밑바닥부터 시작하는 딥러닝

박 철

과정 개요

- ✓ 딥러닝을 프레임워크 없이 구현
- ✓ 교재의 소스 위주로 분석하면서 이론 설명
- ✓ Colab으로 소스 실행
- ✓ 딥러닝강좌를 역사를 순으로 진행

- ✓ 목표 : 밑바닥부터 시작하는 딥러닝 코드의 구조와 각각의 기능을 이해하는 것

교재 사이트

https://github.com/WegraLee/deep-learning-from-scratch/blob/master/common/layers.py

WegraLee Update README.md	82457ee on 16 Nov 2020	**205** commits
ch01	이미지 교체	2 years ago
ch02	Update README.md	5 years ago
ch03	Update README.md	5 years ago
ch04	Update README.md	5 years ago
ch05	Create README.md	5 years ago
ch06	오류 수정	3 years ago
ch07	레나 이미지를 대체 이미지로 교체	10 months ago
ch08	Create README.md	5 years ago
common	미세한 값(델타) 1e-7 추가	4 years ago
dataset	4채널 이미지를 1채널 이미지로 교체	10 months ago
.gitignore	add src files	5 years ago
LICENSE.md	Add License	5 years ago
README.md	Update README.md	9 months ago
cover_image.jpg	새 표지 등록	5 years ago
equations_and_figures.zip	Add files via upload	4 years ago
map.png	개앞맵시 프리뷰 사진 업로드	5 years ago

코드 구조 요약

Dataset	Network Model	Softmax	Cost 함수 Loss 함수 Object 함수	Gradient	Optimizer
텐서로 관리됨 2차원/4차원 데이터 전처리		Back propagation	mean_squared_error cross_entropy_error	Back propagation	Gradient descent

MNIST 데이터 셋 사용
영상인 경우 openCV

multi_layer_net
multi_layer_net_extend

SoftmaxWithLoss

SGD:
Momentum:
Nesterov
AdaGrad
RMSprop
Adam

Layers
- **Affine**
- **Relu**
- **Sigmoid**
- **Convolution**
- **Pooling**
- **SoftmaxWithLoss**
- **Dropout**
- **BatchNormalization**

각 mothod 마다
forwad와 backward가 있음

W b

기존 파일에서 로딩
훈련을 통하여 생성

텐서로 관리됨
Filter/kernel
pkl 파일로 관리

목차

Functions.py/gradient.py/util.py/mnist.py

common/functions.py

identity_function(x):
step_function(x):
sigmoid(x):
sigmoid_grad(x):
relu(x):
relu_grad(x):
softmax(x):
mean_squared_error(y, t):
cross_entropy_error(y, t):
softmax_loss(X, t):

common/gradient.py

_numerical_gradient_1d(f, x):
numerical_gradient_2d(f, X):
numerical_gradient(f, x):

common/util.py

smooth_curve(x):
shuffle_dataset(x, t):
conv_output_size(input_size, filter_size, stride=1, pad=0):
im2col(input_data, filter_h, filter_w, stride=1, pad=0):
col2im(col, input_shape, filter_h, filter_w, stride=1, pad=0):

/dataset/mnist.py

```
def _download(file_name):
def download_mnist():
def _load_label(file_name):
def _load_img(file_name):
def _convert_numpy():
def init_mnist():
def _change_one_hot_label(X):
def load_mnist(normalize=True,
                        flatten=True,
                        one_hot_label=False):
```

Layers.py

Class로 구현됨

common/layers.py

```
class Relu:
    __init__(self):
    forward(self, x):
    backward(self, dout):

class Sigmoid:
    __init__(self):
    forward(self, x):
    backward(self, dout):

class Affine:
    __init__(self, W, b):
    forward(self, x):
    backward(self, dout):

class SoftmaxWithLoss:
    __init__(self):
    forward(self, x, t):
    backward(self, dout=1):

class Dropout:
    __init__(self, dropout_ratio=0.5):
    forward(self, x, train_flg=True):
    backward(self, dout):
```

```
class BatchNormalization:
    __init__(self, gamma, beta, momentum=0.9,
                running_mean=None, running_var=None):
    forward(self, x, train_flg=True):
    __forward(self, x, train_flg):
    backward(self, dout):
    __backward(self, dout):

class Convolution:
    __init__(self, W, b, stride=1, pad=0):
    forward(self, x):
    backward(self, dout):

class Pooling:
    __init__(self, pool_h, pool_w, stride=1, pad=0):
    forward(self, x):
    backward(self, dout):
```

multi_layer_net.py & multi_layer_net_extend.py

common/multi_layer_net.py

```
class MultiLayerNet:
    __init__(self, input_size, hidden_size_list, output_size,
             activation='relu', weight_init_std='relu', weight_decay_lambda=0):
    __init_weight(self, weight_init_std):
    predict(self, x):
    loss(self, x, t):
    accuracy(self, x, t):
    numerical_gradient(self, x, t):
    gradient(self, x, t):
```

common/multi_layer_net_extend.py

```
class MultiLayerNetExtend:
    __init__(self, input_size, hidden_size_list, output_size,
             activation='relu', weight_init_std='relu', weight_decay_lambda=0,
             use_dropout = False, dropout_ration = 0.5, use_batchnorm=False):
    __init_weight(self, weight_init_std):
    predict(self, x, train_flg=False):
    loss(self, x, t, train_flg=False):
    accuracy(self, X, T):
    numerical_gradient(self, X, T):
    gradient(self, x, t):
```

optimizer.py & trainer.py

common/optimizer.py

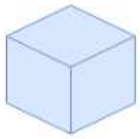
```
class SGD:
    __init__(self, lr=0.01):
    update(self, params, grads):

class Momentum:
    __init__(self, lr=0.01, momentum=0.9):
    update(self, params, grads):

class Nesterov:
    __init__(self, lr=0.01, momentum=0.9):
    update(self, params, grads):

class AdaGrad:
    __init__(self, lr=0.01):
    update(self, params, grads):

class RMSprop:
    __init__(self, lr=0.01, decay_rate = 0.99):
    update(self, params, grads):

class Adam:
    __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
    update(self, params, grads):
```

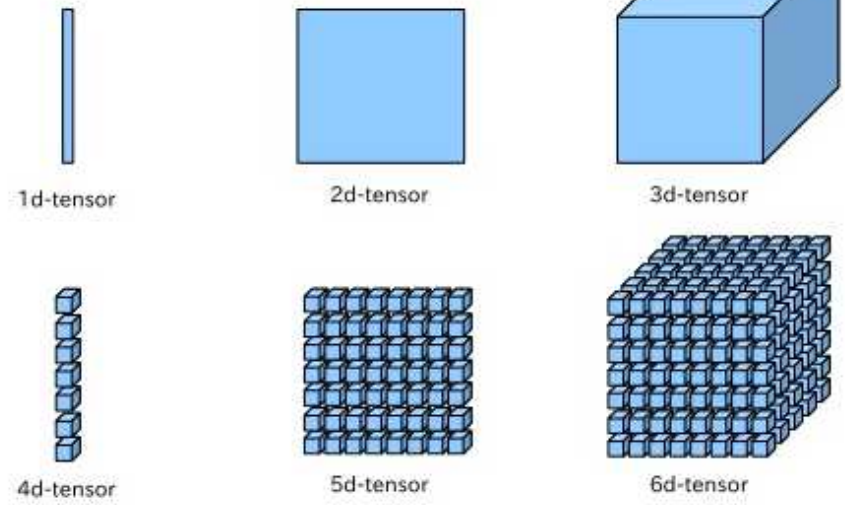
common/trainer.py

```
class Trainer:
    def __init__(self, network, x_train, t_train, x_test, t_test,
                 epochs=20, mini_batch_size=100,
                 optimizer='SGD', optimizer_param={'lr':0.01},
                 evaluate_sample_num_per_epoch=None, verbose=True):
    def train_step(self):
    def train(self):
```

Scalar, Vector, Matrix, Tensor

Dimension
Shape
Type

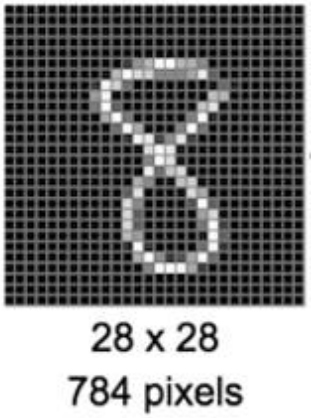
Type	Scalar	Vector	Matrix	Tensor
Definition	a single number	an array of numbers	2-D array of numbers	k-D array of numbers
Notation	x	$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$	$\boldsymbol{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,n} \\ X_{2,1} & X_{2,2} & \dots & X_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ X_{m,1} & X_{m,2} & \dots & X_{m,n} \end{bmatrix}$	$\mathbf{X}$ $X_{i,j,k}$
Example	1.333	$\boldsymbol{x} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 9 \end{bmatrix}$	$\boldsymbol{X} = \begin{bmatrix} 1 & 2 & \dots & 4 \\ 5 & 6 & \dots & 8 \\ \vdots & \vdots & \vdots & \vdots \\ 13 & 14 & \dots & 16 \end{bmatrix}$	$x = \begin{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} & \begin{bmatrix} 10 & 20 & 30 \\ 50 & 60 & 80 \\ 80 & 90 & \end{bmatrix} \\ \begin{bmatrix} 100 & 200 & 300 \\ 00 & 600 & \\ 00 & 900 & \end{bmatrix} & \end{bmatrix}$
Python code example	x = np.array(1.333)	x = np.array([1,2,3, 4,5,6, 7,8,9])	x = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12], [13,14,15,16]])	x = np.array([[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[10, 20, 30], [40, 50, 60], [70, 80, 90]], [[100, 200, 300], [400, 500, 600], [700, 800, 900]]])
Visualization				 3-D Tensor

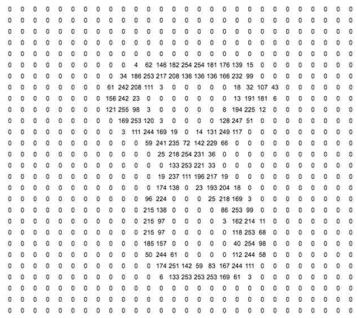


https://medium.datadriveninvestor.com/from-scalar-to-tensor-fundamental-mathematics-for-machine-learning-with-intuitive-examples-part-163727dfea8d

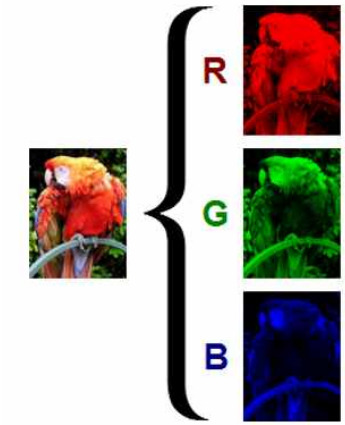
이미지와 텐서

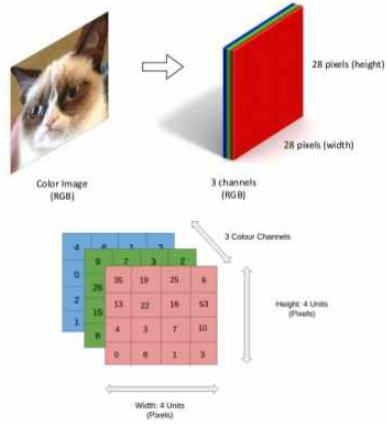
흑백 이미지 : 행렬
흑백 이미지들의 데이터 : **3**차원 텐서





컬러 이미지 : **3**차원 텐서
컬러 이미지들의 데이터 : **4**차원 텐서





/common/functions.py

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

def identity_function(x):
    return x

def step_function(x):
    return np.array(x > 0, dtype=np.int)

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

def sigmoid_grad(x):
    return (1.0 - sigmoid(x)) * sigmoid(x)

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

def relu_grad(x):
    grad = np.zeros(x)
    grad[x>=0] = 1
    return grad
```

```python
def softmax(x):
    if x.ndim == 2:
        x = x.T
        x = x - np.max(x, axis=0)
        y = np.exp(x) / np.sum(np.exp(x), axis=0)
        return y.T

    x = x - np.max(x) # 오버플로 대책
    return np.exp(x) / np.sum(np.exp(x))

def mean_squared_error(y, t):
    return 0.5 * np.sum((y-t)**2)


def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    # 훈련 데이터가 원-핫 벡터라면 정답 레이블의 인덱스로 반환
    if t.size == y.size:
        t = t.argmax(axis=1)

    batch_size = y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size


def softmax_loss(X, t):
    y = softmax(X)
    return cross_entropy_error(y, t)
```

/common/gradient.py

```python
def _numerical_gradient_1d(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    for idx in range(x.size):
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)

        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2*h)

        x[idx] = tmp_val # 값 복원

    return grad


def numerical_gradient_2d(f, X):
    if X.ndim == 1:
        return _numerical_gradient_1d(f, X)
    else:
        grad = np.zeros_like(X)

        for idx, x in enumerate(X):
            grad[idx] = _numerical_gradient_1d(f, x)

        return grad
```

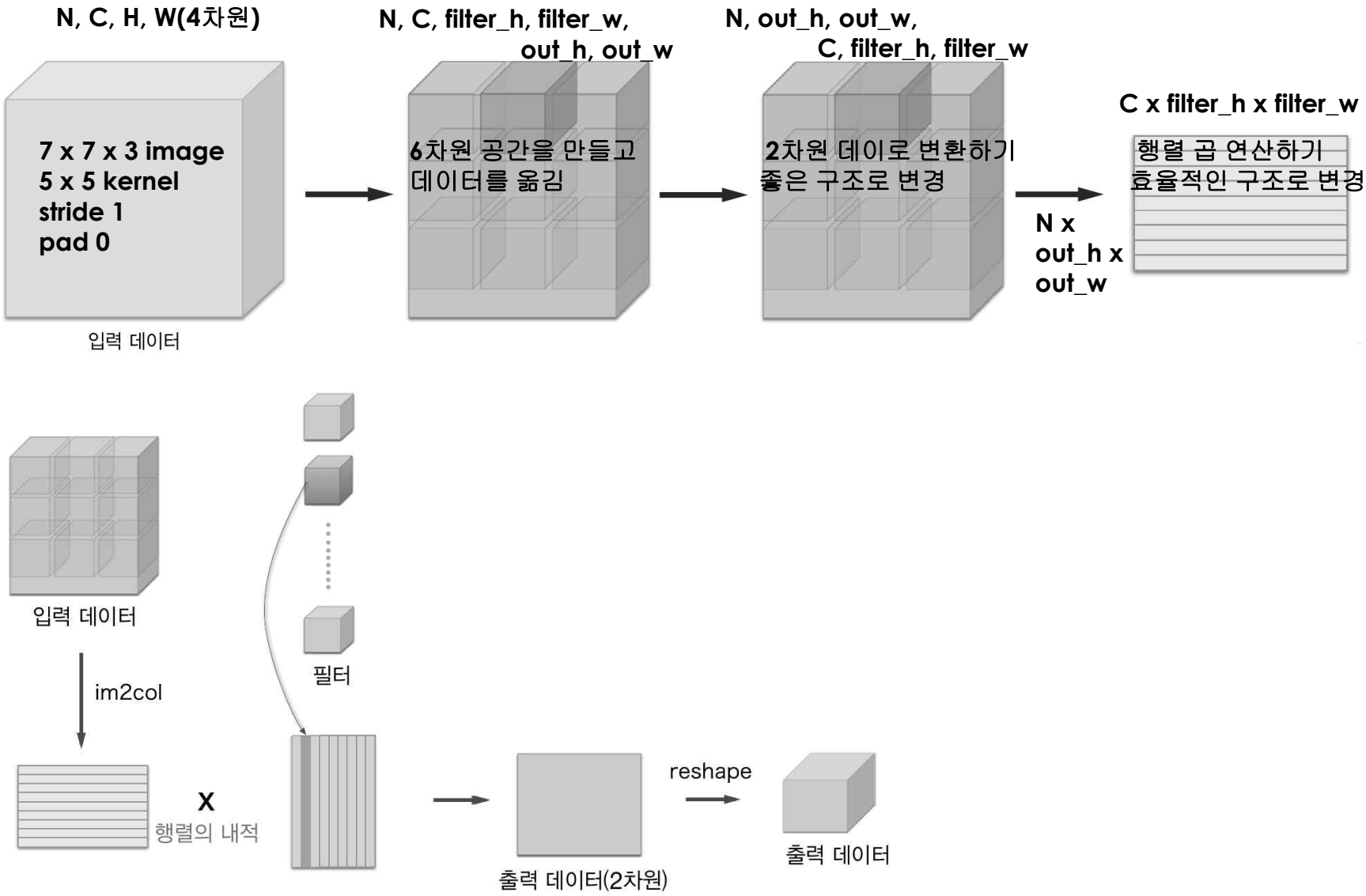
수치연산은 실제는 사용하지 않음

```python
def numerical_gradient(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)

        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2*h)

        x[idx] = tmp_val # 값 복원
        it.iternext()

    return grad
```

/common/util.py

```
5   def smooth_curve(x):
6       """손실 함수의 그래프를 매끄럽게 하기 위해 사용
7
8       참고 : http://glowingpython.blogspot.jp/2012/02/convolution-with-numpy.html
9       """
10      window_len = 11
11      s = np.r_[x[window_len-1:0:-1], x, x[-1:-window_len:-1]]
12      w = np.kaiser(window_len, 2)
13      y = np.convolve(w/w.sum(), s, mode='valid')
14      return y[5:len(y)-5]
17  def shuffle_dataset(x, t):
18      """데이터셋을 뒤섞는다.
19
20      Parameters
21      ----------
22      x : 훈련 데이터
23      t : 정답 레이블
24
25      Returns
26      -------
27      x, t : 뒤섞은 훈련 데이터와 정답 레이블
28      """
29      permutation = np.random.permutation(x.shape[0])
30      x = x[permutation,:] if x.ndim == 2 else x[permutation,:,:,:]
31      t = t[permutation]
32
33      return x, t
35  def conv_output_size(input_size, filter_size, stride=1, pad=0):
36      return (input_size + 2*pad - filter_size) / stride + 1
```

im2col 함수


N, C, H, W(4차원)
7 x 7 x 3 image
5 x 5 kernel
stride 1
pad 0
입력 데이터
N, C, filter_h, filter_w,
out_h, out_w
6차원 공간을 만들고
데이터를 옮김
N, out_h, out_w,
C, filter_h, filter_w
2차원 데이로 변환하기
좋은 구조로 변경
N x
out_h x
out_w
C x filter_h x filter_w
행렬 곱 연산하기
효율적인 구조로 변경
입력 데이터
im2col
필터
X
행렬의 내적
출력 데이터(2차원)
reshape
출력 데이터

/common/util.py

```
39  def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
40      """다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화).
41
42      Parameters
43      ----------
44      input_data : 4차원 배열 형태의 입력 데이터(이미지 수, 채널 수, 높이, 너비)
45      filter_h : 필터의 높이
46      filter_w : 필터의 너비
47      stride : 스트라이드
48      pad : 패딩
49
50      Returns
51      -------
52      col : 2차원 배열
53      """
54      N, C, H, W = input_data.shape
55      out_h = (H + 2*pad - filter_h)//stride + 1
56      out_w = (W + 2*pad - filter_w)//stride + 1
57
58      img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'constant')
59      col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))
60
61      for y in range(filter_h):
62          y_max = y + stride*out_h
63          for x in range(filter_w):
64              x_max = x + stride*out_w
65              col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]
66
67      col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N*out_h*out_w, -1)
68      return col
```

for문 최대한 사용하지 않는 방법으로 구현 - **for**문 느려지기 때문**(**성능이 떨어진다**.)**

입력 데이터 **(7,7,3)**
필터 **(5,5,3)**
스트라이드 **1**
패딩 **0**

출력의 크기 계산

PAD 처리

6차원 **tensor** 만듬

6차원 **tensor**에 데이터 맵핑

2차원 **tensor**로 변환

/common/util.py

```python
def col2im(col, input_shape, filter_h, filter_w, stride=1, pad=0):
    """(im2col과 반대) 2차원 배열을 입력받아 다수의 이미지 묶음으로 변환한다.

    Parameters
    ----------
    col : 2차원 배열(입력 데이터)
    input_shape : 원래 이미지 데이터의 형상 (예 : (10, 1, 28, 28))
    filter_h : 필터의 높이
    filter_w : 필터의 너비
    stride : 스트라이드
    pad : 패딩

    Returns
    -------
    img : 변환된 이미지들
    """
    N, C, H, W = input_shape
    out_h = (H + 2*pad - filter_h)//stride + 1
    out_w = (W + 2*pad - filter_w)//stride + 1
    col = col.reshape(N, out_h, out_w, C, filter_h, filter_w).transpose(0, 3, 4, 5, 1, 2)

    img = np.zeros((N, C, H + 2*pad + stride - 1, W + 2*pad + stride - 1))
    for y in range(filter_h):
        y_max = y + stride*out_h
        for x in range(filter_w):
            x_max = x + stride*out_w
            img[:, :, y:y_max:stride, x:x_max:stride] += col[:, :, y, x, :, :]

    return img[:, :, pad:H + pad, pad:W + pad]
```

/common/layers.py

```python
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0
        out = x.copy()
        out[self.mask] = 0

        return out

    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx

class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = sigmoid(x)
        self.out = out
        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx
```

```python
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b

        self.x = None
        self.original_x_shape = None
        # 가중치와 편향 매개변수의 미분
        self.dW = None
        self.db = None

    def forward(self, x):
        # 텐서 대응
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out
```

/common/layers.py

```python
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None # 손실함수
        self.y = None    # softmax의 출력
        self.t = None    # 정답 레이블(원-핫 인코딩 형태)

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)

        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        if self.t.size == self.y.size: # 정답 레이블이 원-핫 인코딩 형태일 때
            dx = (self.y - self.t) / batch_size
        else:
            dx = self.y.copy()
            dx[np.arange(batch_size), self.t] -= 1
            dx = dx / batch_size

        return dx
```

```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
```

/common/layers.py

```python
class BatchNormalization:
    """
    http://arxiv.org/abs/1502.03167
    """
    def __init__(self, gamma, beta, momentum=0.9, running_mean=None, running_var=None):
        self.gamma = gamma
        self.beta = beta
        self.momentum = momentum
        self.input_shape = None

        # 시험할 때 사용할 평균과 분산
        self.running_mean = running_mean
        self.running_var = running_var

        # backward 시에 사용할 중간 데이터
        self.batch_size = None
        self.xc = None
        self.std = None
        self.dgamma = None
        self.dbeta = None

    def forward(self, x, train_flg=True):
        self.input_shape = x.shape
        if x.ndim != 2:
            N, C, H, W = x.shape
            x = x.reshape(N, -1)

        out = self.__forward(x, train_flg)

        return out.reshape(*self.input_shape)

    def __forward(self, x, train_flg):
        if self.running_mean is None:
            N, D = x.shape
            self.running_mean = np.zeros(D)
            self.running_var = np.zeros(D)

        if train_flg:
            mu = x.mean(axis=0)
            xc = x - mu
            var = np.mean(xc**2, axis=0)
            std = np.sqrt(var + 10e-7)
            xn = xc / std

            self.batch_size = x.shape[0]
            self.xc = xc
            self.xn = xn
            self.std = std
            self.running_mean = self.momentum * self.running_mean + (1-self.momentum) * mu
            self.running_var = self.momentum * self.running_var + (1-self.momentum) * var
        else:
            xc = x - self.running_mean
            xn = xc / ((np.sqrt(self.running_var + 10e-7)))

        out = self.gamma * xn + self.beta
        return out
```

/common/layers.py

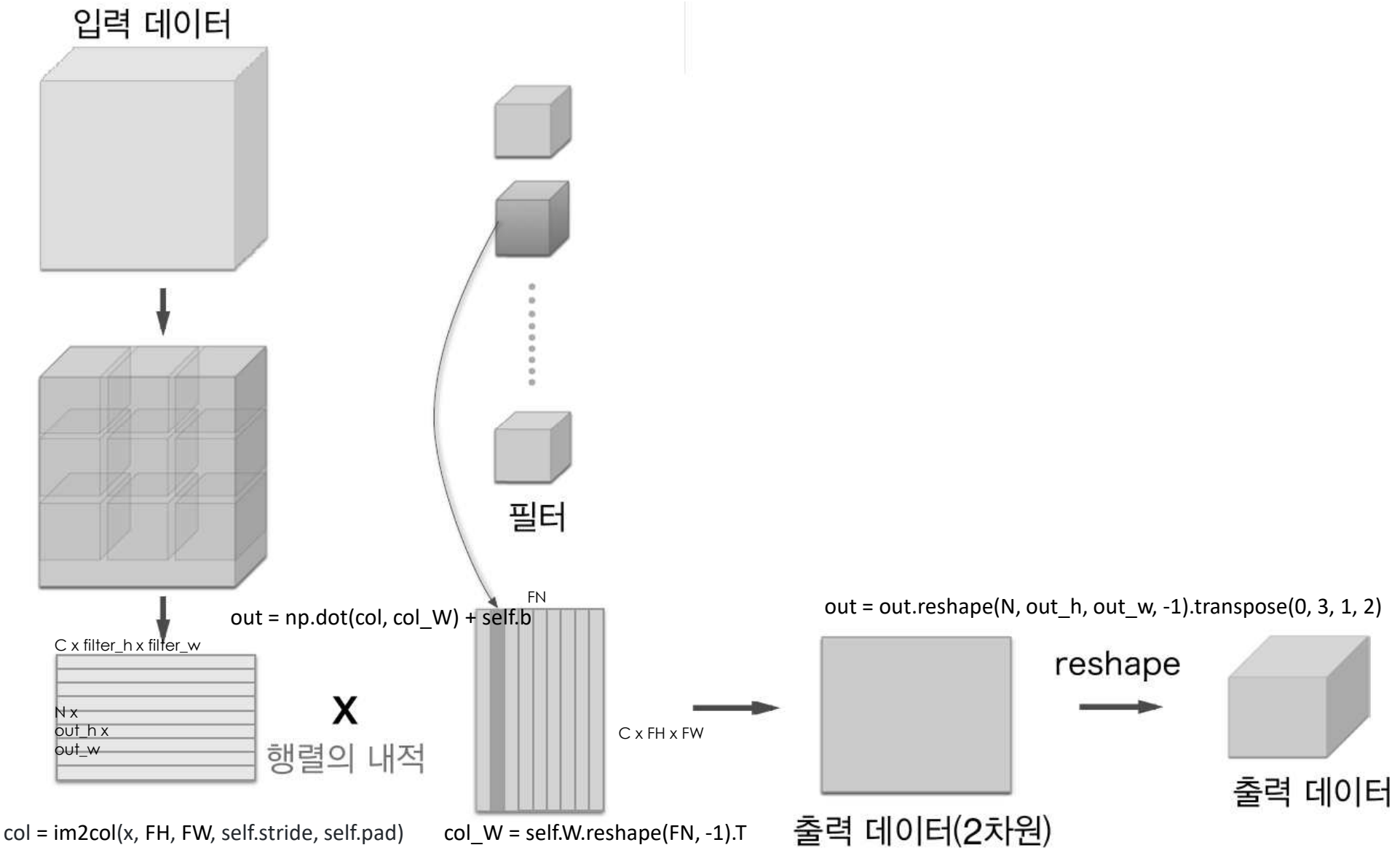
```
171     def backward(self, dout):
172         if dout.ndim != 2:
173             N, C, H, W = dout.shape
174             dout = dout.reshape(N, -1)
175 
176         dx = self.__backward(dout)
177 
178         dx = dx.reshape(*self.input_shape)
179         return dx
180 
181     def __backward(self, dout):
182         dbeta = dout.sum(axis=0)
183         dgamma = np.sum(self.xn * dout, axis=0)
184         dxn = self.gamma * dout
185         dxc = dxn / self.std
186         dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
187         dvar = 0.5 * dstd / self.std
188         dxc += (2.0 / self.batch_size) * self.xc * dvar
189         dmu = np.sum(dxc, axis=0)
190         dx = dxc - dmu / self.batch_size
191 
192         self.dgamma = dgamma
193         self.dbeta = dbeta
194 
195         return dx
```

/common/layers.py

```python
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

        # 중간 데이터 (backward 시 사용)
        self.x = None
        self.col = None
        self.col_W = None

        # 가중치와 편향 매개변수의 기울기
        self.dW = None
        self.db = None
```

```python
    def forward(self, x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
        out_w = 1 + int((W + 2*self.pad - FW) / self.stride)

        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T

        out = np.dot(col, col_W) + self.b
        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        self.x = x
        self.col = col
        self.col_W = col_W

        return out

    def backward(self, dout):
        FN, C, FH, FW = self.W.shape
        dout = dout.transpose(0,2,3,1).reshape(-1, FN)

        self.db = np.sum(dout, axis=0)
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)

        dcol = np.dot(dout, self.col_W.T)
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)

        return dx
```

Convolution Class



/common/layers.py

```python
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

        self.x = None
        self.arg_max = None

    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)

        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)

        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        self.x = x
        self.arg_max = arg_max

        return out

    def backward(self, dout):
        dout = dout.transpose(0, 2, 3, 1)

        pool_size = self.pool_h * self.pool_w
        dmax = np.zeros((dout.size, pool_size))
        dmax[np.arange(self.arg_max.size), self.arg_max.flatten()] = dout.flatten()
        dmax = dmax.reshape(dout.shape + (pool_size,))

        dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
        dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride, self.pad)

        return dx
```

/common/multi_layer_net.py

```python
class MultiLayerNet:
    """완전연결 다층 신경망

    Parameters
    ----------
    input_size : 입력 크기 (MNIST의 경우엔 784)
    hidden_size_list : 각 은닉층의 뉴런 수를 담은 리스트 (e.g. [100, 100, 100])
    output_size : 출력 크기 (MNIST의 경우엔 10)
    activation : 활성화 함수 - 'relu' 혹은 'sigmoid'
    weight_init_std : 가중치의 표준편차 지정 (e.g. 0.01)
        'relu'나 'he'로 지정하면 'He 초깃값'으로 설정
        'sigmoid'나 'xavier'로 지정하면 'Xavier 초깃값'으로 설정
    weight_decay_lambda : 가중치 감소(L2 법칙)의 세기
    """
    def __init__(self, input_size, hidden_size_list, output_size,
                 activation='relu', weight_init_std='relu', weight_decay_lambda=0):
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size_list = hidden_size_list
        self.hidden_layer_num = len(hidden_size_list)
        self.weight_decay_lambda = weight_decay_lambda
        self.params = {}

        # 가중치 초기화
        self.__init_weight(weight_init_std)

        # 계층 생성
        activation_layer = {'sigmoid': Sigmoid, 'relu': Relu}
        self.layers = OrderedDict()
        for idx in range(1, self.hidden_layer_num+1):
            self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
                                                      self.params['b' + str(idx)])
            self.layers['Activation_function' + str(idx)] = activation_layer[activation]()

        idx = self.hidden_layer_num + 1
        self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
            self.params['b' + str(idx)])

        self.last_layer = SoftmaxWithLoss()
```

/common/multi_layer_net.py

```
50      def __init_weight(self, weight_init_std):
51          """가중치 초기화
52
53          Parameters
54          ----------
55          weight_init_std : 가중치의 표준편차 지정 (e.g. 0.01)
56              'relu'나 'he'로 지정하면 'He 초깃값'으로 설정
57              'sigmoid'나 'xavier'로 지정하면 'Xavier 초깃값'으로 설정
58          """
59          all_size_list = [self.input_size] + self.hidden_size_list + [self.output_size]
60          for idx in range(1, len(all_size_list)):
61              scale = weight_init_std
62              if str(weight_init_std).lower() in ('relu', 'he'):
63                  scale = np.sqrt(2.0 / all_size_list[idx - 1])  # ReLU를 사용할 때의 권장 초깃값
64              elif str(weight_init_std).lower() in ('sigmoid', 'xavier'):
65                  scale = np.sqrt(1.0 / all_size_list[idx - 1])  # sigmoid를 사용할 때의 권장 초깃값
66              self.params['W' + str(idx)] = scale * np.random.randn(all_size_list[idx-1], all_size_list[idx])
67              self.params['b' + str(idx)] = np.zeros(all_size_list[idx])
68
69      def predict(self, x):
70          for layer in self.layers.values():
71              x = layer.forward(x)
72
73          return x
```

/common/multi_layer_net.py

```
75      def loss(self, x, t):
76          """손실 함수를 구한다.
77
78          Parameters
79          ----------
80          x : 입력 데이터
81          t : 정답 레이블
82
83          Returns
84          -------
85          손실 함수의 값
86          """
87          y = self.predict(x)
88
89          weight_decay = 0
90          for idx in range(1, self.hidden_layer_num + 2):
91              W = self.params['W' + str(idx)]
92              weight_decay += 0.5 * self.weight_decay_lambda * np.sum(W ** 2)
93
94          return self.last_layer.forward(y, t) + weight_decay
95
96      def accuracy(self, x, t):
97          y = self.predict(x)
98          y = np.argmax(y, axis=1)
99          if t.ndim != 1 : t = np.argmax(t, axis=1)
100
101         accuracy = np.sum(y == t) / float(x.shape[0])
102         return accuracy
```

/common/multi_layer_net.py

```
104     def numerical_gradient(self, x, t):
105         """기울기를 구한다(수치 미분).
106 
107         Parameters
108         ----------
109         x : 입력 데이터
110         t : 정답 레이블
111 
112         Returns
113         -------
114         각 층의 기울기를 담은 딕셔너리(dictionary) 변수
115             grads['W1']、grads['W2']、... 각 층의 가중치
116             grads['b1']、grads['b2']、... 각 층의 편향
117         """
118         loss_W = lambda W: self.loss(x, t)
119 
120         grads = {}
121         for idx in range(1, self.hidden_layer_num+2):
122             grads['W' + str(idx)] = numerical_gradient(loss_W, self.params['W' + str(idx)])
123             grads['b' + str(idx)] = numerical_gradient(loss_W, self.params['b' + str(idx)])
124 
125         return grads
```

/common/multi_layer_net.py

```
127     def gradient(self, x, t):
128         """기울기를 구한다(오차역전파법).
130         Parameters
131         ----------
132         x : 입력 데이터
133         t : 정답 레이블
135         Returns
136         -------
137         각 층의 기울기를 담은 딕셔너리(dictionary) 변수
138             grads['W1']、grads['W2']、... 각 층의 가중치
139             grads['b1']、grads['b2']、... 각 층의 편향
140         """
141         # forward
142         self.loss(x, t)
143 
144         # backward
145         dout = 1
146         dout = self.last_layer.backward(dout)
147 
148         layers = list(self.layers.values())
149         layers.reverse()
150         for layer in layers:
151             dout = layer.backward(dout)
152 
153         # 결과 저장
154         grads = {}
155         for idx in range(1, self.hidden_layer_num+2):
156             grads['W' + str(idx)] = self.layers['Affine' + str(idx)].dW + self.weight_decay_lambda * self.layers['Affine' + str(idx)].W
157             grads['b' + str(idx)] = self.layers['Affine' + str(idx)].db
159         return grads
```

/common/multi_layer_net_extend.py

```python
# coding: utf-8
import sys, os
sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from collections import OrderedDict
from common.layers import *
from common.gradient import numerical_gradient

class MultiLayerNetExtend:
    """완전 연결 다층 신경망(확장판)
    가중치 감소, 드롭아웃, 배치 정규화 구현

    Parameters
    ----------
    input_size : 입력 크기 (MNIST의 경우엔 784)
    hidden_size_list : 각 은닉층의 뉴런 수를 담은 리스트 (e.g. [100, 100, 100])
    output_size : 출력 크기 (MNIST의 경우엔 10)
    activation : 활성화 함수 - 'relu' 혹은 'sigmoid'
    weight_init_std : 가중치의 표준편차 지정 (e.g. 0.01)
        'relu'나 'he'로 지정하면 'He 초깃값'으로 설정
        'sigmoid'나 'xavier'로 지정하면 'Xavier 초깃값'으로 설정
    weight_decay_lambda : 가중치 감소(L2 법칙)의 세기
    use_dropout : 드롭아웃 사용 여부
    dropout_ration : 드롭아웃 비율
    use_batchNorm : 배치 정규화 사용 여부
    """
    def __init__(self, input_size, hidden_size_list, output_size,
                 activation='relu', weight_init_std='relu', weight_decay_lambda=0,
                 use_dropout = False, dropout_ration = 0.5, use_batchnorm=False):
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size_list = hidden_size_list
        self.hidden_layer_num = len(hidden_size_list)
        self.use_dropout = use_dropout
        self.weight_decay_lambda = weight_decay_lambda
        self.use_batchnorm = use_batchnorm
        self.params = {}

        # 가중치 초기화
        self.__init_weight(weight_init_std)
```

/common/multi_layer_net_extend.py

```
42          # 계층 생성
43          activation_layer = {'sigmoid': Sigmoid, 'relu': Relu}
44          self.layers = OrderedDict()
45          for idx in range(1, self.hidden_layer_num+1):
46              self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
47                                                        self.params['b' + str(idx)])
48              if self.use_batchnorm:
49                  self.params['gamma' + str(idx)] = np.ones(hidden_size_list[idx-1])
50                  self.params['beta' + str(idx)] = np.zeros(hidden_size_list[idx-1])
51                  self.layers['BatchNorm' + str(idx)] = BatchNormalization(self.params['gamma' + str(idx)], self.params['beta' + str(idx)])
52
53              self.layers['Activation_function' + str(idx)] = activation_layer[activation]()
54
55              if self.use_dropout:
56                  self.layers['Dropout' + str(idx)] = Dropout(dropout_ration)
57
58          idx = self.hidden_layer_num + 1
59          self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)], self.params['b' + str(idx)])
60
61          self.last_layer = SoftmaxWithLoss()
```

/common/multi_layer_net_extend.py

```
63      def __init_weight(self, weight_init_std):
64          """가중치 초기화
65
66          Parameters
67          ----------
68          weight_init_std : 가중치의 표준편차 지정 (e.g. 0.01)
69              'relu'나 'he'로 지정하면 'He 초깃값'으로 설정
70              'sigmoid'나 'xavier'로 지정하면 'Xavier 초깃값'으로 설정
71          """
72          all_size_list = [self.input_size] + self.hidden_size_list + [self.output_size]
73          for idx in range(1, len(all_size_list)):
74              scale = weight_init_std
75              if str(weight_init_std).lower() in ('relu', 'he'):
76                  scale = np.sqrt(2.0 / all_size_list[idx - 1])  # ReLUを使う場合に推奨される初期値
77              elif str(weight_init_std).lower() in ('sigmoid', 'xavier'):
78                  scale = np.sqrt(1.0 / all_size_list[idx - 1])  # sigmoidを使う場合に推奨される初期値
79              self.params['W' + str(idx)] = scale * np.random.randn(all_size_list[idx-1], all_size_list[idx])
80              self.params['b' + str(idx)] = np.zeros(all_size_list[idx])
```

/common/multi_layer_net_extend.py

```
82      def predict(self, x, train_flg=False):
83          for key, layer in self.layers.items():
84              if "Dropout" in key or "BatchNorm" in key:
85                  x = layer.forward(x, train_flg)
86              else:
87                  x = layer.forward(x)
88
89          return x



91      def loss(self, x, t, train_flg=False):
92          """손실 함수를 구한다.
93
94          Parameters
95          ----------
96          x : 입력 데이터
97          t : 정답 레이블
98          """
99          y = self.predict(x, train_flg)
100
101         weight_decay = 0
102         for idx in range(1, self.hidden_layer_num + 2):
103             W = self.params['W' + str(idx)]
104             weight_decay += 0.5 * self.weight_decay_lambda * np.sum(W**2)
105
106         return self.last_layer.forward(y, t) + weight_decay
```

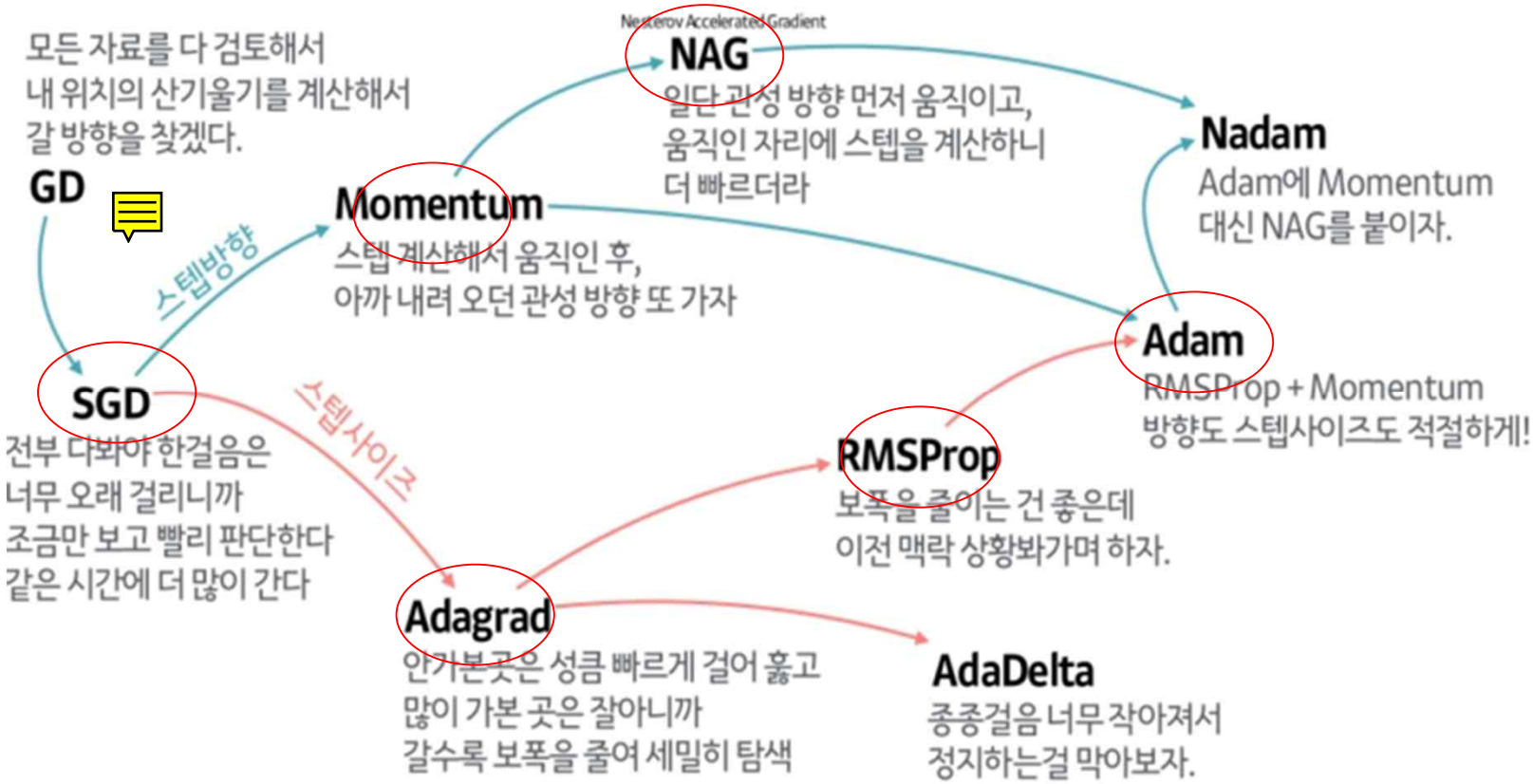
/common/multi_layer_net_extend.py

```
108     def accuracy(self, X, T):
109         Y = self.predict(X, train_flg=False)
110         Y = np.argmax(Y, axis=1)
111         if T.ndim != 1 : T = np.argmax(T, axis=1)
112 
113         accuracy = np.sum(Y == T) / float(X.shape[0])
114         return accuracy
116     def numerical_gradient(self, X, T):
117         """기울기를 구한다(수치 미분).
118 
119         Parameters
120         ----------
121         x : 입력 데이터
122         t : 정답 레이블
123 
124         Returns
125         -------
126         각 층의 기울기를 담은 사전(dictionary) 변수
127             grads['W1']、grads['W2']、... 각 층의 가중치
128             grads['b1']、grads['b2']、... 각 층의 편향
129         """
130         loss_W = lambda W: self.loss(X, T, train_flg=True)
131 
132         grads = {}
133         for idx in range(1, self.hidden_layer_num+2):
134             grads['W' + str(idx)] = numerical_gradient(loss_W, self.params['W' + str(idx)])
135             grads['b' + str(idx)] = numerical_gradient(loss_W, self.params['b' + str(idx)])
136 
137             if self.use_batchnorm and idx != self.hidden_layer_num+1:
138                 grads['gamma' + str(idx)] = numerical_gradient(loss_W, self.params['gamma' + str(idx)])
139                 grads['beta' + str(idx)] = numerical_gradient(loss_W, self.params['beta' + str(idx)])
140 
141         return grads
```

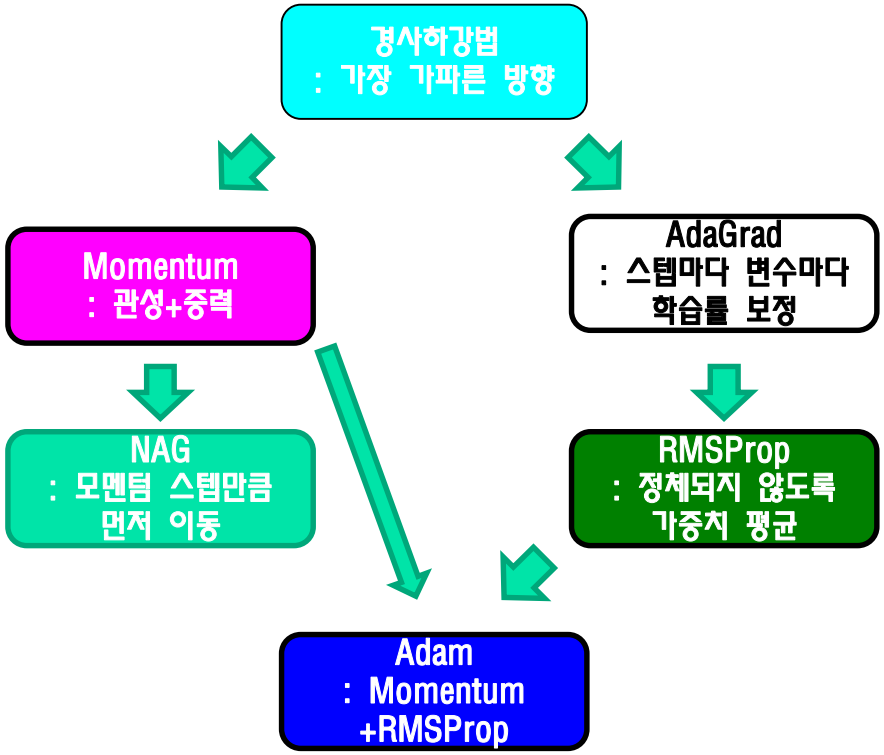
/common/multi_layer_net_extend.py

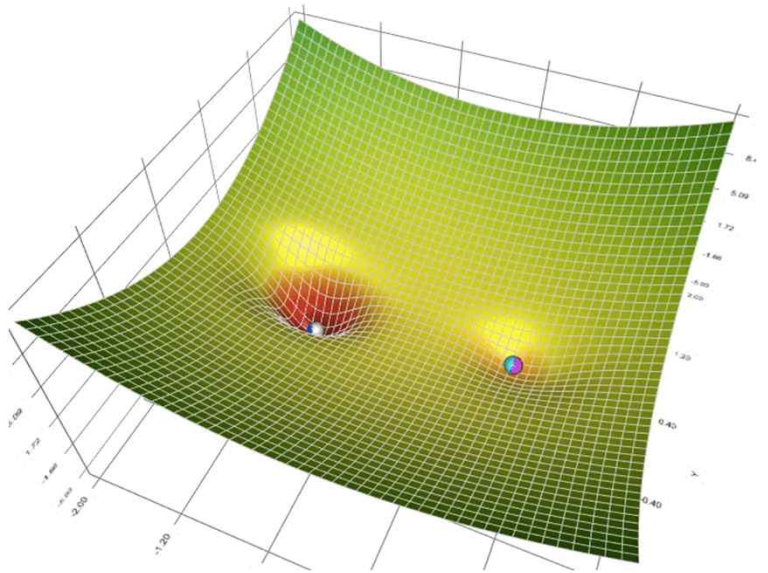
```
143     def gradient(self, x, t):
144         # forward
145         self.loss(x, t, train_flg=True)
146 
147         # backward
148         dout = 1
149         dout = self.last_layer.backward(dout)
150 
151         layers = list(self.layers.values())
152         layers.reverse()
153         for layer in layers:
154             dout = layer.backward(dout)
155 
156         # 결과 저장
157         grads = {}
158         for idx in range(1, self.hidden_layer_num+2):
159             grads['W' + str(idx)] = self.layers['Affine' + str(idx)].dW + self.weight_decay_lambda * self.params['W' + str(idx)]
160             grads['b' + str(idx)] = self.layers['Affine' + str(idx)].db
161 
162             if self.use_batchnorm and idx != self.hidden_layer_num+1:
163                 grads['gamma' + str(idx)] = self.layers['BatchNorm' + str(idx)].dgamma
164                 grads['beta' + str(idx)] = self.layers['BatchNorm' + str(idx)].dbeta
165 
166         return grads
```

optimizer


모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
스텝사이즈
Adagrad
안가본곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색
RMSProp
보폭을 줄이는 건 좋은데
이전 맥락 상황봐가며 하자.
AdaDelta
종종걸음 너무 작아져서
정지하는걸 막아보자.
Adam
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!
Nadam
Adam에 Momentum
대신 NAG를 붙이자.

옵티마이저 계보


경사하강법
: 가장 가파른 방향
Momentum
: 관성+증력
AdaGrad
: 스텝마다 변수마다
학습률 보정
NAG
: 모멘텀 스텝만큼
먼저 이동
RMSProp
: 정체되지 않도록
가중치 평균
Adam
: Momentum
+RMSProp



/common/optimizer.py

```python
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]

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]
```

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

스텝 계산해서 움직인 후, 아까 내려 오던 관성 방향 또 가자

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

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

/common/optimizer.py

일단 관성 방향 먼저 움직이고, 움직인 자리에 스텝을 계산하니 더 빠르더라

```python
class Nesterov:

    """Nesterov's Accelerated Gradient (http://arxiv.org/abs/1212.0901)"""
    # NAG는 모멘텀에서 한 단계 발전한 방법이다. (http://newsight.tistory.com/224)

    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.momentum * self.momentum * self.v[key]
            params[key] -= (1 + self.momentum) * self.lr * grads[key]
```

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

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

/common/optimizer.py

안가 본 곳은 성큼 빠르게 걸어 훑고 가본 곳은 잘 아니까 갈 수록 보폭을 줄여 세밀히 탐색

```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)
```

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

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

/common/optimizer.py

보폭을 줄이는 건 좋은데 이전 맥락 상황 봐가며 가자.

```python
class RMSprop:

    """RMSprop"""

    def __init__(self, lr=0.01, decay_rate = 0.99):
        self.lr = lr
        self.decay_rate = decay_rate
        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] *= self.decay_rate
            self.h[key] += (1 - self.decay_rate) * grads[key] * grads[key]
            params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```

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

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

/common/optimizer.py

```
99  class Adam:
103     def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
104         self.lr = lr
105         self.beta1 = beta1
106         self.beta2 = beta2
107         self.iter = 0
108         self.m = None
109         self.v = None
110
111     def update(self, params, grads):
112         if self.m is None:
113             self.m, self.v = {}, {}
114             for key, val in params.items():
115                 self.m[key] = np.zeros_like(val)
116                 self.v[key] = np.zeros_like(val)
118         self.iter += 1
119         lr_t  = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.iter)
120
121         for key in params.keys():
122             #self.m[key] = self.beta1*self.m[key] + (1-self.beta1)*grads[key]
123             #self.v[key] = self.beta2*self.v[key] + (1-self.beta2)*(grads[key]**2)
124             self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
125             self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])
126
127             params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)
128
129             #unbias_m += (1 - self.beta1) * (grads[key] - self.m[key]) # correct bias
130             #unbisa_b += (1 - self.beta2) * (grads[key]*grads[key] - self.v[key]) # correct bias
131             #params[key] += self.lr * unbias_m / (np.sqrt(unbisa_b) + 1e-7)
```

방향도 스텝사이즈도 적절하게

$$\mathbf{m}_n = \beta_1 \mathbf{m}_{n-1} + (1-\beta_1)\nabla f(\mathbf{x}_n), \qquad \mathbf{m}_{-1} = \mathbf{0}$$

$$\mathbf{v}_n = \beta_2 \mathbf{v}_{n-1} + (1-\beta_2)\nabla f(\mathbf{x}_n) \odot \nabla f(\mathbf{x}_n), \qquad \mathbf{v}_{-1} = \mathbf{0}$$

$$\hat{\mathbf{m}}_n = \frac{\mathbf{m}_n}{1-\beta_1^{n+1}}, \qquad \hat{\mathbf{v}}_n = \frac{\mathbf{v}_n}{1-\beta_2^{n+1}}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \frac{1}{\sqrt{\hat{\mathbf{v}}_n}} \odot \hat{\mathbf{m}}_n$$

/common/trainer.py

```python
class Trainer:
    """신경망 훈련을 대신 해주는 클래스
    """
    def __init__(self, network, x_train, t_train, x_test, t_test,
                 epochs=20, mini_batch_size=100,
                 optimizer='SGD', optimizer_param={'lr':0.01},
                 evaluate_sample_num_per_epoch=None, verbose=True):
        self.network = network
        self.verbose = verbose
        self.x_train = x_train
        self.t_train = t_train
        self.x_test = x_test
        self.t_test = t_test
        self.epochs = epochs
        self.batch_size = mini_batch_size
        self.evaluate_sample_num_per_epoch = evaluate_sample_num_per_epoch

        # optimzer
        optimizer_class_dict = {'sgd':SGD, 'momentum':Momentum, 'nesterov':Nesterov,
                                'adagrad':AdaGrad, 'rmsprpo':RMSprop, 'adam':Adam}
        self.optimizer = optimizer_class_dict[optimizer.lower()](**optimizer_param)

        self.train_size = x_train.shape[0]
        self.iter_per_epoch = max(self.train_size / mini_batch_size, 1)
        self.max_iter = int(epochs * self.iter_per_epoch)
        self.current_iter = 0
        self.current_epoch = 0

        self.train_loss_list = []
        self.train_acc_list = []
        self.test_acc_list = []
```

/common/trainer.py

```
39      def train_step(self):
40          batch_mask = np.random.choice(self.train_size, self.batch_size)
41          x_batch = self.x_train[batch_mask]
42          t_batch = self.t_train[batch_mask]
43
44          grads = self.network.gradient(x_batch, t_batch)
45          self.optimizer.update(self.network.params, grads)
46
47          loss = self.network.loss(x_batch, t_batch)
48          self.train_loss_list.append(loss)
49          if self.verbose: print("train loss:" + str(loss))
50
51          if self.current_iter % self.iter_per_epoch == 0:
52              self.current_epoch += 1
53
54              x_train_sample, t_train_sample = self.x_train, self.t_train
55              x_test_sample, t_test_sample = self.x_test, self.t_test
56              if not self.evaluate_sample_num_per_epoch is None:
57                  t = self.evaluate_sample_num_per_epoch
58                  x_train_sample, t_train_sample = self.x_train[:t], self.t_train[:t]
59                  x_test_sample, t_test_sample = self.x_test[:t], self.t_test[:t]
60
61              train_acc = self.network.accuracy(x_train_sample, t_train_sample)
62              test_acc = self.network.accuracy(x_test_sample, t_test_sample)
63              self.train_acc_list.append(train_acc)
64              self.test_acc_list.append(test_acc)
65
66              if self.verbose: print("=== epoch:" + str(self.current_epoch) + ", train acc:" + str(train_acc) + ", test acc:" + str(test_acc) + " ===")
67          self.current_iter += 1
```

/common/trainer.py

```
69      def train(self):
70          for i in range(self.max_iter):
71              self.train_step()
72
73          test_acc = self.network.accuracy(self.x_test, self.t_test)
74
75          if self.verbose:
76              print("=============== Final Test Accuracy ===============")
77              print("test acc:" + str(test_acc))
```