = \int p(z)f(x,z)dz$$

Consider a single neuron.



At test time we have: $E\left[a\right] = w_1x + w_2y$

During training we have:

$$\begin{aligned} E\left[a\right] =& \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ =& \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

Fei-Fei Li & Justin Johnson & Serena Yeung　Lecture 7 - 67　April 25, 2017

바른 학습을 위해

모두의연구소

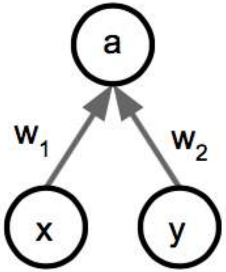
- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\left[f(x,z)\right] = \int p(z) f(x,z) dz$$

Consider a single neuron.



At test time we have: $E\left[a\right] = w_1 x + w_2 y$

During training we have:

$$\begin{aligned} E\left[a\right] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

At test time, **multiply** by dropout probability

바른 학습을 위해

모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

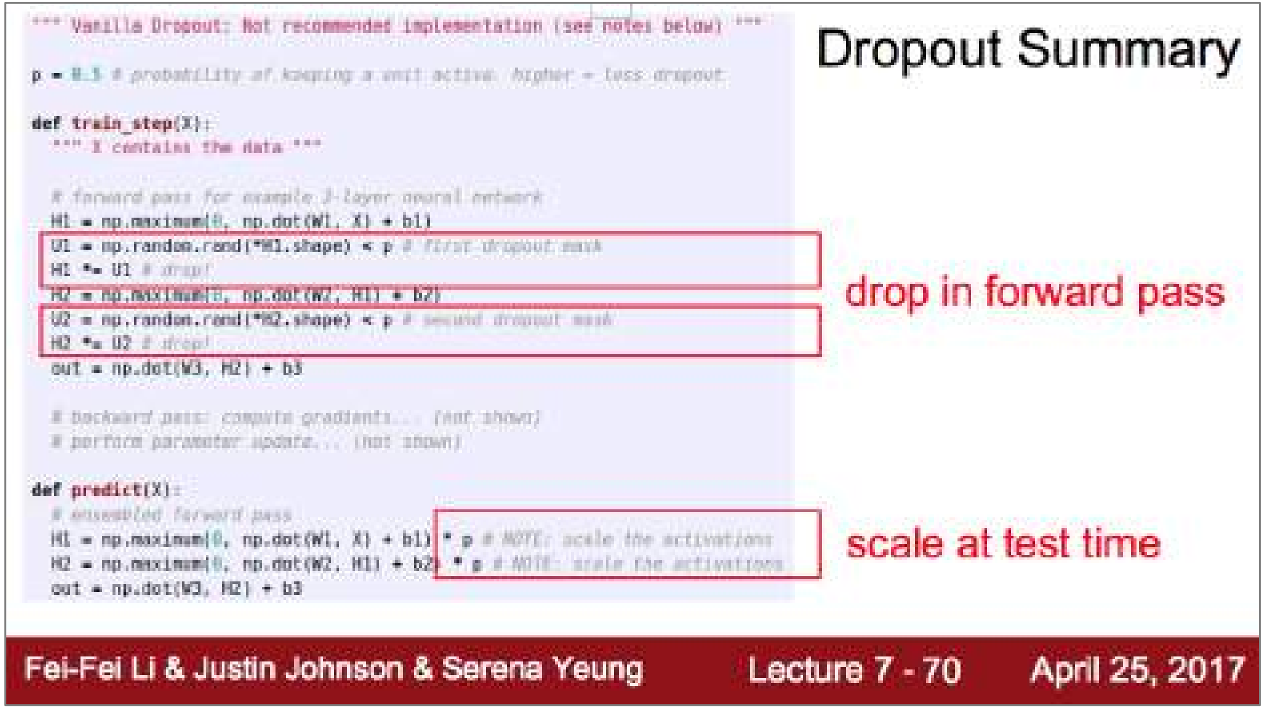
```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

At test time all neurons are active always
=> We must scale the activations so that for each neuron:
<u>output at test time</u> = <u>expected output at training time</u>

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 69 April 25, 2017

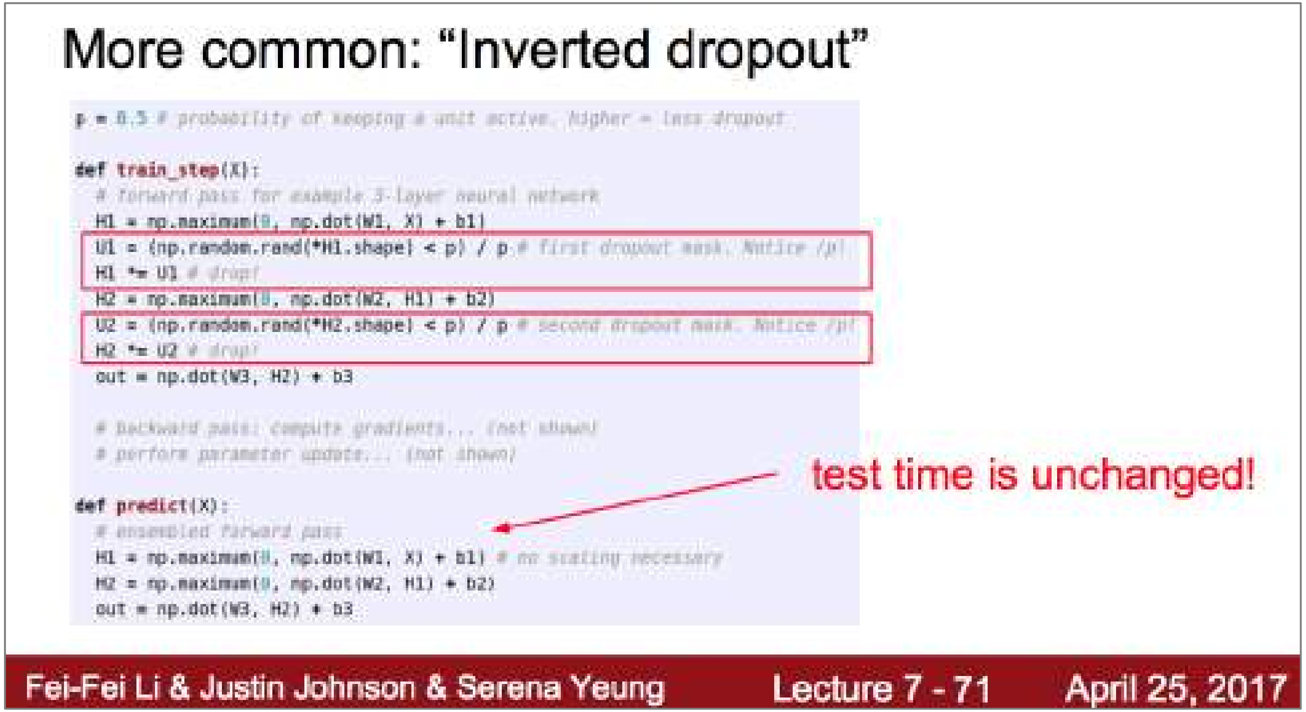
바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)



바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)



More common: "Inverted dropout"

```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

test time is unchanged!

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 71 April 25, 2017

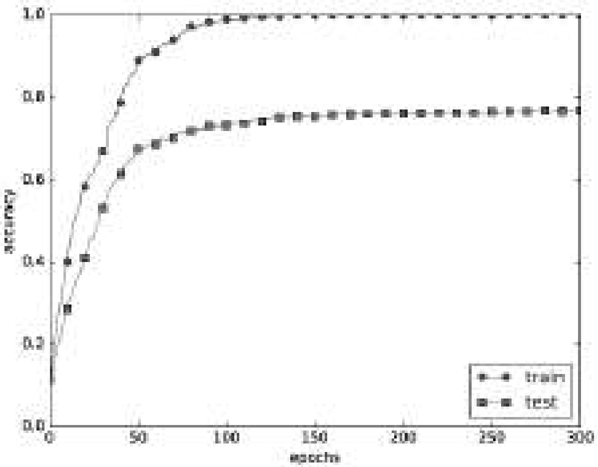
바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)
    - common/layers.py

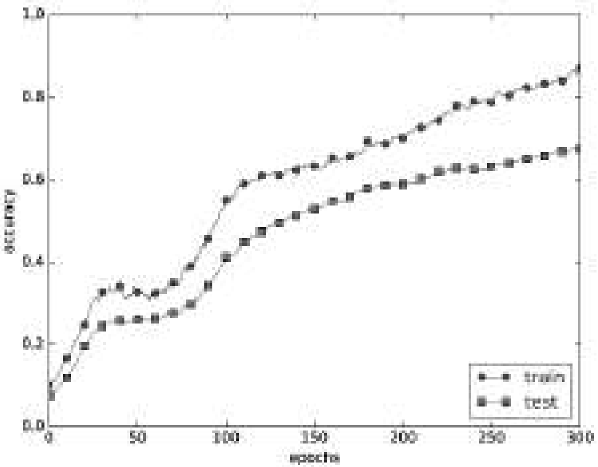
```python
class Dropout:
    """
    http://arxiv.org/abs/1207.0580
    """
    def __init__(self, dropout_ratio=0.5):
        self.dropout_ratio = dropout_ratio
        self.mask = None

    def forward(self, x, train_flg=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
```

바른 학습을 위해

- 오버피팅 억제 : **드롭아웃** (Dropout)



드랍아웃 미적용



드랍아웃 적용

적절한 하이퍼파라미터 값 찾기

- 뉴런 수는 어떻게 하지? 배치 크기는? 학습율은? 가중치 감소는 ?

하이퍼 파라미터의 값을 효율적으로 탐색하는 방법을 알아 봅시다

적절한 하이퍼파라미터 값 찾기

모두의연구소

- **검증 데이터 (validation data)**
  - 하이퍼 파라미터를 찾을 때 시험 데이터를 사용하면 안됩니다
    - 하이퍼 파라미터 값이 시험 데이터에 오버피팅 함
  - 검증 데이터 : 하이퍼파라미터 전용 확인 데이터
  - **데이터의 구분**
    - 훈련 데이터 : 매개변수 학습
    - 검증 데이터 : 하이퍼파라미터 성능 평가
    - 시험 데이터 : 신경망의 범용 성능 평가 (마지막에 이용)

적절한 하이퍼파라미터 값 찾기

- **검증 데이터 (validation data)**

  - MNIST 훈련 데이터 중 20%를 검증 데이터로 가져오기

```
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)

# 결과를 빠르게 얻기 위해 훈련 데이터를 줄임
x_train = x_train[:500]
t_train = t_train[:500]

# 20%를 검증 데이터로 분할
validation_rate = 0.20
validation_num = x_train.shape[0] * validation_rate
x_train, t_train = shuffle_dataset(x_train, t_train) # 데이터 섞기
x_val = x_train[:validation_num]
t_val = t_train[:validation_num]
x_train = x_train[validation_num:]
t_train = t_train[validation_num:]
```

적절한 하이퍼파라미터 값 찾기

- **하이퍼 파라미터 최적화**
  - 방법론
    - 하이퍼 파라미터 대략적 범위를 설정한 후 이 값을 무작위로 선택하여 정확도 평가 후 범위를 재설정하고 다시 찾기를 반복
  - 범위 설정
    - '대략적으로' 지정
    - 실제로도 0.001에서 1000사이 (10^{-3}~10^{3})과 같이 '10의 거듭제곱' 단위로 범위를 지정 : 로그 스케일(log scale)로 지정

적절한 하이퍼파라미터 값 찾기

- **하이퍼 파라미터 최적화**
  - 하이퍼 파라미터 최적화는 몇일 혹은 몇 주 이상의 오랜 시간이 걸림
    - 나쁠듯한 값은 일찍 포기하는게 좋음
  - 시간이 오래 걸리기 때문에 에폭을 작게 하여, 1회 평가에 걸리는 시간을 단축하는게 효과적임

- **하이퍼 파라미터 최적화 정리**
  - 0 단계 : 하이퍼파라미터 값의 범위를 설정
  - 1 단계 : 설정된 범위에서 하이퍼파라미터의 값을 무작위로 추출
  - 2 단계 : 1단계에서 샘플링한 하이퍼파라미터 값을 사용하여 학습하고, 검증 데이터로 정확도를 평가 (단, 에폭은 작게 설정)
  - 3단계 : 1단계와 2단계를 특정 횟수(100회 등) 반복하며, 그 정확도의 결과를 보고 하이퍼파라미터의 범위를 좁힘

적절한 하이퍼파라미터 값 찾기

모두의연구소

- **하이퍼 파라미터 최적화**

- **Note** : 여기에서 설명한 하이퍼파라미터 최적화 방법은 실용적인 방법입니다. 하지만 과학이라기 보다는 다분히 수행자의 '지혜'와 '직관'에 의존한다는 느낌이 들죠. 더 세련된 기법을 원한다면 베이즈 최적화(Bayesian optimization)를 소개할 수 있겠네요. 베이즈 최적화는 베이즈 정리(bayes's theorem)을 중심으로 한 수학 이론을 구사하여 더 엄밀하고 효율적으로 최적화를 수행합니다. 자세한 내용은 <Practical Bayesian Optimization of Machine Learning Algorithms> 논문 등을 참조하세요.

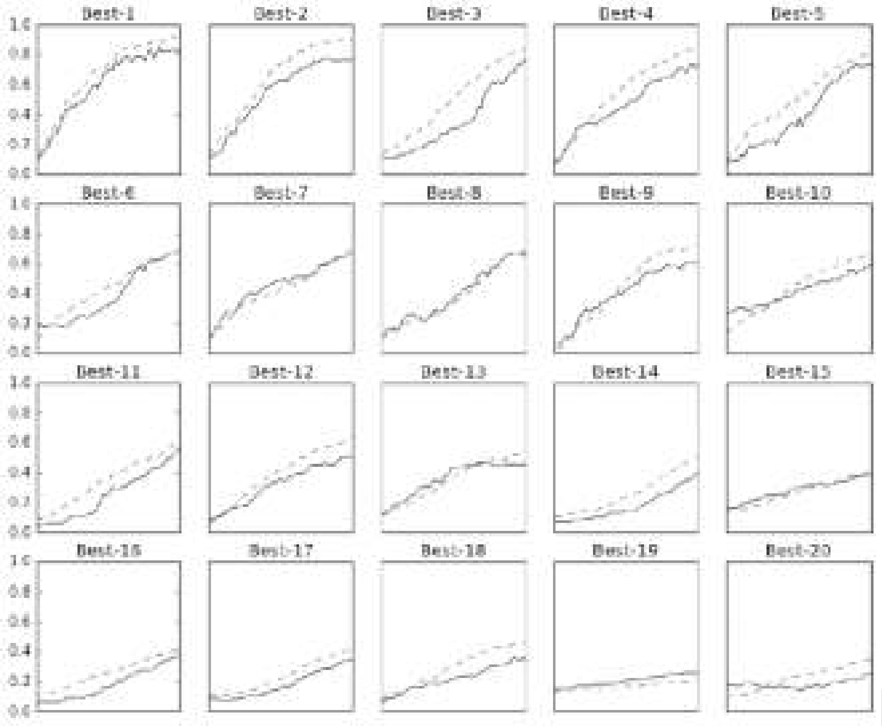
적절한 하이퍼파라미터 값 찾기

- **하이퍼 파라미터 최적화 구현**

```
# 하이퍼파라미터 무작위 탐색======================================
optimization_trial = 100
results_val = {}
results_train = {}
for _ in range(optimization_trial):
    # 탐색한 하이퍼파라미터의 범위 지정===============
    weight_decay = 10 ** np.random.uniform(-8, -4)
    lr = 10 ** np.random.uniform(-6, -2)
    # ================================================

    val_acc_list, train_acc_list = __train(lr, weight_decay)
    print("val acc:" + str(val_acc_list[-1]) + " | lr:" + str(lr) + ",
    key = "lr:" + str(lr) + ", weight decay:" + str(weight_decay)
    results_val[key] = val_acc_list
    results_train[key] = train_acc_list
```

적절한 하이퍼파라미터 값 찾기

- **하이퍼 파라미터 최적화 구현**



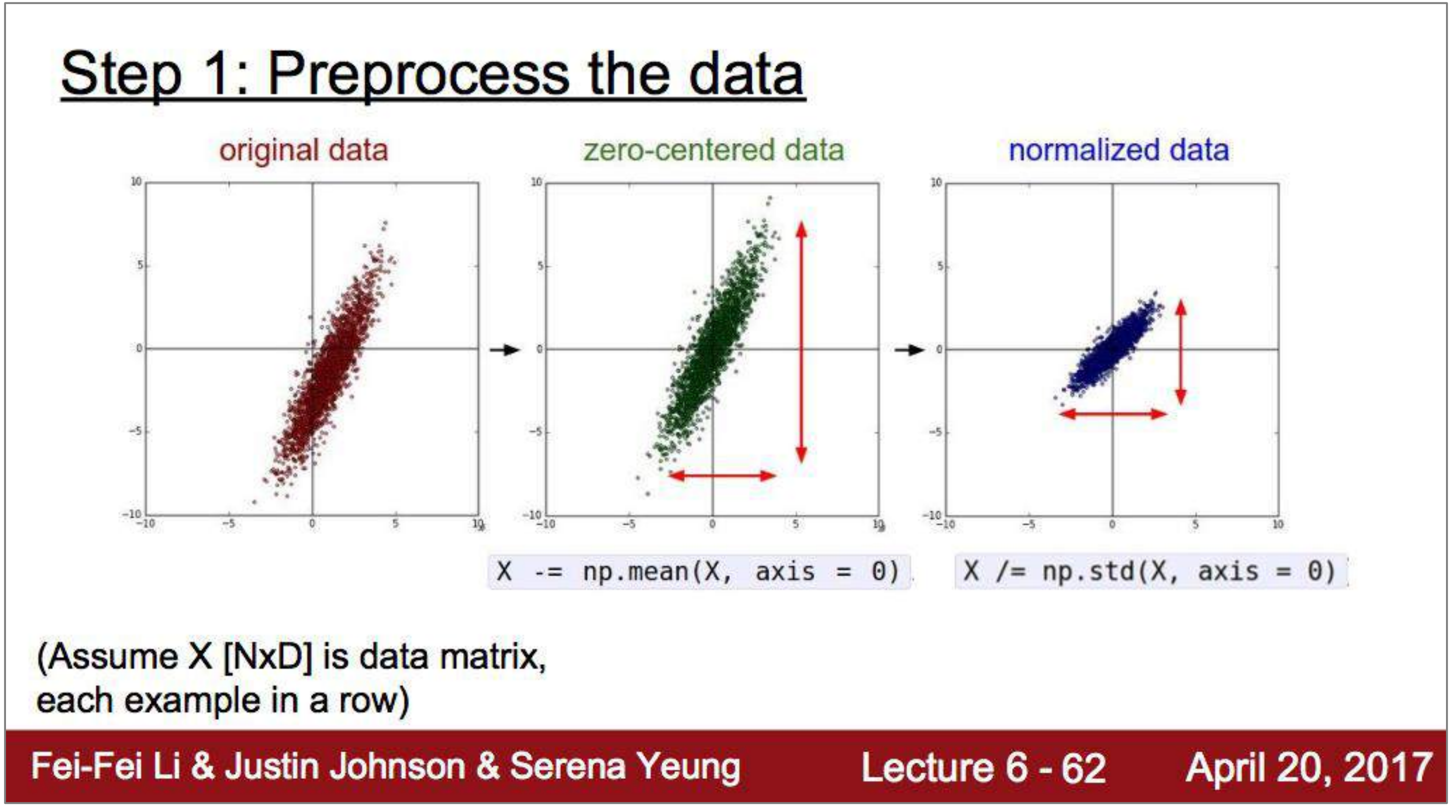
적절한 하이퍼파라미터 값 찾기

모두의연구소

Babysitting the Learning Process

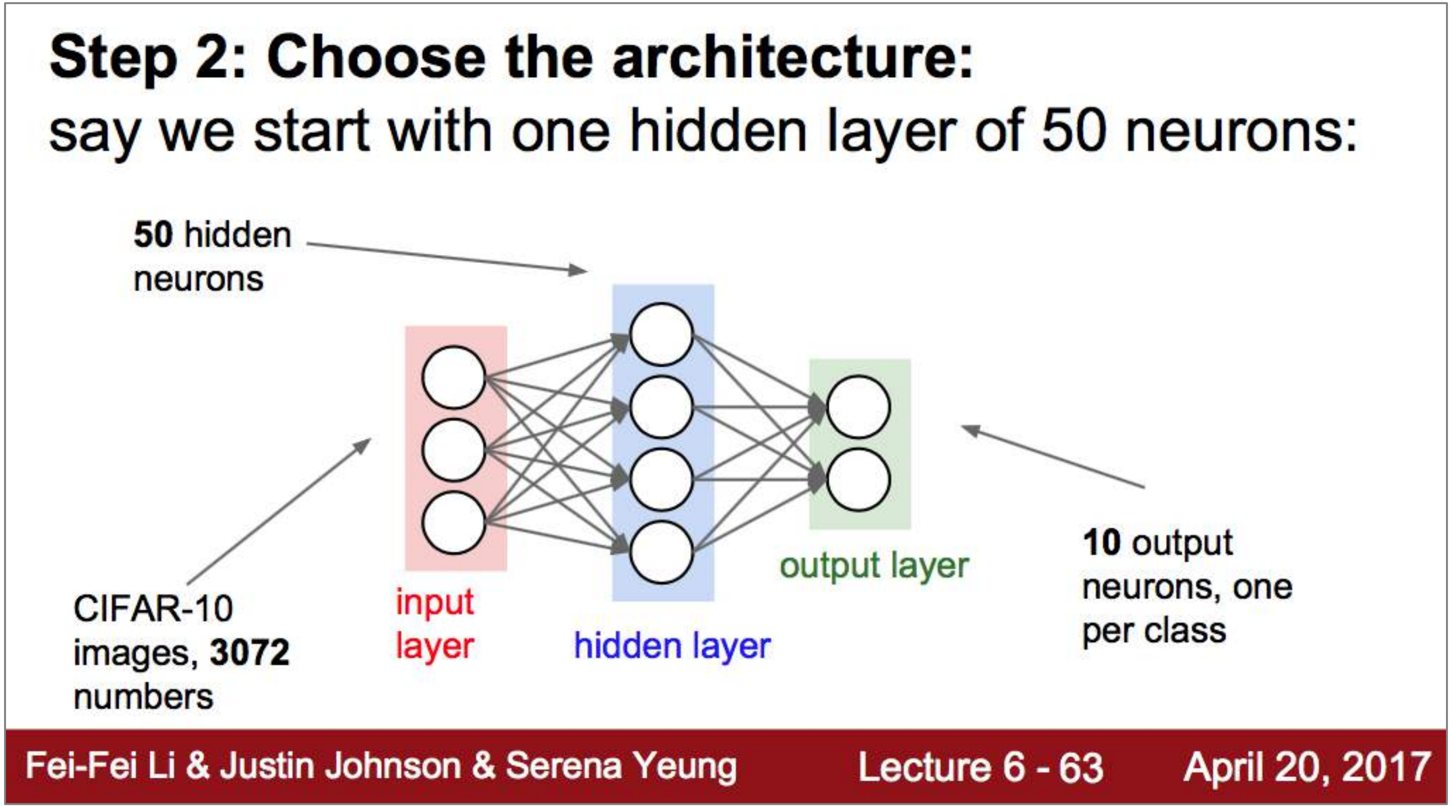
Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 61 April 20, 2017

적절한 하이퍼파라미터 값 찾기



적절한 하이퍼파라미터 값 찾기

모두의연구소



적절한 하이퍼파라미터 값 찾기

모두의연구소

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
```

```
2.30261216167
```

disable regularization

loss ~2.3.
"correct " for
10 classes

returns the loss and the
gradient for all parameters

적절한 하이퍼파라미터 값 찾기

모두의연구소

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, 1e3)
print loss
```

crank up regularization

```
3.06859716482
```

loss went up, good. (sanity check)

적절한 하이퍼파라미터 값 찾기

모두의연구소

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’

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 6 - 66   April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소

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)
```

```
Finished epoch 1 / 200: cost 2.302603, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 2 / 200: cost 2.302258, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 3 / 200: cost 2.301849, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 4 / 200: cost 2.301196, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 5 / 200: cost 2.300044, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 6 / 200: cost 2.297864, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 7 / 200: cost 2.293595, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 8 / 200: cost 2.285096, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 9 / 200: cost 2.268094, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 10 / 200: cost 2.234787, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 11 / 200: cost 2.173187, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 12 / 200: cost 2.076862, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 13 / 200: cost 1.974090, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 14 / 200: cost 1.895885, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 15 / 200: cost 1.820876, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 16 / 200: cost 1.737430, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 17 / 200: cost 1.642356, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 18 / 200: cost 1.535239, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 19 / 200: cost 1.421527, train: 0.600000, val 0.600000, lr 1.000000e-03
```

Very small loss, train accuracy 1.00, nice!

```
Finished epoch 195 / 200: cost 0.002694, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 196 / 200: cost 0.002674, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 197 / 200: cost 0.002655, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 198 / 200: cost 0.002635, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 199 / 200: cost 0.002617, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 200 / 200: cost 0.002597, train: 1.000000, val 1.000000, lr 1.000000e-03
finished optimization. best validation accuracy: 1.000000
```

적절한 하이퍼파라미터 값 찾기

Lets try to train now…

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

```
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=1e-6, verbose=True)
```

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 6 - 68   April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소

Lets try to train now…

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

```
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=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Loss barely changing

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 69 April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소

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

```
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=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Loss barely changing: Learning rate is probably too low

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 6 - 70   April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소

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

```
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=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Loss barely changing: Learning rate is probably too low

Notice train/val accuracy goes to 20% though, what's up with that? (remember this is softmax)

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 6 - 71   April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소

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

```
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,
```

Now let's try learning rate 1e6.

적절한 하이퍼파라미터 값 찾기

모두의연구소

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...

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 6 - 73   April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소

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

모두의연구소

Hyperparameter Optimization

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 75 April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소

Cross-validation strategy

coarse -> fine cross-validation in stages

First stage: only a few epochs to get rough idea of what params work
Second stage: longer running time, finer search
... (repeat as necessary)

Tip for detecting explosions in the solver:
If the cost is ever > 3 * original cost, break out early

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 76 April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소

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

적절한 하이퍼파라미터 값 찾기

모두의연구소

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.

적절한 하이퍼파라미터 값 찾기

모두의연구소

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)
```

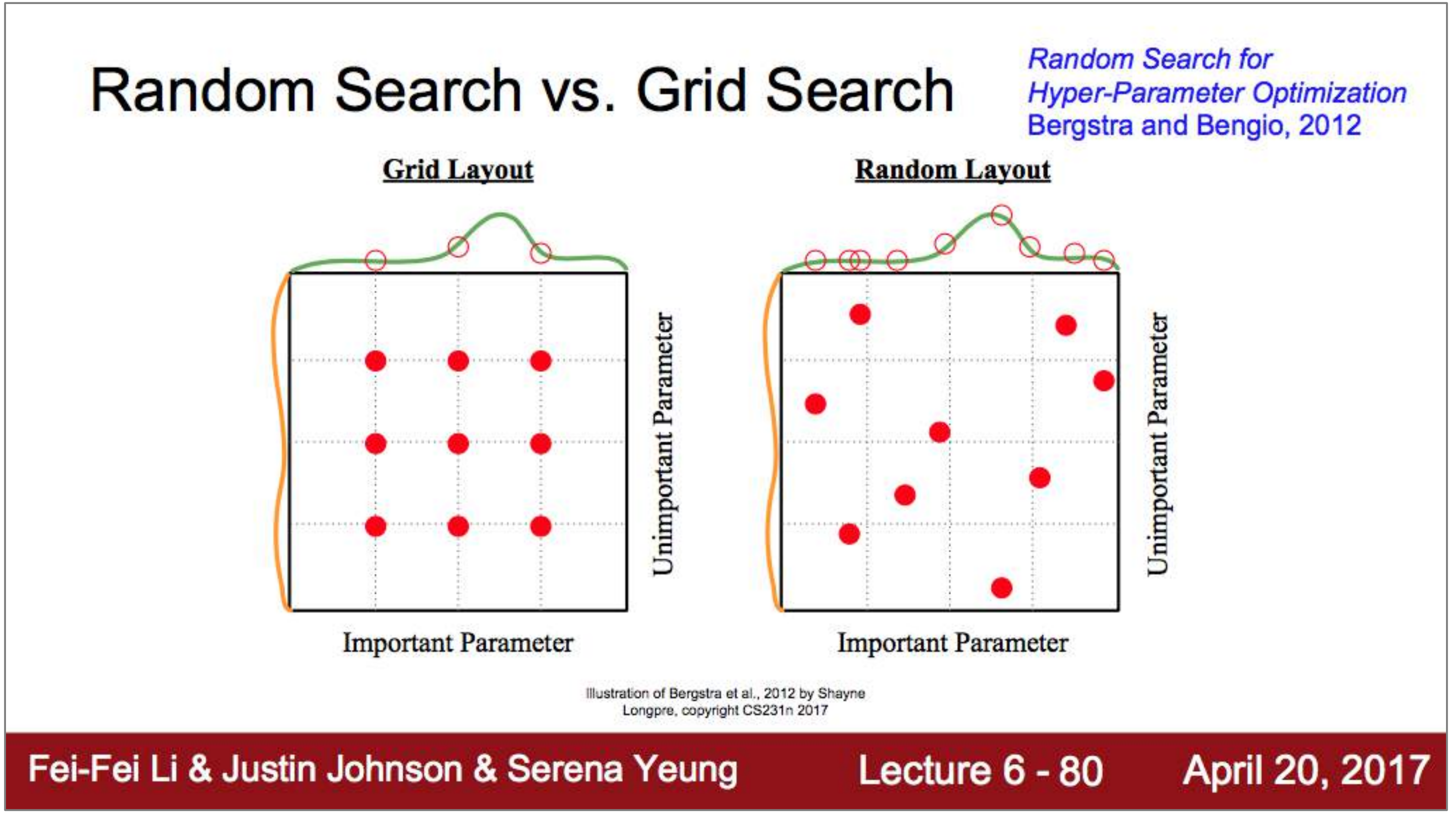
```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, 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

모두의연구소



적절한 하이퍼파라미터 값 찾기

모두의연구소

Hyperparameters to play with:

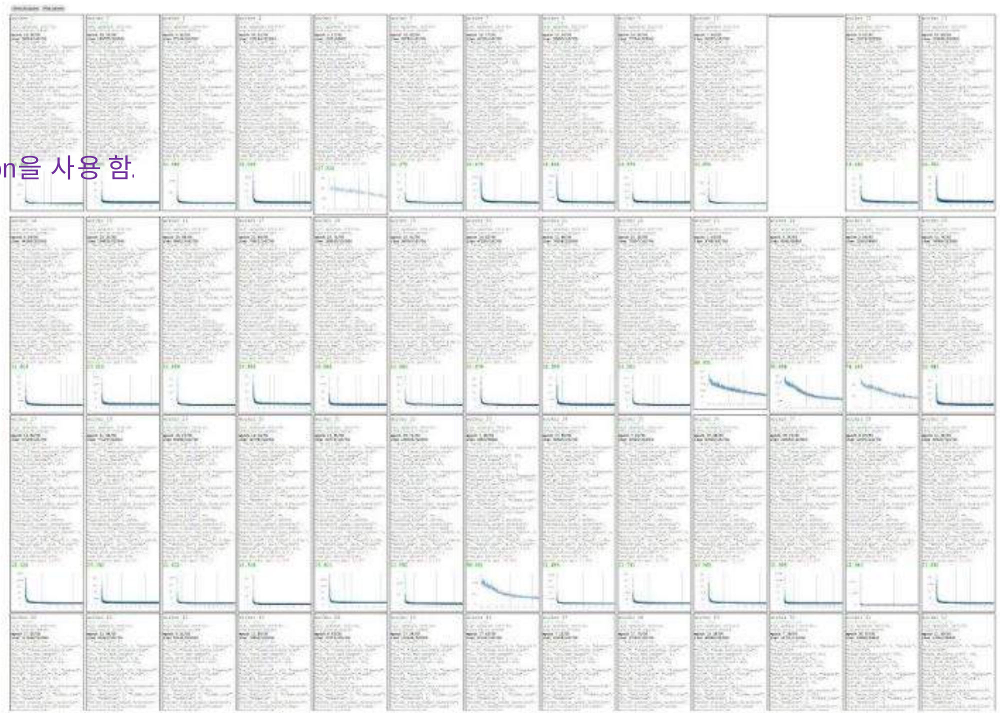
- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

neural networks practitioner
music = loss function

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 6 - 81   April 20, 2017

적절한 하이퍼파라미터 값 찾기

Cross-validation
“command center”



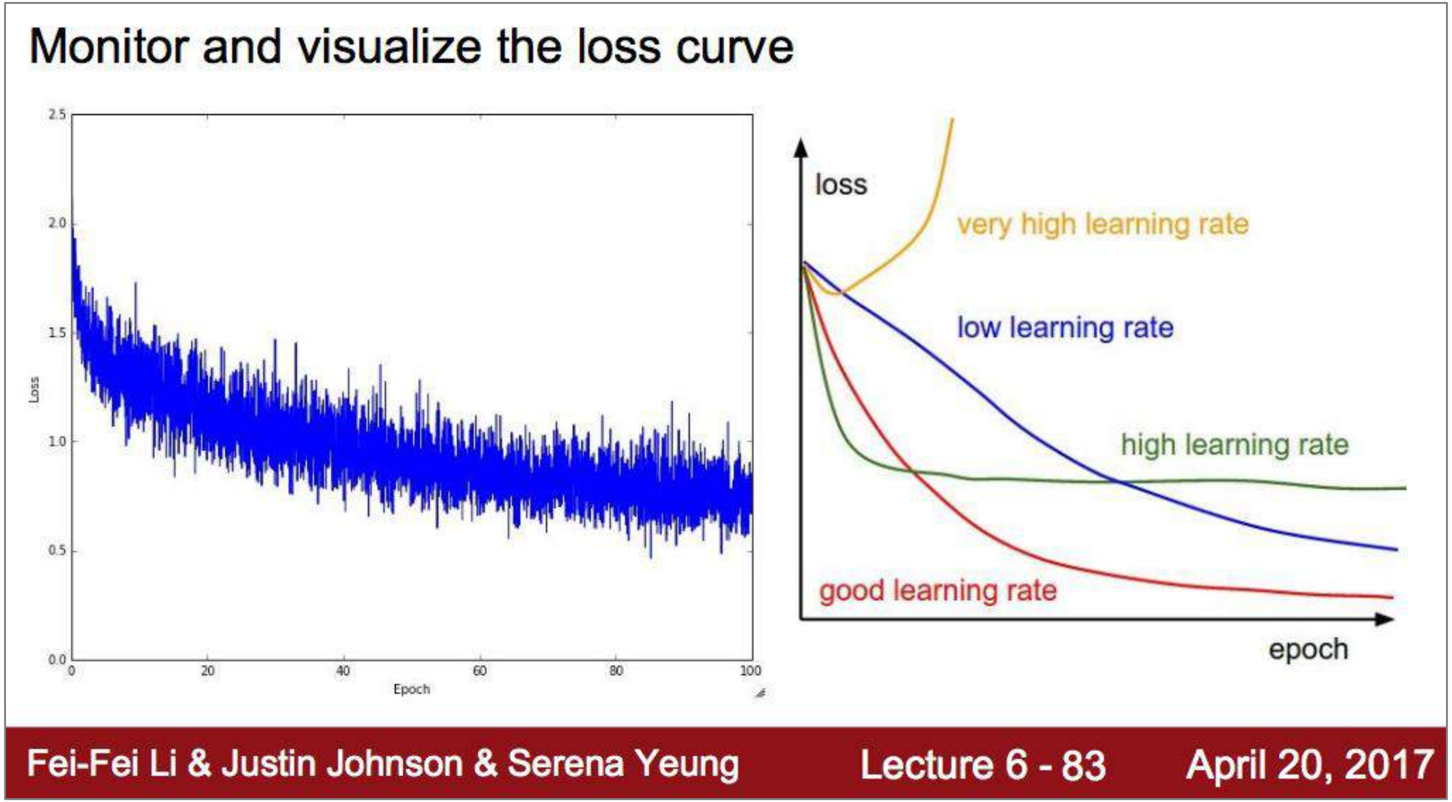
Data가 부족할 때 Cross-validation을 사용 함.

ex) 5 fold cross-validation

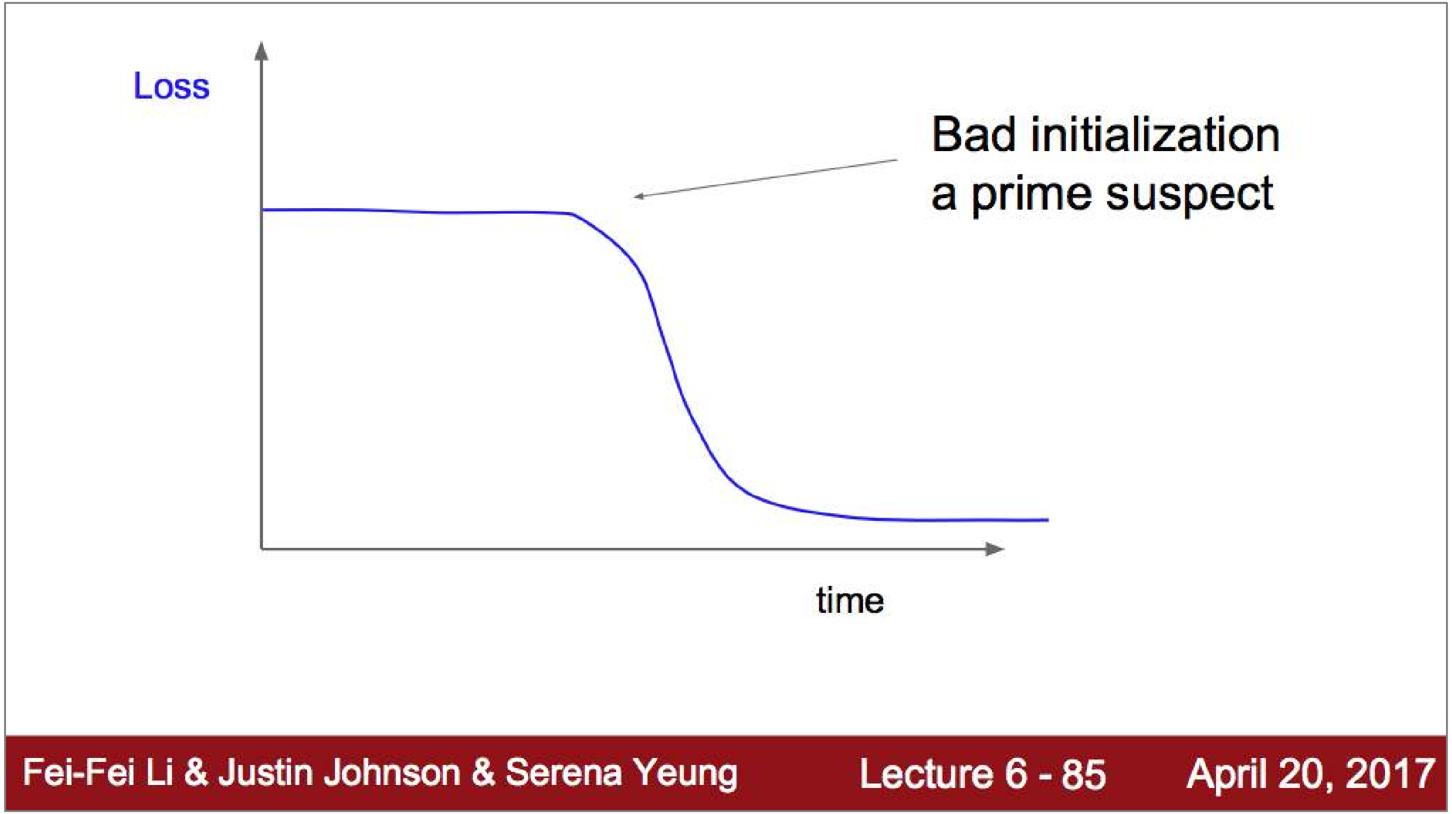
Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 82 April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소



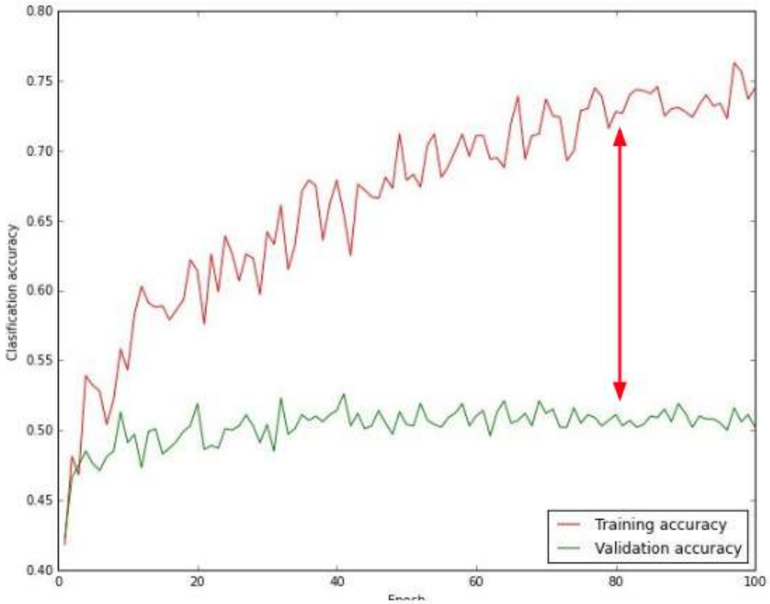
적절한 하이퍼파라미터 값 찾기



적절한 하이퍼파라미터 값 찾기

모두의연구소

Monitor and visualize the accuracy:



big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 6 - 86   April 20, 2017

적절한 하이퍼파라미터 값 찾기

모두의연구소

Track the ratio of weight updates / weight magnitudes:

```
# assume parameter vector W and its gradient vector dW
param_scale = np.linalg.norm(W.ravel())
update = -learning_rate*dW # simple SGD update
update_scale = np.linalg.norm(update.ravel())
W += update # the actual update
print update_scale / param_scale # want ~1e-3
```

ratio between the updates and values: ~ 0.0002 / 0.02 = 0.01 (about okay)
want this to be somewhere around 0.001 or so

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 87 April 20, 2017

적절한 하이퍼파라미터 값 찾기

Summary

TLDRs

We looked in detail at:

- Activation Functions (use ReLU)
- Data Preprocessing (images: subtract mean)
- Weight Initialization (use Xavier init)
- Batch Normalization (use)
- Babysitting the Learning process
- Hyperparameter Optimization
  (random sample hyperparams, in log space when appropriate)

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 6 - 88   April 20, 2017

정리

모두의연구소

- 매개변수 갱신 방법에는 확률적 경사 하강법(SGD) 외에도 모멘텀, AdaGrad, Adam 등이 있다
- 가중치 초깃값을 정하는 방법은 올바른 학습을 하는데 매우 중요하다
- 가중치 초 값으로는 'Xavier 초깃값'과 'He 초깃값'이 효과적이다
- 배치 정규화를 이용하면 학습을 빠르게 진행할 수 있으며, 초깃값에 영향을 덜 받게 된다
- 오버피팅을 억제하는 정규화 기술로는 가중치 감소와 드랍아웃이 있다
- 하이퍼파라미터 값 탐색은 최적 값이 존재할 법한 범위를 점차 좁히면서 하는 것이 효과적이다

모두의연구소

박은수 Research Director

E-mail : es.park@modulabs.co.kr