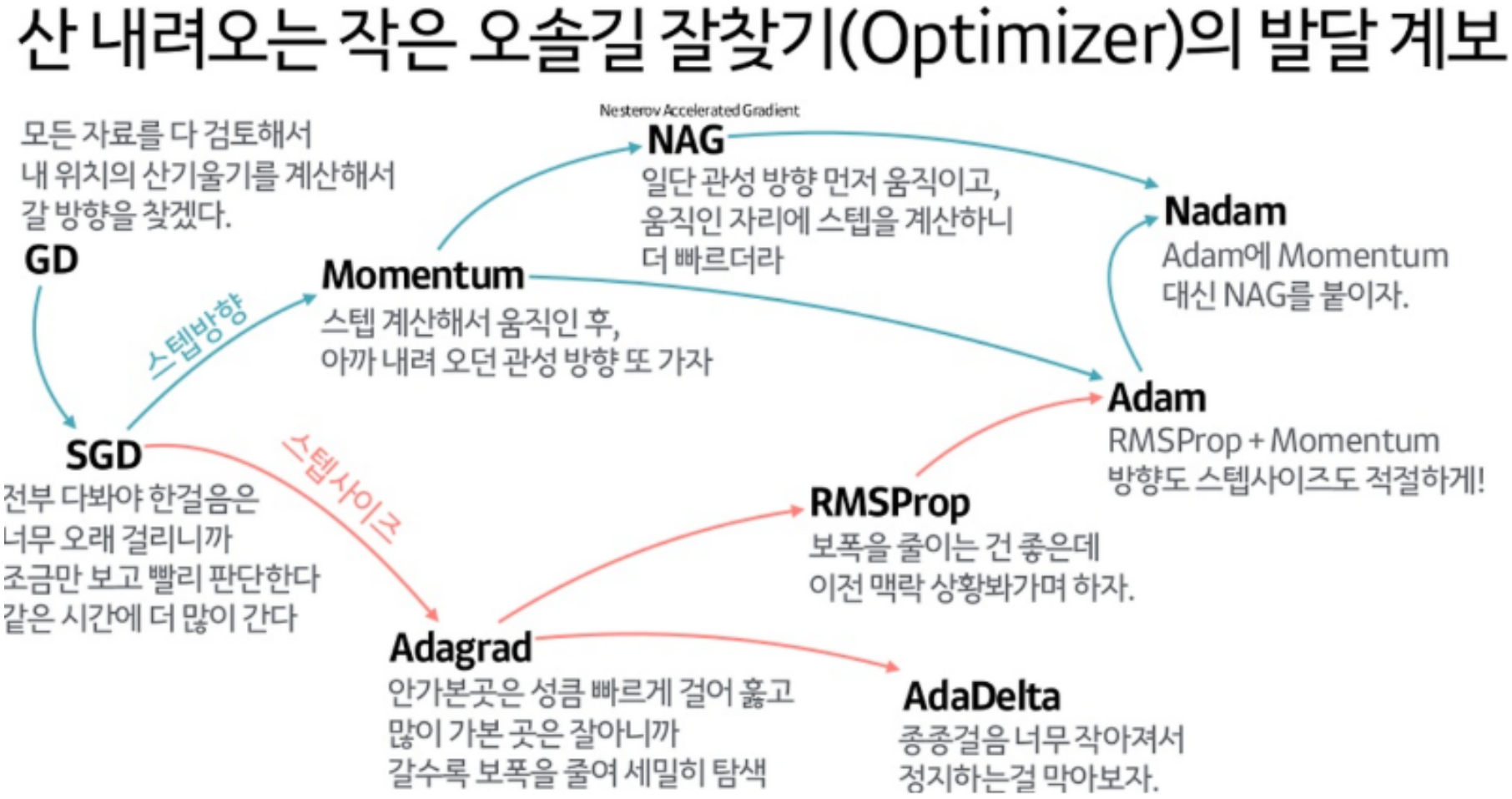
파이썬으로 배우는 딥러닝(Deep Learning)

5회차 수업
매개변수, 정규화, 드롭아웃... 실전 최적화 따라잡기

목 차

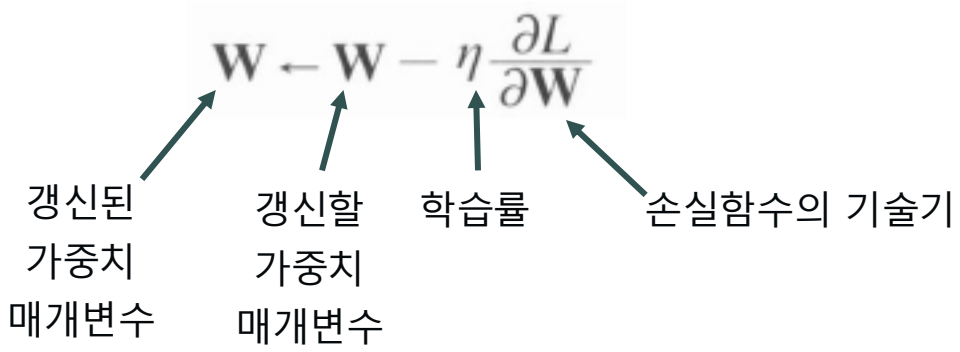
최적화 – Optimizer

- 다양한 Optimizer : 학습률과 기울기 결정방법에 따라 나누어짐.
- Adam Optimizer : Momentum과 RMSProp의 장점 사용



확률적 경사 하강법(Stochastic Gradient Descent)(1)

◆ SGD의 수식



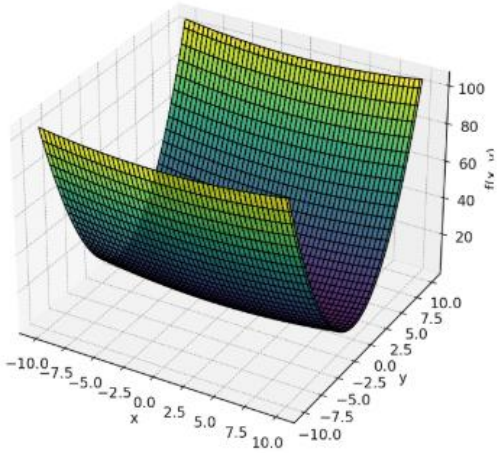
- 다변수 미분 이론에 바탕을 둔 가장 기본이 되는 optimizer
- 경사방향으로 움직일때 가장 빠른 최적화를 보이는 단순한 방식
- 학습률의 초기값에 따라 성능 편차가 큼

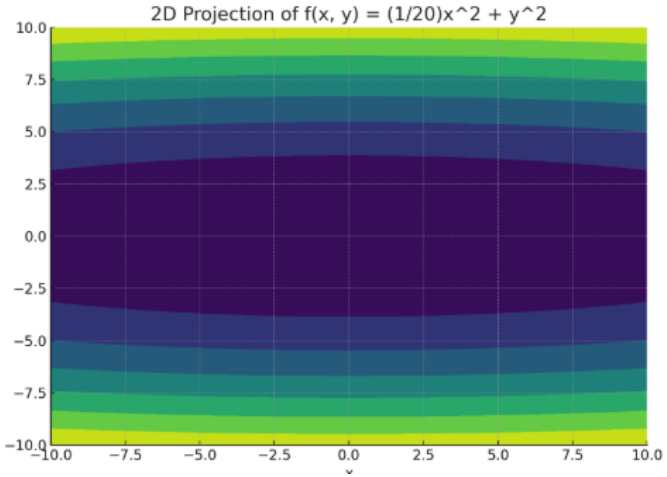
확률적 경사 하강법(Stochastic Gradient Descent)(2)

◆ SGD의 단점

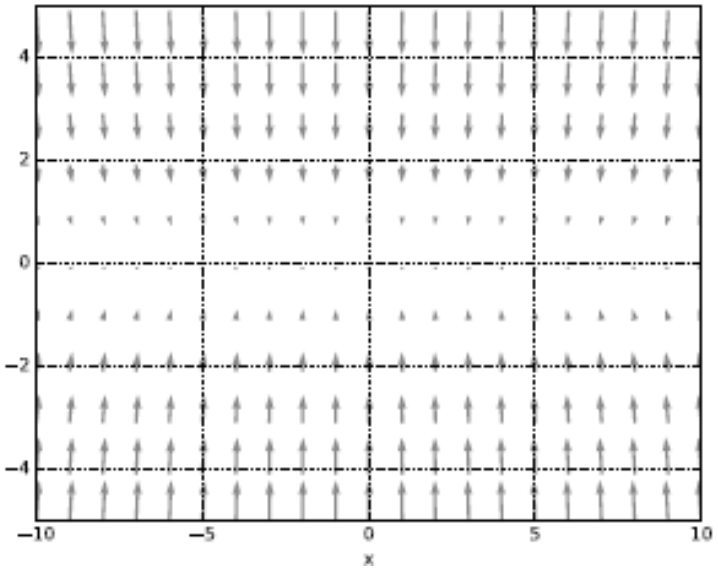
- $f(x,y) = \frac{1}{20}x^2 + y^2$ 함수의 예 :

  y축방향은 크고 x축 방향은 작음

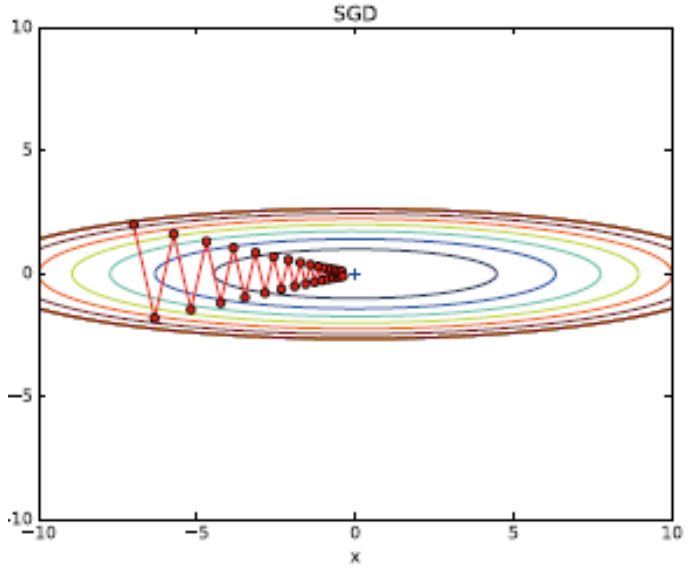




- 함수의 기울기와 최적화 갱신 경로



비효율적인
탐색 경로
유발



확률적 경사 하강법(Stochastic Gradient Descent)(3)

```
4   class SGD:
5
6       """확률적 경사 하강법(Stochastic Gradient Descent)"""
7
8       def __init__(self, lr=0.01):
9           self.lr = lr
10
11      def update(self, params, grads):
12          for key in params.keys():
13              params[key] -= self.lr * grads[key]
```

확률적 경사 하강법에 따라 가중치 업데이트

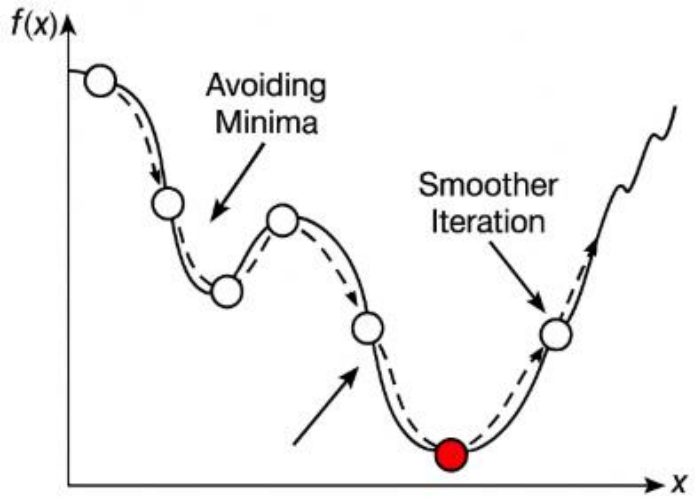
모멘텀(Momentum)(1)

◆ 모멘텀 기법의 수식 : Gradient descent에 현재 관성을 추가함

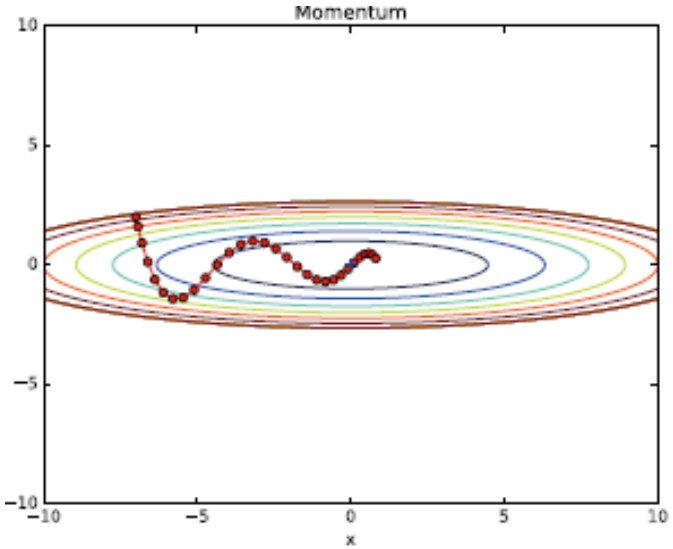
$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v} \qquad \mathbf{v} \leftarrow \alpha\mathbf{v} - \eta\frac{\partial L}{\partial \mathbf{W}}$$

- $\mathbf{W}$: 갱신된 가중치 매개변수
- $\mathbf{W}$: 갱신할 가중치 매개변수
- $\mathbf{v}$: 속도
- α : 반영률 eg.0.9
- $\mathbf{v}$: 이전 속도
- η : 학습률
- $\frac{\partial L}{\partial \mathbf{W}}$: 손실함수의 기울기

◆ 종종 극소점에 빠졌다가 관성의 힘으로 탈출 가능



◆ 모멘텀에 의한 최적화 갱신 경로



모멘텀(Momentum)(2)

```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]
```

모멘텀에 따라 가중치 업데이트

AdaGrad(1)

◆ AdaGrad 기법의 수식 : 변수마다 스텝마다 학습률이 바뀜

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

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

갱신된 가중치 매개변수

갱신할 가중치 매개변수

학습률

학습률 조정값(행렬)

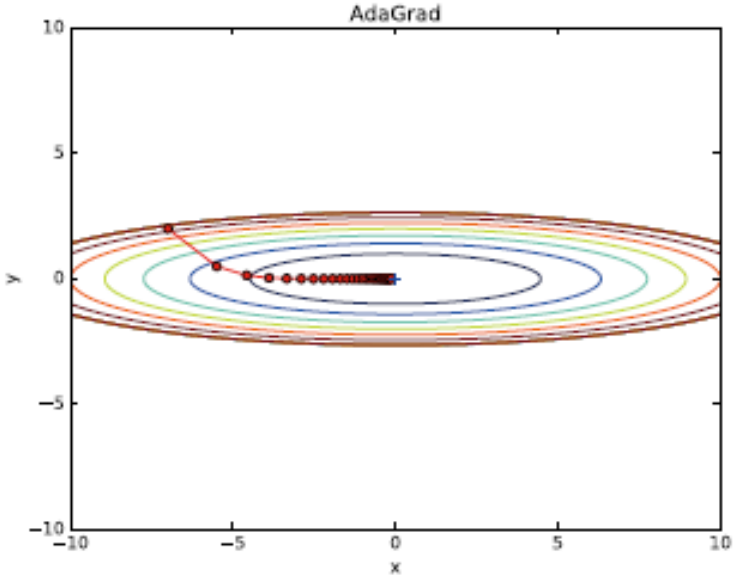
손실함수의 기술기

기존기울기값을 행렬제곱

◆ 학습률 조정값의 특성

- 각 축 별로 학습률을 조정하는 효과
- 큰 변화를 겪은 변수의 학습률은 대폭 작아짐
- 작은 변화를 겪은 변수의 학습률은 소폭 작아짐
- 단점 : 무한 학습시 갱신량이 0이되어 전혀 갱신X

AdaGrad 기법의 수식의 최적화 갱신 경로



AdaGrad(2)

```
59  class AdaGrad:
60
61      """AdaGrad"""
62
63      def __init__(self, lr=0.01):
64          self.lr = lr
65          self.h = None
66
67      def update(self, params, grads):
68          if self.h is None:
69              self.h = {}
70              for key, val in params.items():
71                  self.h[key] = np.zeros_like(val)
72
73          for key in params.keys():
74              self.h[key] += grads[key] * grads[key]
75              params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```

AdaGrad에 따라 가중치 업데이트

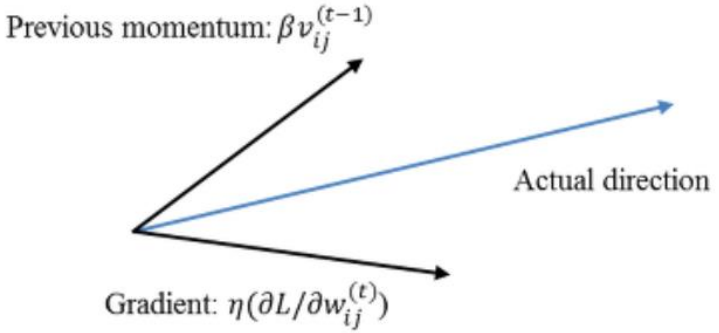
NAG (Nesterov Accelated Gradient)

- NAG 기법의 수식 : 현재위치에서 관성과 기울기 반대 방향을 합함

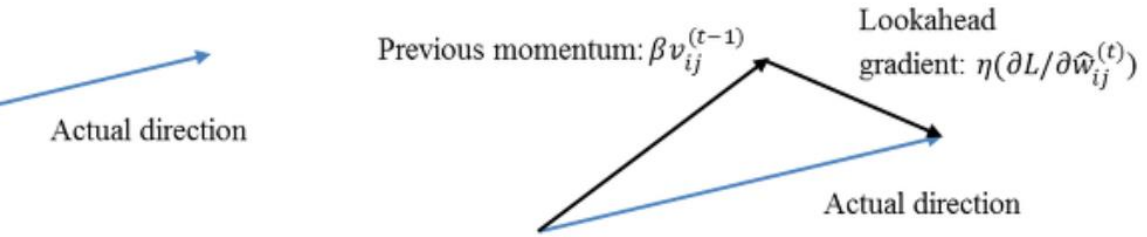
$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{v}_n$$

$$\mathbf{v}_n = \alpha \mathbf{v}_{n-1} - \eta \nabla f\left(\mathbf{x}_n + \alpha \mathbf{v}_{n-1}\right), \qquad \mathbf{v}_{-1} = \mathbf{0}$$

- Momentum을 공격적인 방식으로 변형
- 현재 위치에서의 관성과 관성방향으로 움직인 후 위치에서의 기울기 반대 방향을 합함.



(a) Momentum update



(b) NAG momentum update

RMSProp

- RMSProp 기법의 수식 : 이전 누적치와 현재 기울기의 좌표별 제곱의 가중치 평균을 보완

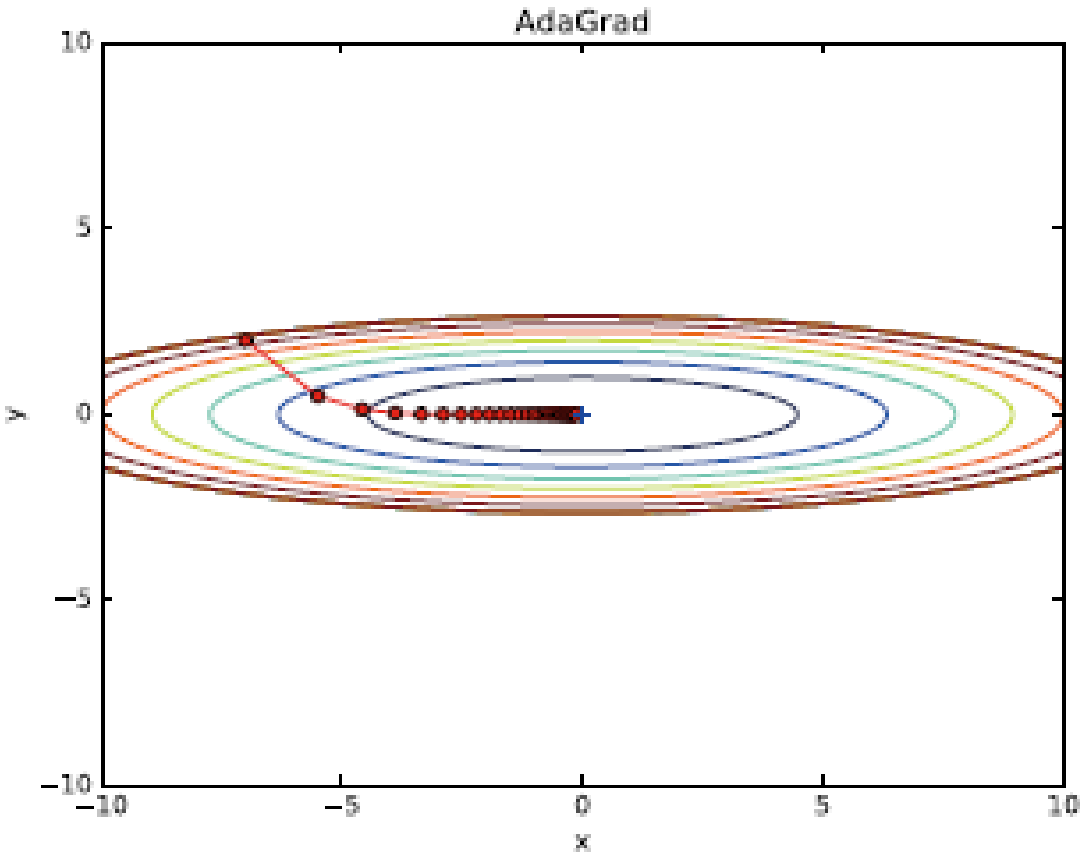
$$\mathbf{h}_n = \gamma \mathbf{h}_{n-1} + (1-\gamma)\nabla f(\mathbf{x}_n) \odot \nabla f(\mathbf{x}_n), \qquad \mathbf{h}_{-1} = \mathbf{0}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \frac{1}{\sqrt{\mathbf{h}_n}} \odot \nabla f(\mathbf{x}_n)$$

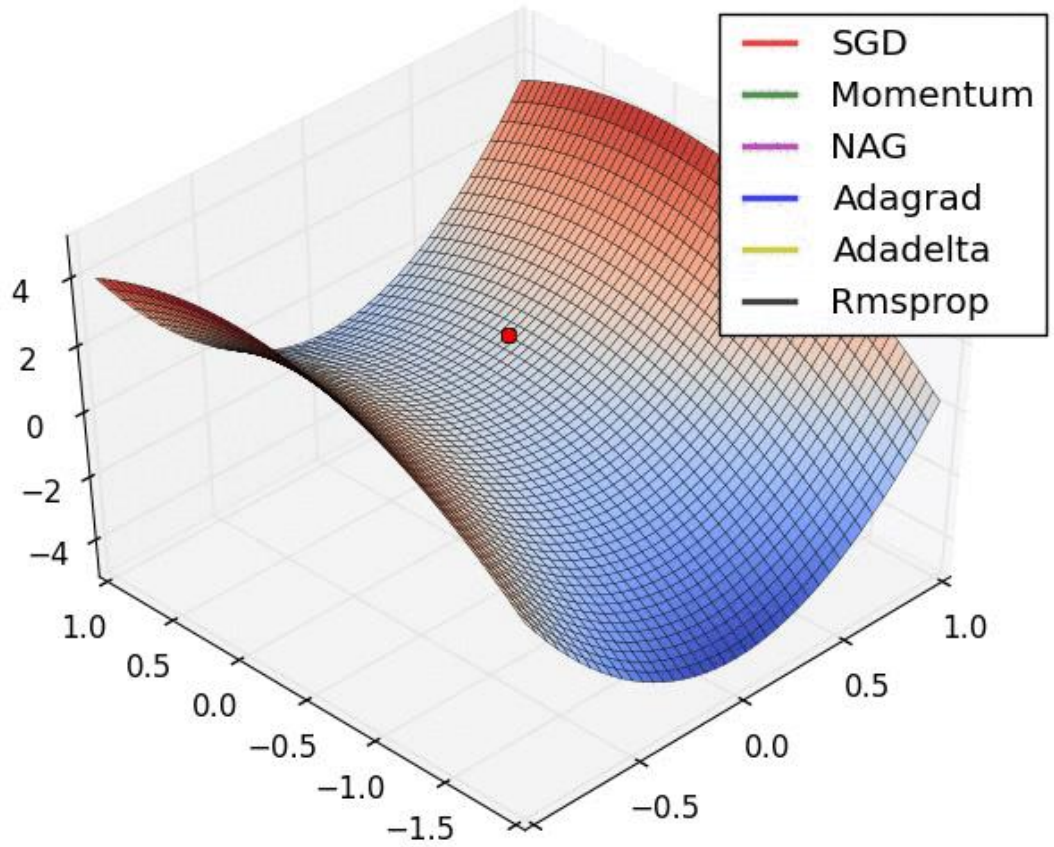
- AdaGrad는 스텝이 많이 진행되면 누적치 hn이 너무 커져서 학습률이 너무 작아져 학습이 거의 되지 않는 문제가 발생함.
- RMSProp은 이를 보완한 방법으로 γ 를 조정하여 학습률 조정값의 과거 값 반영 비율을 조정함.

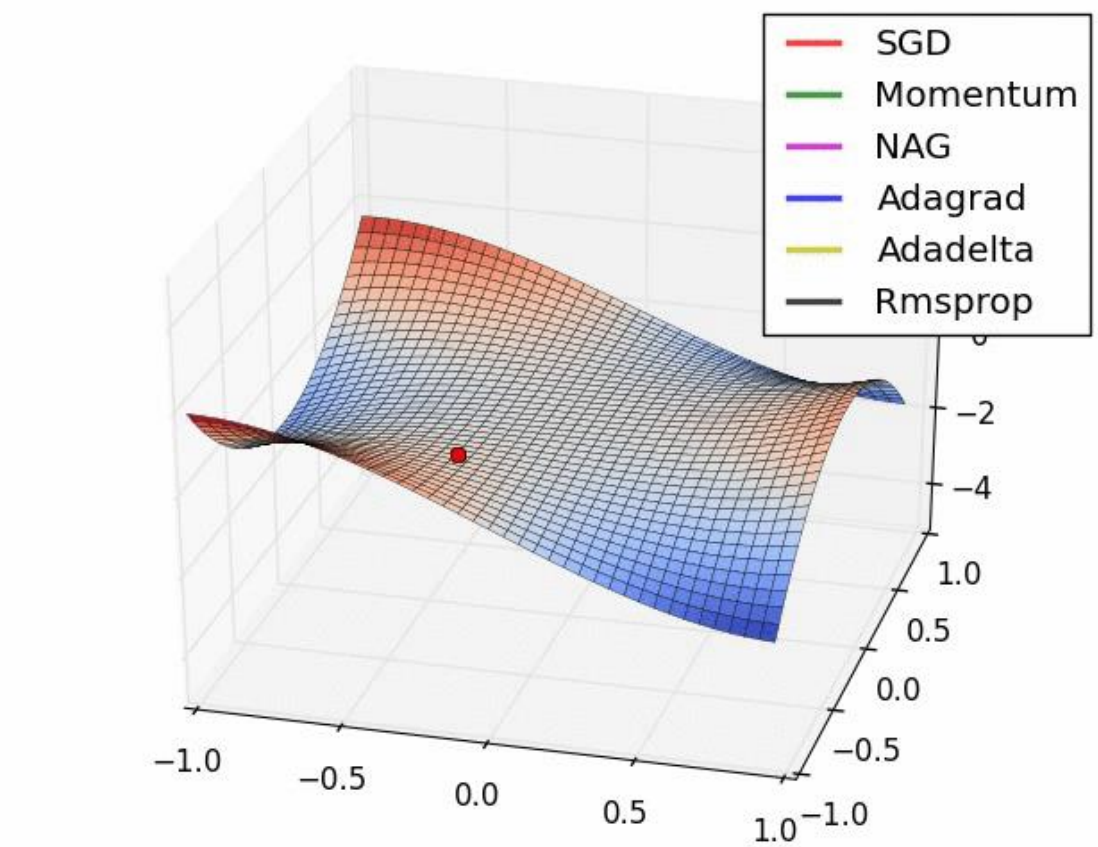
Adam(1)

- 속도를 조절하는 모멘텀과 매개변수의 원소마다 조정하는 AdaGrad를 융합한 방법
- 경우에 따라 다르지만 일반적으로 가장 최고의 성능을 보여주는 Optimizer
- 하이퍼 파라미터의 '변향 보정' 이 진행됨.



어떤 갱신 방법을 사용할 것인가?





<실습 과제>

각 Optimizer를 활용하여 최적화를 수행하는 프로그램을 구현하라

참고 > ch05/optimizer_compare_naive.py 에 작성함.

실습 과제

실습: ch05/optimizer_compare_naive.ipynb

```
1   import os, sys
2   print(os.getcwd())
3   current_dir = os.path.dirname(os.getcwd())
4   print(current_dir)
5   os.chdir(current_dir)
6
7   import numpy as np
8   import matplotlib.pyplot as plt
9   from collections import OrderedDict
10  from common.optimizer import *
11
12
13  def f(x, y):
14      return x**2 / 20.0 + y**2
15
16
17  def df(x, y):
18      return x / 10.0, 2.0*y
19
20  init_pos = (-7.0, 2.0)
21  params = {}
22  params['x'], params['y'] = init_pos[0], init_pos[1]
23  grads = {}
24  grads['x'], grads['y'] = 0, 0
25
26
27  optimizers = OrderedDict()
28  optimizers["SGD"] = SGD(lr=0.95)
29  optimizers["Momentum"] = Momentum(lr=0.1)
30  optimizers["AdaGrad"] = AdaGrad(lr=1.5)
31  optimizers["Adam"] = Adam(lr=0.3)
32
33  idx = 1
```

미분전의 함수

Optimizer 종류별로 Ordered Dictionary에 정리해둠

실습 과제

실습: ch05/optimizer_compare_naive.ipynb

```
for key in optimizers:
    optimizer = optimizers[key]
    x_history = []
    y_history = []
    params['x'], params['y'] = init_pos[0], init_pos[1]

    for i in range(30):
        x_history.append(params['x'])
        y_history.append(params['y'])

        grads['x'], grads['y'] = df(params['x'], params['y'])
        optimizer.update(params, grads)


    x = np.arange(-10, 10, 0.01)
    y = np.arange(-5, 5, 0.01)

    X, Y = np.meshgrid(x, y)
    Z = f(X, Y)

    # 외곽선 단순화
    mask = Z > 7
    Z[mask] = 0

    # 그래프 그리기
    plt.subplot(2, 2, idx)
    idx += 1
    plt.plot(x_history, y_history, 'o-', color="red")
    plt.contour(X, Y, Z)
    plt.ylim(-10, 10)
    plt.xlim(-10, 10)
    plt.plot(0, 0, '+')
    #colorbar()
    #spring()
    plt.title(key)
    plt.xlabel("x")
    plt.ylabel("y")

plt.show()
```

각 좌표에 따른 기울기를 구함

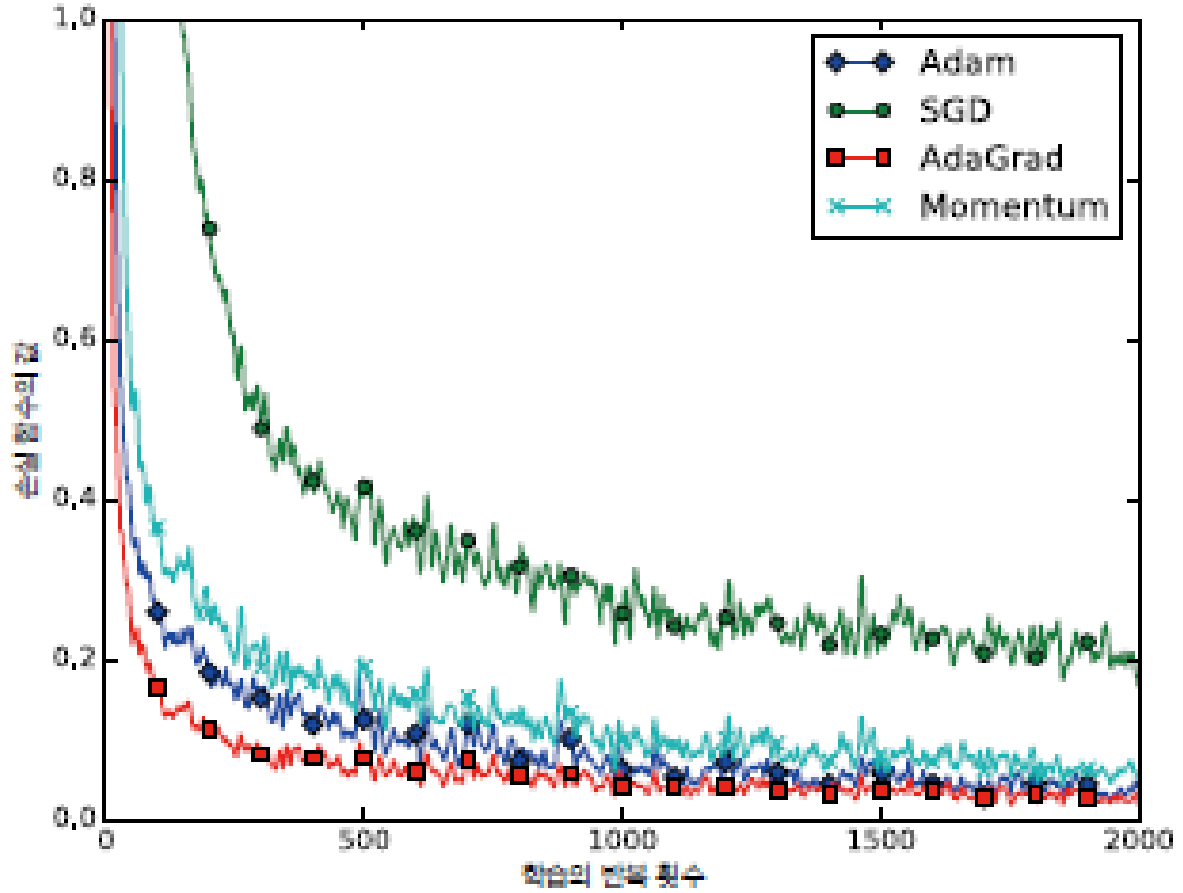
Optimizer 별로 기울기를 업데이트함

비어있는 2차원 평면 좌표 만듦

등고선 상에서 기울기 업데이트 History를 표현함

MNIST 데이터 셋으로 본 갱신 방법 비교

- SGD의 학습 진도가 가장 느리고, 나머지 세 시법의 진도는 비슷.



- 주의할 점 : 학습률과 신경망의 구조(층, 깊이)에 따라 결과가 달라질 수 있음.

실습 과제

<실습 과제>

각 Optimizer를 활용하여 MNIST 데이터 학습시 최적화를 수행하는 프로그램을 구현하라

참고 > ch05/optimizer_compare_mnist.ipynb 에 작성함.

실습 과제

실습: ch05/optimizer_compare_mnist.ipynb

```python
import os, sys
print(os.getcwd())
current_dir = os.path.dirname(os.getcwd())
print(current_dir)
os.chdir(current_dir)

import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.util import smooth_curve
from common.multi_layer_net import MultiLayerNet
from common.optimizer import *


# 0. MNIST 데이터 읽기==========
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)

train_size = x_train.shape[0]
batch_size = 128
max_iterations = 2000


# 1. 실험용 설정==========
optimizers = {}
optimizers['SGD'] = SGD()
optimizers['Momentum'] = Momentum()
optimizers['AdaGrad'] = AdaGrad()
optimizers['Adam'] = Adam()
#optimizers['RMSprop'] = RMSprop()

networks = {}
train_loss = {}
for key in optimizers.keys():
    networks[key] = MultiLayerNet(
        input_size=784, hidden_size_list=[100, 100, 100, 100],
        output_size=10)
    train_loss[key] = []
```

필요한 Library를 Import 함

숫자 데이터 다운로드
from internet

Optimizer를 종류별로 정의함

Optimizer별로 신경망을 생성함

실습 과제

실습: ch05/optimizer_compare_mnist.ipynb

```
38
39  # 2. 훈련 시작==========
40  for i in range(max_iterations):
41      batch_mask = np.random.choice(train_size, batch_size)
42      x_batch = x_train[batch_mask]
43      t_batch = t_train[batch_mask]
44
45      for key in optimizers.keys():
46          grads = networks[key].gradient(x_batch, t_batch)
47          optimizers[key].update(networks[key].params, grads)
48
49          loss = networks[key].loss(x_batch, t_batch)
50          train_loss[key].append(loss)
51
52      if i % 100 == 0:
53          print( "===========" + "iteration:" + str(i) + "===========")
54          for key in optimizers.keys():
55              loss = networks[key].loss(x_batch, t_batch)
56              print(key + ":" + str(loss))
57
58
59  # 3. 그래프 그리기==========
60  markers = {"SGD": "o", "Momentum": "x", "AdaGrad": "s", "Adam": "D"}
61  x = np.arange(max_iterations)
62  for key in optimizers.keys():
63      plt.plot(x, smooth_curve(train_loss[key]), marker=markers[key], markevery=100, label=key)
64  plt.xlabel("iterations")
65  plt.ylabel("loss")
66  plt.ylim(0, 1)
67  plt.legend()
68  plt.show()
69
```

Train data 에서 배치 크기만큼 데이터 추출

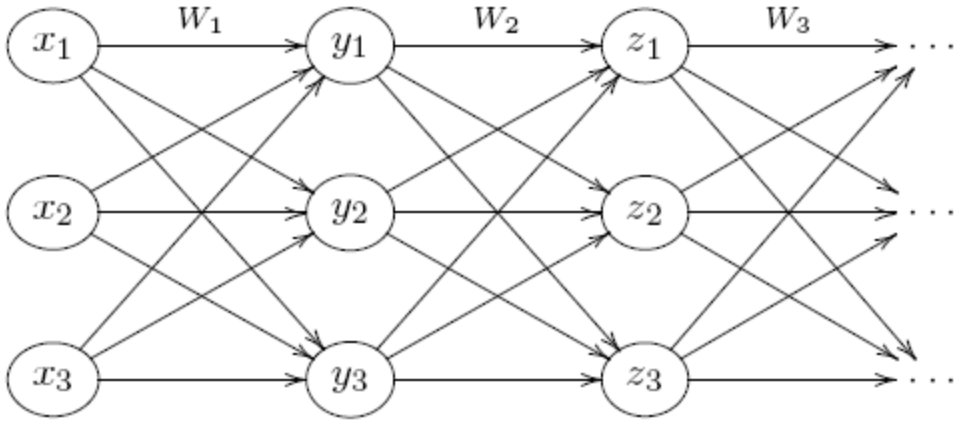
기울기를 추출하여 Optimizer별로 가중치를 Update함

목 차

가중치의 초기값 설정(1)

◆ 초기값을 0이나 균일한 값으로 설정할 경우

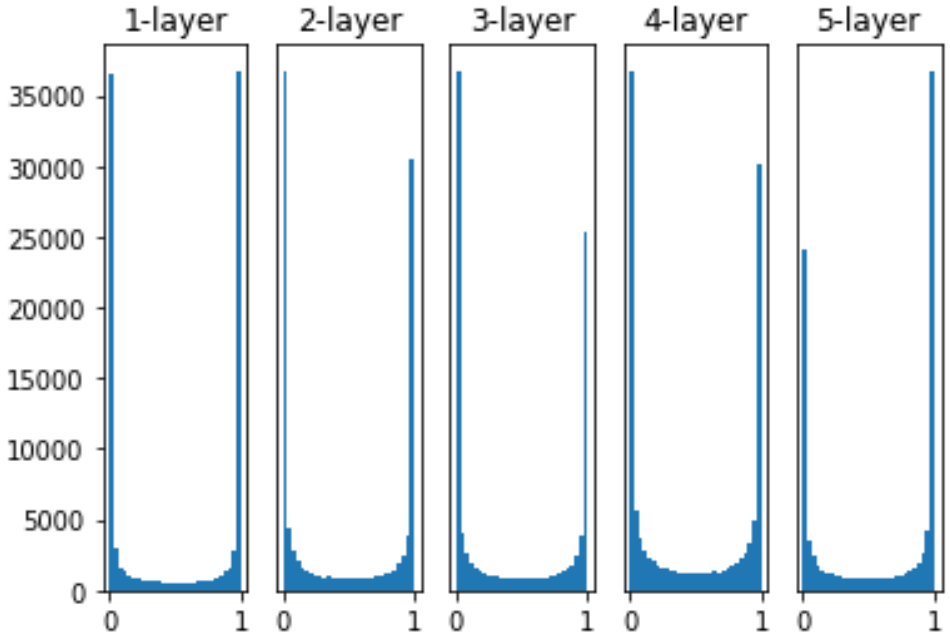
- 학습이 올바로 이뤄지지 않음
- 오차역전파법에서 모든 가중치의 값이 똑같이 갱신되기 때문

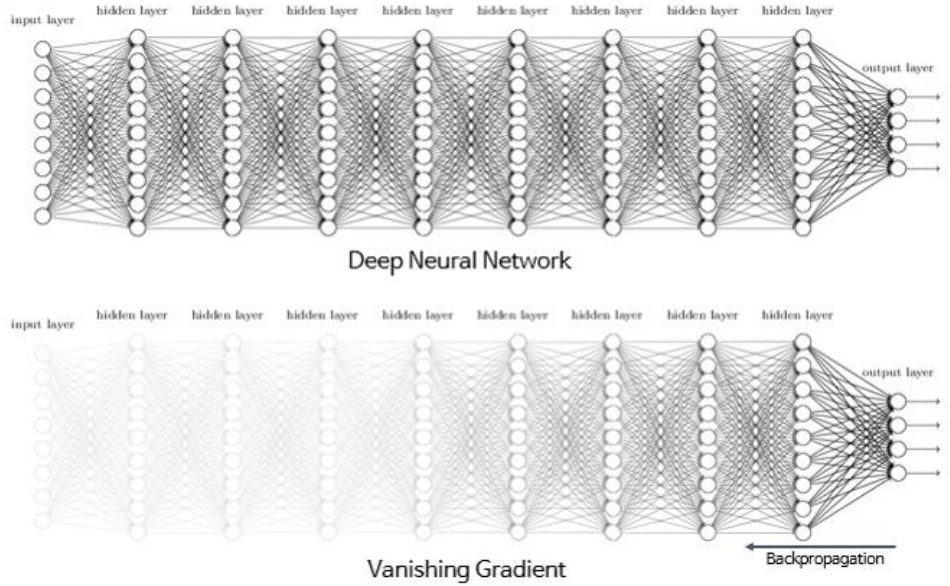


- 가중치의 대칭적인 구조를 무너뜨리기 위해 초깃값을 무작위로 설정해야 함.

가중치의 초기값 설정(2)

◆ 가중치를 표준편차1인 정규분포로 초기화할때 각 층의 활성화 값 분포



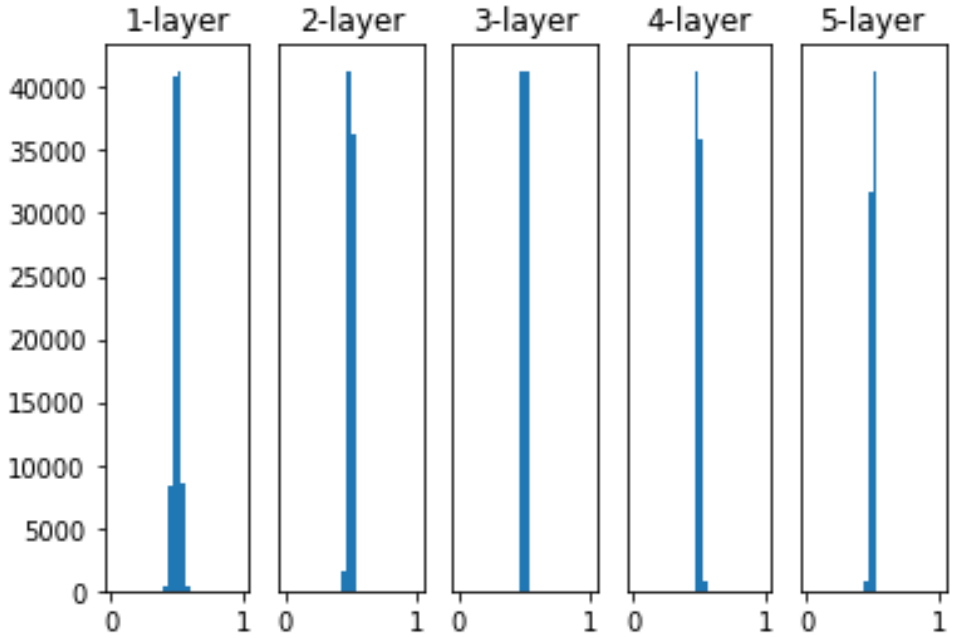


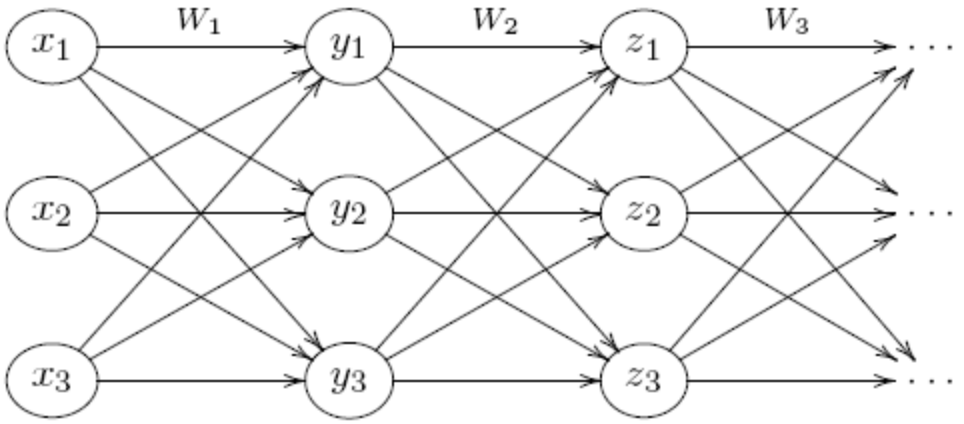
Vanishing Gradient 문제 발생

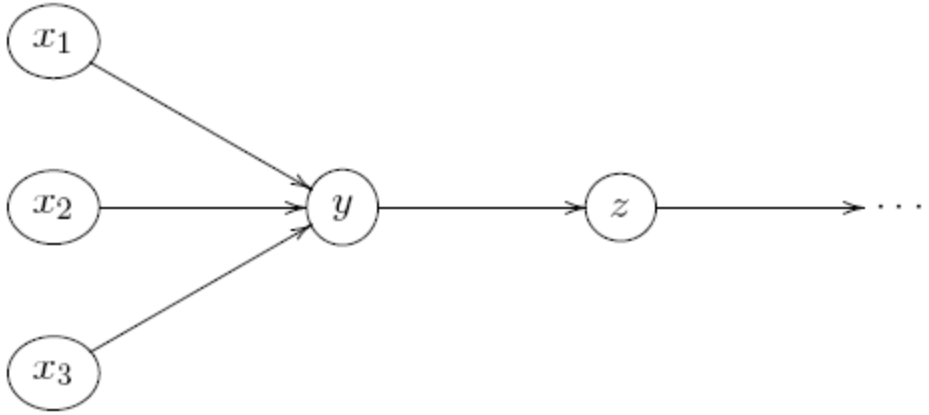
이유: Sigmoid 활성화 값들이 0과 1에 치우쳐 있어서 미분값은 0에 다가가기 때문

가중치의 초기값 설정(3)

◆ 가중치를 평균이 0, 표준편차0.01인 정규분포로 초기화할때 각 층의 활성화 값 분포







표현력 제한 문제 발생

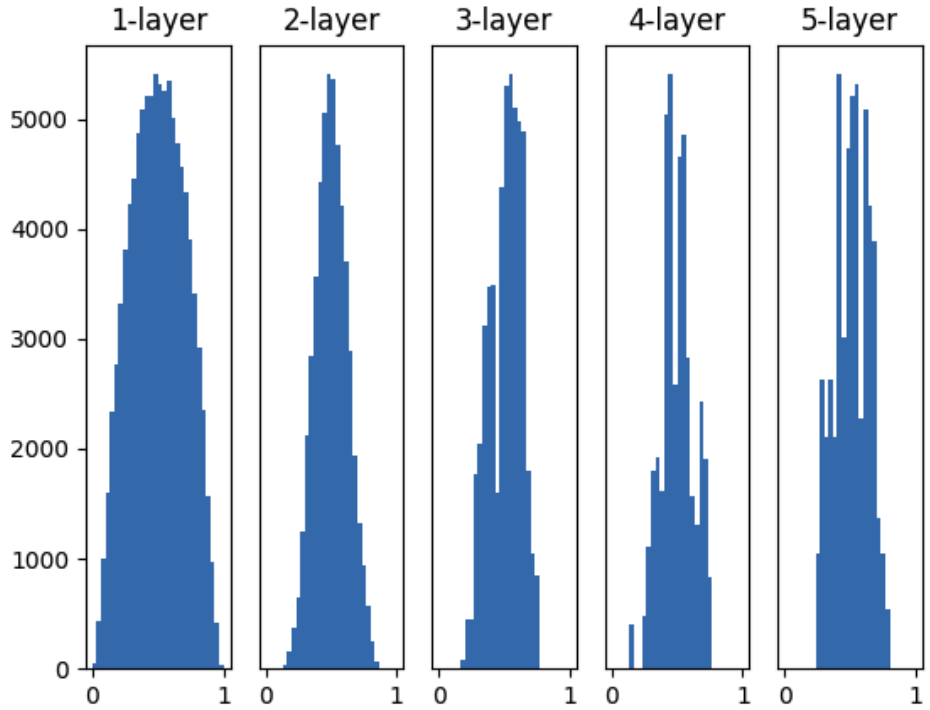
이유: 다수의 뉴런이 거의 같은 값을 출력해서
한개의 뉴런과 차이 없음

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

가중치의 초기값 설정(4)

◆ Xavier 초기값 : 각 층의 활성화 값들을 광범위하게 분포시키는 것이 목표

◆ 각 층의 활성화 값(Weight 값)을 표준편차가 $\frac{1}{\sqrt{n}}$ 인 정규분표로 초기화




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

- 층이 깊어지면서 넓게 분포됨.
- 각 층에 흐르는 데이터는 적당히 퍼져있음.
- 시그모이드 함수 표현력에 제한받지않고
- 학습이 효율적으로 이루어질 수 있음.

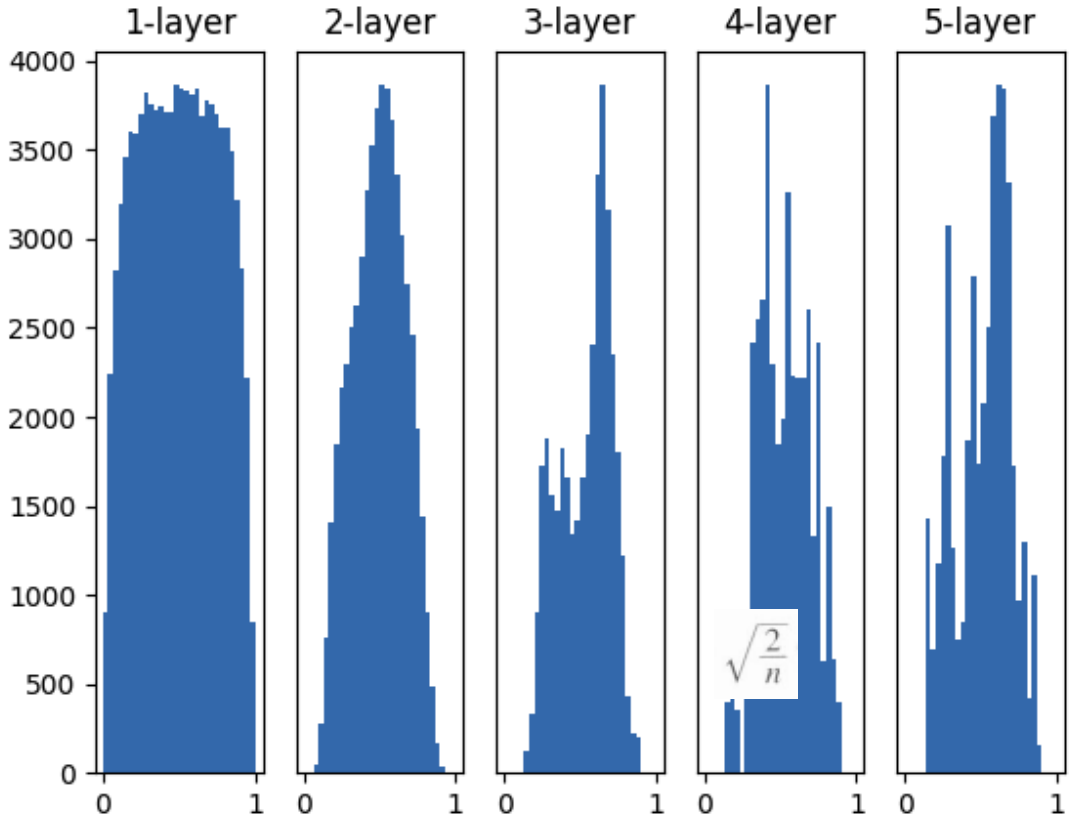
가중치의 초기값 설정(5)

◆ ReLU를 사용할때의 가중치 초기값 : He 초기값

◆ 각 층의 활성화 값(Weight 값)을 표준편차가 $\frac{2}{\sqrt{n}}$ 인 정규분표로 초기화

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

```
38      # 활성화 함수도 바꿔가며 실험해보자!
39      #z = sigmoid(a)
40      z = ReLU(a)
41      #z = tanh(a)
```



<실습 과제>

가중치의 초기값과 활성화 함수를 변경해가며 가중치의 히스토그램을 확인하는 프로그램을 구현하라

참고 > ch05/weight_init_activation_histogram.ipynb 에 작성함.

실습 과제

실습: ch05/weight_init_activation_histogram.ipynb

```python
# coding: utf-8
import numpy as np
import matplotlib.pyplot as plt


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


def ReLU(x):
    return np.maximum(0, x)


def tanh(x):
    return np.tanh(x)

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

x = input_data

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

    # 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)

```

Sigmoid 함수 선언

ReLU 함수 선언

이전 노드의 출력을 입력값으로 사용

가중치 초기값이 표준편차 1

가중치 초기값이 표준편차 0.01

Xavier 초기값

He 초기값

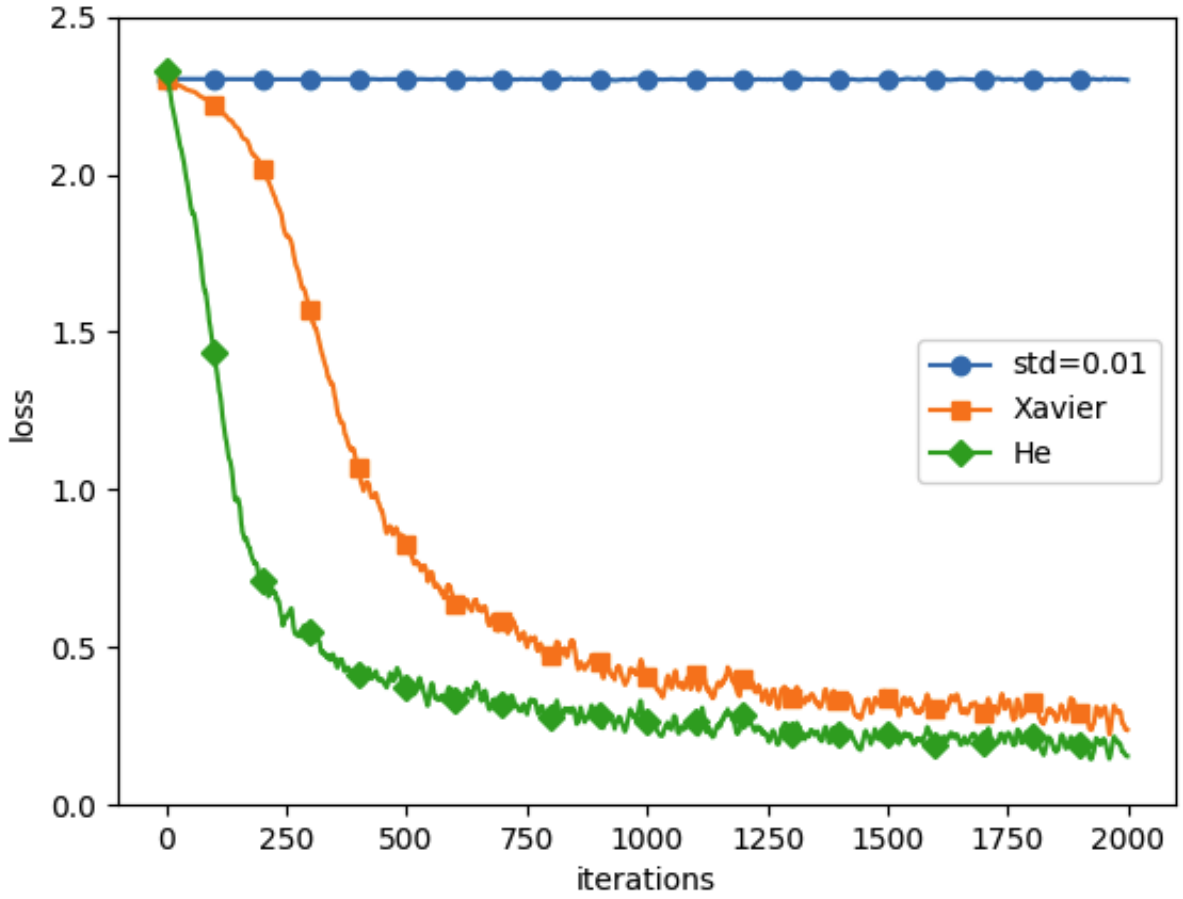
실습 과제

실습: ch05/weight_init_activation_histogram.ipynb

```
34
35        a = np.dot(x, w)
36
37
38        # 활성화 함수도 바꿔가며 실험해보자!
39        z = sigmoid(a)
40        #z = ReLU(a)
41        #z = tanh(a)
42
43        activations[i] = z
44
45    # 히스토그램 그리기
46    for i, a in activations.items():
47        plt.subplot(1, len(activations), i+1)
48        plt.title(str(i+1) + "-layer")
49        if i != 0: plt.yticks([], [])
50        # plt.xlim(0.1, 1)
51        # plt.ylim(0, 7000)
52        plt.hist(a.flatten(), 30, range=(0,1))
53    plt.show()
54
```

MNIST 데이터셋으로 본 가중치 초기값 비교

◆ 층별 뉴런수 100개, 5층 신경망, ReLU 활성화 함수



- std 0.01 일때 → 학습이 전혀 이뤄지지 않음.
  활성화 값의 분포처럼 순전파때 너무 작은 값으로 흐르고 역전파때 기울기가 작아져 가중치가 거의 갱신되지 않음.
- He/Xavier → 학습이 순조롭게 이루어짐
- He --> Xavier 대비 학습 진도가 더 빠름.

실습 과제

<실습 과제>

표준편차 0.01, Xavier, He 초기값 각각

에 대해 학습속도를 측정해보자

참고 > ch05/weight_init_compare.ipynb 에 작성함.

실습 과제

실습: ch05/weight_init_compare.ipynb

```
1   import os, sys
2   print(os.getcwd())
3   current_dir = os.path.dirname(os.getcwd())
4   print(current_dir)
5   os.chdir(current_dir)
6
7   import numpy as np
8   import matplotlib.pyplot as plt
9   from dataset.mnist import load_mnist
10  from common.util import smooth_curve
11  from common.multi_layer_net import MultiLayerNet
12  from common.optimizer import SGD
13
14
15  # 0. MNIST 데이터 읽기==========
16  (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)
17
18  train_size = x_train.shape[0]
19  batch_size = 128
20  max_iterations = 2000
21
22
23  # 1. 실험용 설정==========
24  weight_init_types = {'std=0.01': 0.01, 'Xavier': 'sigmoid', 'He': 'relu'}
25  optimizer = SGD(lr=0.01)
26
27  networks = {}
28  train_loss = {}
29  for key, weight_type in weight_init_types.items():
30      networks[key] = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100],
31                                    output_size=10, weight_init_std=weight_type)
32      train_loss[key] = []
33
```

필요한. Library를 import

숫자 데이터 다운로드

Weight 초기값 종류별 지정

초기값 별로 신경망 생성

실습 과제

실습: ch05/weight_init_compare.ipynb

```
34
35  # 2. 훈련 시작==========
36  for i in range(max_iterations):
37      batch_mask = np.random.choice(train_size, batch_size)
38      x_batch = x_train[batch_mask]
39      t_batch = t_train[batch_mask]
40
41      for key in weight_init_types.keys():
42          grads = networks[key].gradient(x_batch, t_batch)
43          optimizer.update(networks[key].params, grads)
44
45          loss = networks[key].loss(x_batch, t_batch)
46          train_loss[key].append(loss)
47
48      if i % 100 == 0:
49          print("===========" + "iteration:" + str(i) + "===========")
50          for key in weight_init_types.keys():
51              loss = networks[key].loss(x_batch, t_batch)
52              print(key + ":" + str(loss))
53
54
55  # 3. 그래프 그리기==========
56  markers = {'std=0.01': 'o', 'Xavier': 's', 'He': 'D'}
57  x = np.arange(max_iterations)
58  for key in weight_init_types.keys():
59      plt.plot(x, smooth_curve(train_loss[key]), marker=markers[key], markevery=100, label=key)
60  plt.xlabel("iterations")
61  plt.ylabel("loss")
62  plt.ylim(0, 2.5)
63  plt.legend()
64  plt.show()
```

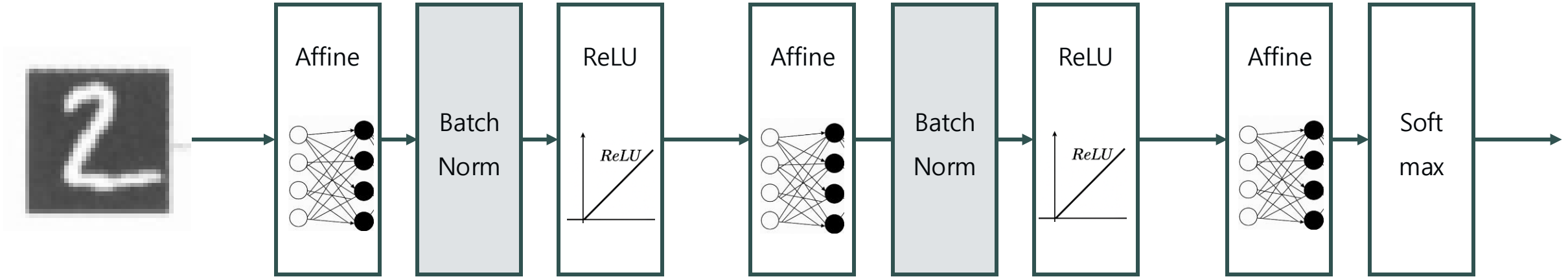
기울기를 구하고 가중치 업데이트

학습 100회마다 손실값 출력

목 차

배치 정규화 알고리즘(1)

◆ 기본 아이디어 : 각 층에서의 활성화 값이 적당히 분포되도록 조정하는 것. 입력값 평균 0, 분산 1로 조정



- 미니배치 m개의 입력 데이터 집합의 평균과 분산 → $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \quad \sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
- 활성화 함수 입력값의 정규화 → $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$
- 최종 출력시 γ 는 확대, β는 이동을 담당 → $y_i \leftarrow \gamma \hat{x}_i + \beta$
- γ와 β는 최적의 성능을 낼 수 있도록 학습 과정에서 머신이 조정함.

목 차

오버피팅(Over-fitting) 현상

◆ 훈련 데이터에만 너무 적응해 버려서 시험데이터에 제대로 대응하지 못하는 현상

◆ 매개변수에 비해 상대적으로 훈련 데이터 수가 적을때 발생함

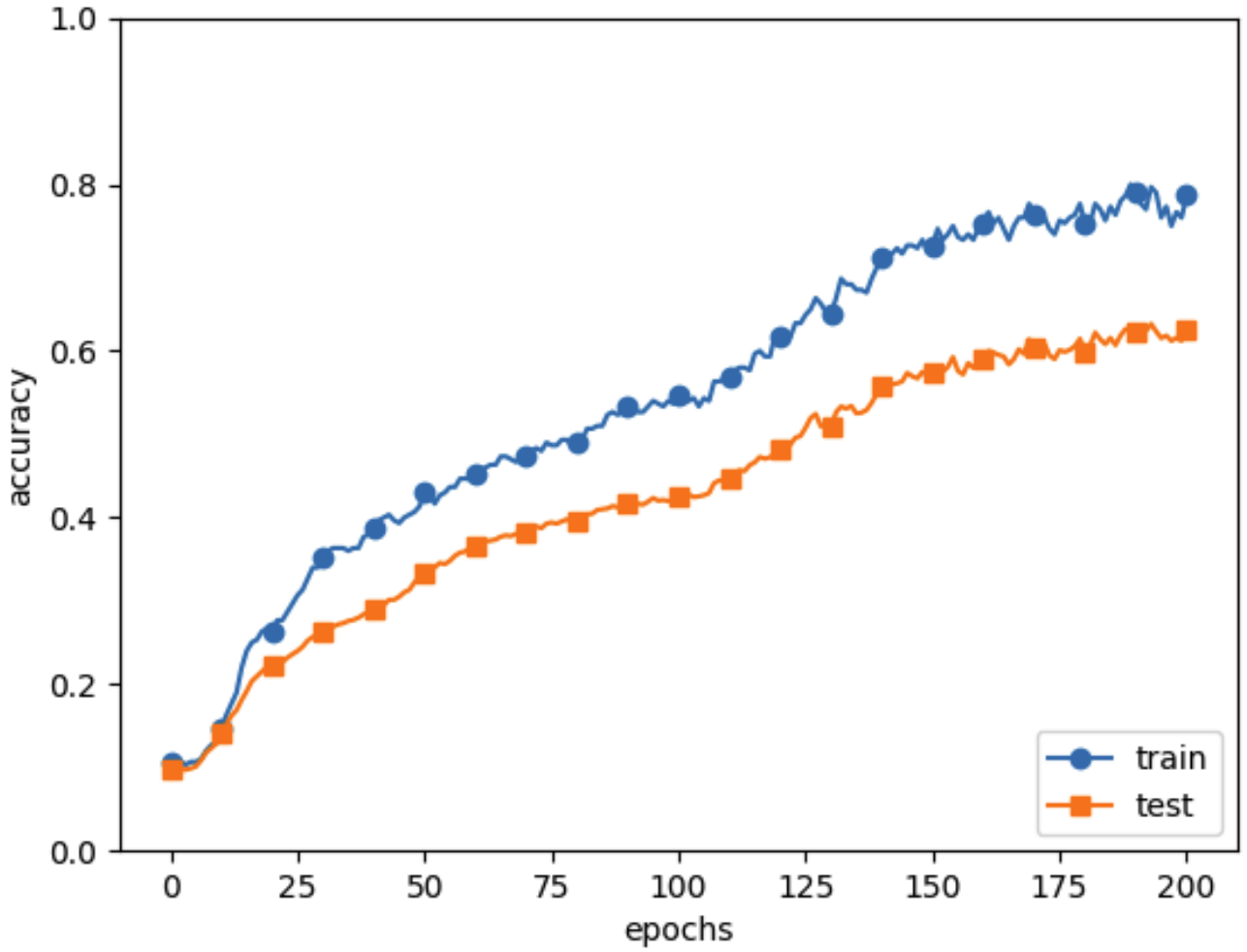
◆ 예제

- MNIST 데이터셋 60,000개중 훈련데이터로 300개만 사용
- 7층 네트워크를 사용하여 네트워크 복잡성 증가시킴
- 각 층의 뉴런은 100개, 활성화 함수 ReLU 사용
- 학습데이터 성공률은 높으나

  테스트데이터 성공률이 지나치게 낮게 나옴

오버피팅(Over-fitting) 억제 : 가중치 감소(Weight decay)

◆ 학습과정에서 큰 가중치에 대해서는 그에 상응하는 큰 페널티 부여하여 오버피팅 억제



<실습 과제>

가중치 감소를 적용한 신경망으로

학습 및 테스트를 구현하자

참고 > ch05/ovefit_weight_decay.ipynb 에 작성함.

실습 과제

실습: ch05/overfit_weight_decay.ipynb

```python
import os, sys
print(os.getcwd())
current_dir = os.path.dirname(os.getcwd())
print(current_dir)
os.chdir(current_dir)

import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.multi_layer_net import MultiLayerNet
from common.optimizer import SGD

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

# 오버피팅을 재현하기 위해 학습 데이터 수를 줄임
x_train = x_train[:300]
t_train = t_train[:300]

# weight decay(가중치 감쇠) 설정 =======================
#weight_decay_lambda = 0 # weight decay를 사용하지 않을 경우
weight_decay_lambda = 0.1
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10,
                        weight_decay_lambda=weight_decay_lambda)
optimizer = SGD(lr=0.01) # 학습률이 0.01인 SGD로 매개변수 갱신

max_epochs = 201
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
train_acc_list = []
test_acc_list = []
```

가중치에 페널티를 부여하는 옵션

실습 과제

실습: ch05/overfit_weight_decay.ipynb

```
35
36  iter_per_epoch = max(train_size / batch_size, 1)
37  epoch_cnt = 0
38
39  for i in range(1000000000):
40      batch_mask = np.random.choice(train_size, batch_size)
41      x_batch = x_train[batch_mask]
42      t_batch = t_train[batch_mask]
43
44      grads = network.gradient(x_batch, t_batch)
45      optimizer.update(network.params, grads)
46
47      if i % iter_per_epoch == 0:
48          train_acc = network.accuracy(x_train, t_train)
49          test_acc = network.accuracy(x_test, t_test)
50          train_acc_list.append(train_acc)
51          test_acc_list.append(test_acc)
52
53          print("epoch:" + str(epoch_cnt) + ", train acc:" + str(train_acc) + ", test acc:" + str(test_acc))
54
55          epoch_cnt += 1
56          if epoch_cnt >= max_epochs:
57              break
58
59
60  # 그래프 그리기==========
61  markers = {'train': 'o', 'test': 's'}
62  x = np.arange(max_epochs)
63  plt.plot(x, train_acc_list, marker='o', label='train', markevery=10)
64  plt.plot(x, test_acc_list, marker='s', label='test', markevery=10)
65  plt.xlabel("epochs")
66  plt.ylabel("accuracy")
67  plt.ylim(0, 1.0)
68  plt.legend(loc='lower right')
69  plt.show()
```

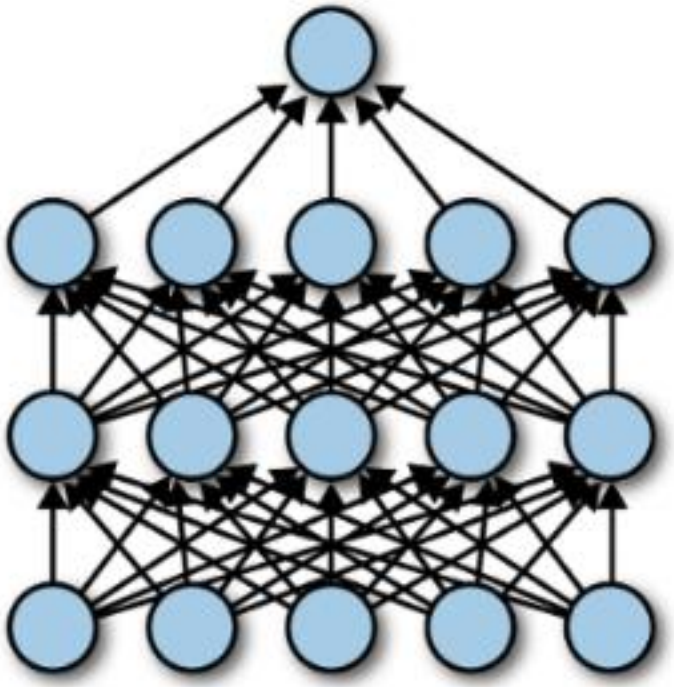
기울기를 구하고 가중치 업데이트

1주기당 정확도 계산

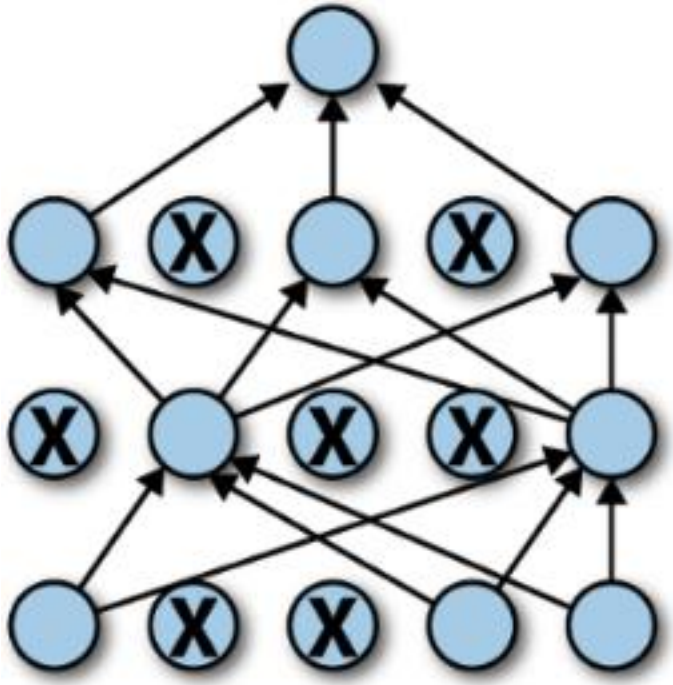
오버피팅(Over-fitting) 억제 : 드롭아웃(Drop-out)

◆ 훈련 때 은닉층의 뉴런을 무작위로 골라 삭제하는 방법. 즉, 신호를 전달하지 않음

일반적인 신경망



드롭아웃을 적용한
신경망



<실습 과제>

드롭아웃을 적용한 신경망으로

학습 및 테스트를 구현하자

참고 > ch05/ovefit_dropout.ipynb 에 작성함.

실습 과제

실습: ch05/overfit_weight_decay.ipynb

```
import os, sys
print(os.getcwd())
current_dir = os.path.dirname(os.getcwd())
print(current_dir)
os.chdir(current_dir)

import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.multi_layer_net_extend import MultiLayerNetExtend
from common.trainer import Trainer

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

# 오버피팅을 재현하기 위해 학습 데이터 수를 줄임
x_train = x_train[:300]
t_train = t_train[:300]

# 드롭아웃 사용 유무와 비율 설정 ========================
use_dropout = True  # 드롭아웃을 쓰지 않을 때는 False
dropout_ratio = 0.2
# ====================================================
```

Dropout 비율 설정

실습 과제

실습: ch05/overfit_weight_decay.ipynb

```
network = MultiLayerNetExtend(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100],
                              output_size=10, use_dropout=use_dropout, dropout_ration=dropout_ratio)
trainer = Trainer(network, x_train, t_train, x_test, t_test,
                  epochs=301, mini_batch_size=100,
                  optimizer='sgd', optimizer_param={'lr': 0.01}, verbose=True)
trainer.train()

train_acc_list, test_acc_list = trainer.train_acc_list, trainer.test_acc_list

# 그래프 그리기==========
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, marker='o', label='train', markevery=10)
plt.plot(x, test_acc_list, marker='s', label='test', markevery=10)
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

신경망에 Dropout 옵션 설정

학습 수행

학습데이터로 테스트시 정확도

테스트데이터로 테스트시 정확도

Parameter vs. Hyperparameter

◆ Parameter

- 데이터를 통해 머신이 학습하는 값
- 예를 들어 가중치, 편향, Scaling, Shift 가 있음.
- 데이터가 많고 신경망 모델이 좋으면 최적의 값을 머신이 스스로 찾아냄

◆ Hyperparameter

- 사람이 결정하는 값
- 예를 들어, 층의 개수, 뉴런의 개수, 학습률, 손실함수의 종류, 배치 크기, 훈련회수,

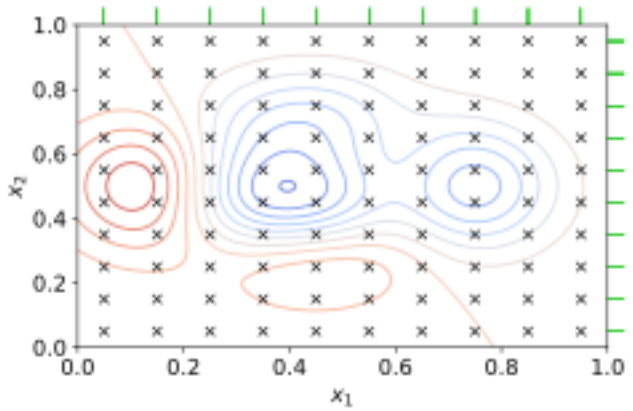
  Optimizer의 종료와 관련 계수값, 가중치 초기값, 가중치 감소 계수, Dropout비율 등
- 좋은 값을 찾기 위해서는 여러 값을 직접 시도해보는 수 밖에 없음.
- 일부 Hyperparameter는 특정 값이 추천되거나 거의 고정되어 있는 경우도 있음.

  예. Momentum계수(0.9), Adam($\beta_1 = 0.9, \beta_2 = 0.999$) 등

Hyperparameter 탐색

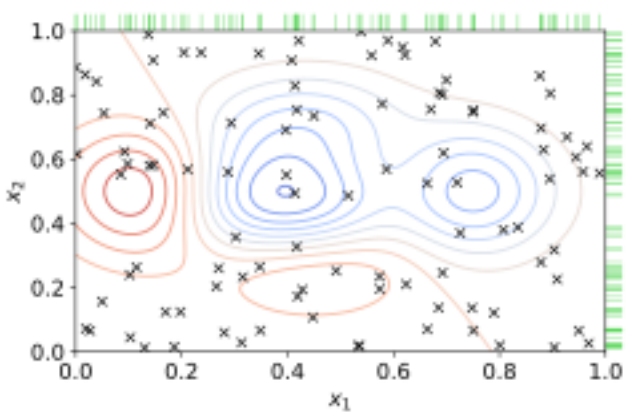
◆ Hyperparameter는 중요도 차이가 큼 → 격자 보다는 무작위 선택이 나음.

Grid search



<

Random search



◆ 성능이 가장 좋은 조합 주위의 영역을 좀인해서 다시 선택함

Bayesian search