



















13

박 철

과정 개요

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

<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		---	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 각 method 마다 - Affine forward와 backward가 있음 - Relu - Sigmoid - Convolution - Pooling - SoftmaxWithLoss - Dropout - BatchNormalization				
	W b 기존 파일에서 로딩 훈련을 통하여 생성 텐서로 관리됨 Filter/kernel pkl 파일로 관리				

목차

- 1장 헬로 파이썬
- 2장 퍼셉트론
- 3장 신경망
- 4장 신경망 학습
- 5장 오차역전파법
- 6장 학습 관련 기술들
- 7장 합성곱 신경망(CNN)
- 8장 딥러닝

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

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

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):
```



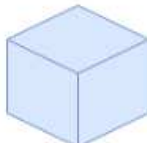
common/trainer.py


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):
```

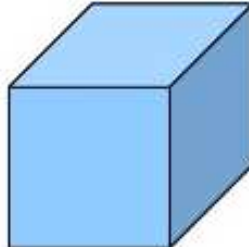

Scalar, Vector, Matrix, Tensor


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	$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$	$\mathbf{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,n} \\ X_{2,1} & X_{2,2} & \dots & X_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m,1} & X_{m,2} & \dots & X_{m,n} \end{bmatrix}$	\mathbf{X} $X_{i,j,k}$
Example	1.333	$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 9 \end{bmatrix}$	$\mathbf{X} = \begin{bmatrix} 1 & 2 & \dots & 4 \\ 5 & 6 & \dots & 8 \\ \vdots & \vdots & \ddots & \vdots \\ 13 & 14 & \dots & 16 \end{bmatrix}$	$\mathbf{x} = \begin{bmatrix} [100, 200, 300] \\ [10, 20, 30] \\ [4, 5, 6] \\ [7, 8, 9] \end{bmatrix}$
Python code example	<code>x = np.array(1.333)</code>	<code>x = np.array([1,2,3,4,5,6,7,8,9])</code>	<code>x = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12], [13,14,15,16]])</code>	<code>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]]]])</code>
Visualization				 3-D Tensor

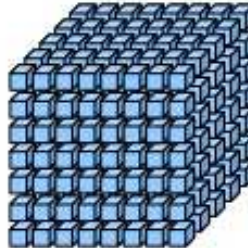

1d-tensor


2d-tensor


3d-tensor


4d-tensor

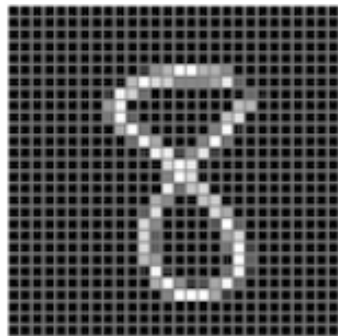

5d-tensor


6d-tensor

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

흑백 이미지 : 행렬

흑백 이미지들의 데이터 : 3차원 텐서

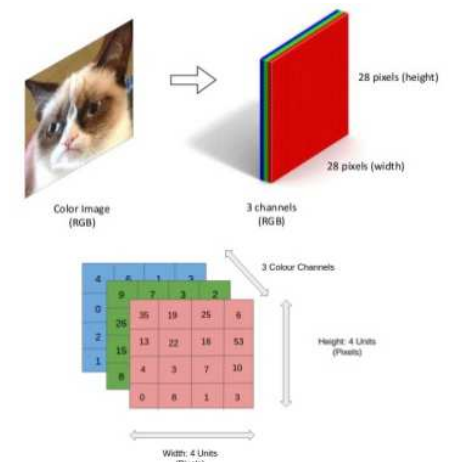
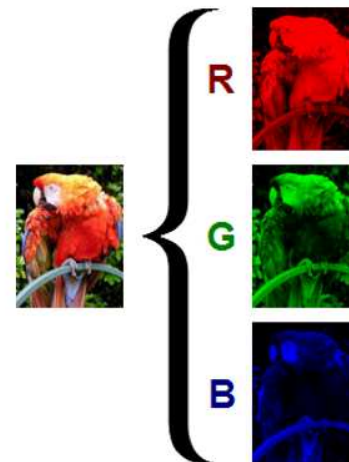


28 x 28
784 pixels

[illegible]

컬러 이미지 : 3차원 텐서

컬러 이미지들의 데이터: 4차원 텐서



/common/functions.py

```
1  # coding: utf-8
2  import numpy as np

5  def identity_function(x):
6      return x

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

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

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

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

25  def relu_grad(x):
26     grad = np.zeros(x)
27     grad[x >= 0] = 1
28     return grad

31  def softmax(x):
32     if x.ndim == 2:
33         x = x.T
34         x = x - np.max(x, axis=0)
35         y = np.exp(x) / np.sum(np.exp(x), axis=0)
36         return y.T
37
38     x = x - np.max(x) # 오버플로 대책
39     return np.exp(x) / np.sum(np.exp(x))

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

46  def cross_entropy_error(y, t):
47     if y.ndim == 1:
48         t = t.reshape(1, t.size)
49         y = y.reshape(1, y.size)
50
51     # 훈련 데이터가 원-핫 벡터라면 정답 레이블의 인덱스로 반환
52     if t.size == y.size:
53         t = t.argmax(axis=1)
54
55     batch_size = y.shape[0]
56     return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size

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

/common/gradient.py

수치연산은 실제로 사용하지 않음

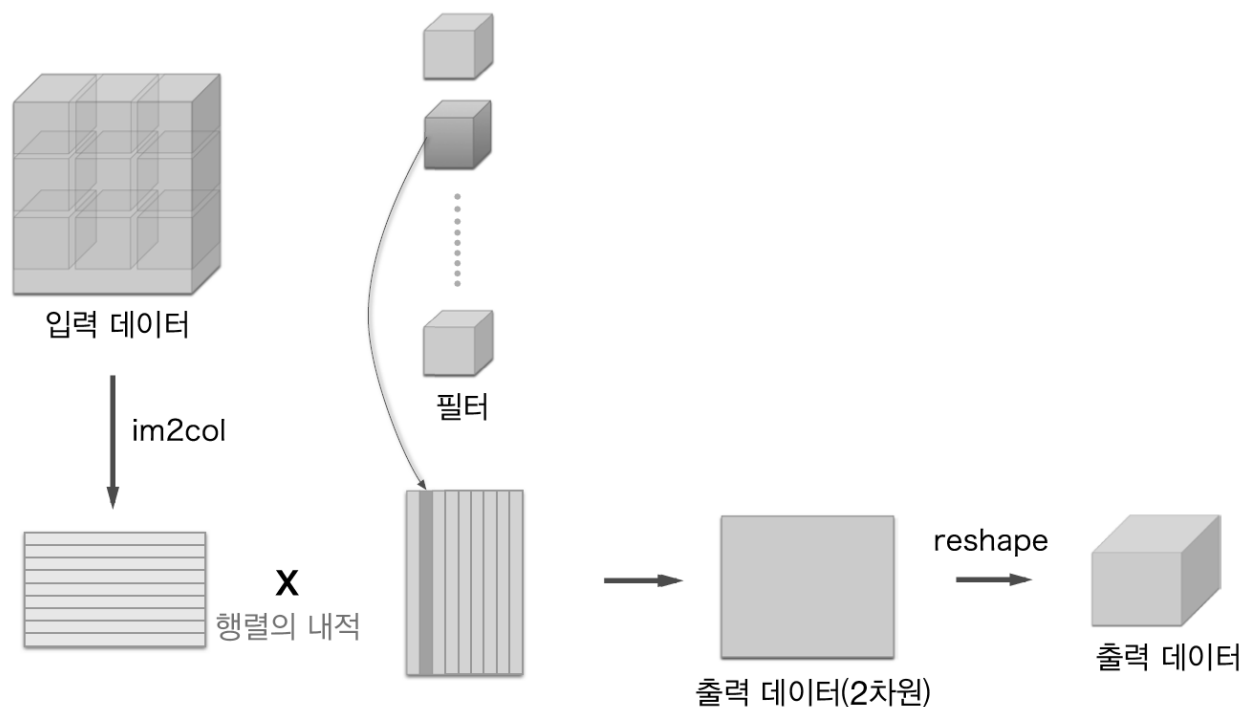
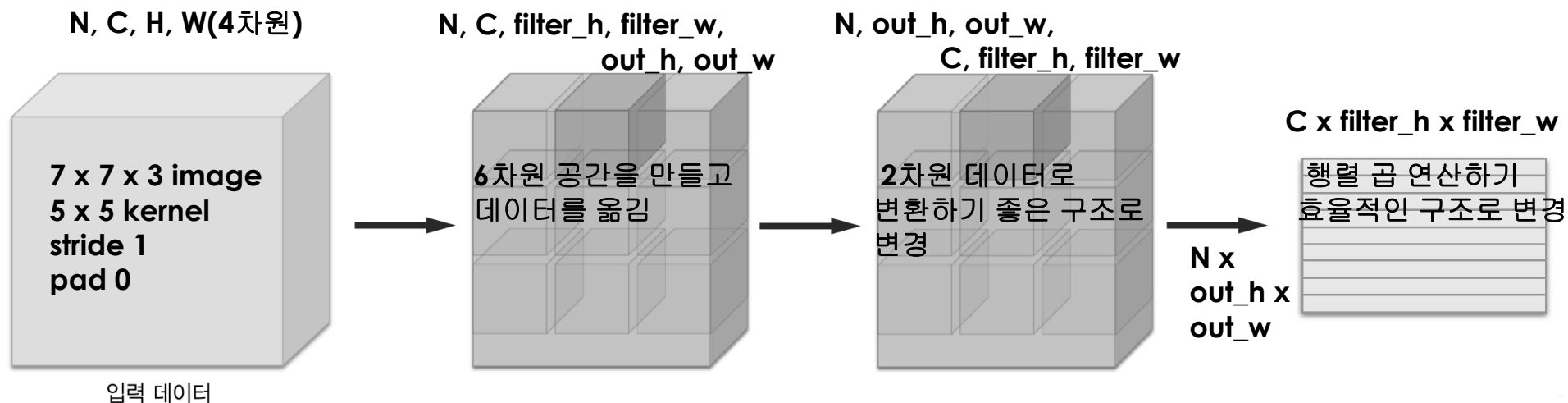
```
4 def _numerical_gradient_1d(f, x):
5     h = 1e-4 # 0.0001
6     grad = np.zeros_like(x)
7
8     for idx in range(x.size):
9         tmp_val = x[idx]
10        x[idx] = float(tmp_val) + h
11        fxh1 = f(x) # f(x+h)
12
13        x[idx] = tmp_val - h
14        fxh2 = f(x) # f(x-h)
15        grad[idx] = (fxh1 - fxh2) / (2*h)
16
17        x[idx] = tmp_val # 값 복원
18
19    return grad
22 def numerical_gradient_2d(f, X):
23     if X.ndim == 1:
24         return _numerical_gradient_1d(f, X)
25     else:
26         grad = np.zeros_like(X)
27
28         for idx, x in enumerate(X):
29             grad[idx] = _numerical_gradient_1d(f, x)
30
31    return grad
```

```
34 def numerical_gradient(f, x):
35     h = 1e-4 # 0.0001
36     grad = np.zeros_like(x)
37
38     it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
39     while not it.finished:
40         idx = it.multi_index
41         tmp_val = x[idx]
42         x[idx] = float(tmp_val) + h
43         fxh1 = f(x) # f(x+h)
44
45         x[idx] = tmp_val - h
46         fxh2 = f(x) # f(x-h)
47         grad[idx] = (fxh1 - fxh2) / (2*h)
48
49         x[idx] = tmp_val # 값 복원
50         it.iternext()
51
52    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



/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 - for문 느려지기 때문(성능이 떨어진다.)

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

출력의 크기 계산

PAD 처리
6차원 tensor 만들

6차원 tensor에 데이터 맵핑

2차원 tensor로 변환

/common/util.py

```
71 def col2im(col, input_shape, filter_h, filter_w, stride=1, pad=0):
72     """(im2col과 반대) 2차원 배열을 입력받아 다수의 이미지 묶음으로 변환한다.
73
74     Parameters
75     -----
76     col : 2차원 배열(입력 데이터)
77     input_shape : 원래 이미지 데이터의 형상 (예 : (10, 1, 28, 28))
78     filter_h : 필터의 높이
79     filter_w : 필터의 너비
80     stride : 스트라이드
81     pad : 패딩
82
83     Returns
84     -----
85     img : 변환된 이미지들
86     """
87     N, C, H, W = input_shape
88     out_h = (H + 2*pad - filter_h)//stride + 1
89     out_w = (W + 2*pad - filter_w)//stride + 1
90     col = col.reshape(N, out_h, out_w, C, filter_h, filter_w).transpose(0, 3, 4, 5, 1, 2)
91
92     img = np.zeros((N, C, H + 2*pad + stride - 1, W + 2*pad + stride - 1))
93     for y in range(filter_h):
94         y_max = y + stride*out_h
95         for x in range(filter_w):
96             x_max = x + stride*out_w
97             img[:, :, y:y_max:stride, x:x_max:stride] += col[:, :, y, x, :, :]
98
99     return img[:, :, pad:H + pad, pad:W + pad]
```


/common/layers.py

```
7 class Relu:
8     def __init__(self):
9         self.mask = None
10
11     def forward(self, x):
12         self.mask = (x <= 0)
13         out = x.copy()
14         out[self.mask] = 0
15
16         return out
17
18     def backward(self, dout):
19         dout[self.mask] = 0
20         dx = dout
21
22         return dx
23
24
25 class Sigmoid:
26     def __init__(self):
27         self.out = None
28
29     def forward(self, x):
30         out = sigmoid(x)
31         self.out = out
32         return out
33
34     def backward(self, dout):
35         dx = dout * (1.0 - self.out) * self.out
36
37         return dx
```

```
40 class Affine:
41     def __init__(self, W, b):
42         self.W = W
43         self.b = b
44
45         self.x = None
46         self.original_x_shape = None
47         # 가중치와 편향 매개변수의 미분
48         self.dW = None
49         self.db = None
50
51     def forward(self, x):
52         # 텐서 대응
53         self.original_x_shape = x.shape
54         x = x.reshape(x.shape[0], -1)
55         self.x = x
56
57         out = np.dot(self.x, self.W) + self.b
58
59         return out
```

/common/layers.py

```
70 class SoftmaxWithLoss:
71     def __init__(self):
72         self.loss = None # 손실함수
73         self.y = None    # softmax의 출력
74         self.t = None    # 정답 레이블 (원-핫 인코딩 형태)
75
76     def forward(self, x, t):
77         self.t = t
78         self.y = softmax(x)
79         self.loss = cross_entropy_error(self.y, self.t)
80
81         return self.loss
82
83     def backward(self, dout=1):
84         batch_size = self.t.shape[0]
85         if self.t.size == self.y.size: # 정답 레이블이 원-핫 인코딩 형태일 때
86             dx = (self.y - self.t) / batch_size
87         else:
88             dx = self.y.copy()
89             dx[np.arange(batch_size), self.t] -= 1
90             dx = dx / batch_size
91
92         return dx
```

```
95 class Dropout:
96     """
97     http://arxiv.org/abs/1207.0580
98     """
99     def __init__(self, dropout_ratio=0.5):
100         self.dropout_ratio = dropout_ratio
101         self.mask = None
102
103     def forward(self, x, train_flg=True):
104         if train_flg:
105             self.mask = np.random.rand(*x.shape) > self.dropout_ratio
106             return x * self.mask
107         else:
108             return x * (1.0 - self.dropout_ratio)
109
110     def backward(self, dout):
111         return dout * self.mask
```

/common/layers.py

```
114 class BatchNormalization:
115     """
116     http://arxiv.org/abs/1502.03167
117     """
118     def __init__(self, gamma, beta, momentum=0.9,
119                 running_mean=None, running_var=None):
120         self.gamma = gamma
121         self.beta = beta
122         self.momentum = momentum
123         self.input_shape = None
124
125         # 시험할 때 사용할 평균과 분산
126         self.running_mean = running_mean
127         self.running_var = running_var
128
129         # backward 시에 사용할 중간 데이터
130         self.batch_size = None
131         self.xc = None
132         self.std = None
133         self.dgamma = None
134         self.dbeta = None
135
136     def forward(self, x, train_flg=True):
137         self.input_shape = x.shape
138         if x.ndim != 2:
139             N, C, H, W = x.shape
140             x = x.reshape(N, -1)
141
142         out = self.__forward(x, train_flg)
143
144         return out.reshape(*self.input_shape)
145
146     def __forward(self, x, train_flg):
147         if self.running_mean is None:
148             N, D = x.shape
149             self.running_mean = np.zeros(D)
150             self.running_var = np.zeros(D)
151
152         if train_flg:
153             mu = x.mean(axis=0)
154             xc = x - mu
155             var = np.mean(xc**2, axis=0)
156             std = np.sqrt(var + 10e-7)
157             xn = xc / std
158
159             self.batch_size = x.shape[0]
160             self.xc = xc
161             self.xn = xn
162             self.std = std
163             self.running_mean = self.momentum * self.running_mean + (1-self.momentum) * mu
164             self.running_var = self.momentum * self.running_var + (1-self.momentum) * var
165         else:
166             xc = x - self.running_mean
167             xn = xc / ((np.sqrt(self.running_var + 10e-7)))
168
169         out = self.gamma * xn + self.beta
170         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

```
198 class Convolution:
199     def __init__(self, W, b, stride=1, pad=0):
200         self.W = W
201         self.b = b
202         self.stride = stride
203         self.pad = pad
204
205         # 중간 데이터 (backward 시 사용)
206         self.x = None
207         self.col = None
208         self.col_W = None
209
210         # 가중치와 편향 매개변수의 기울기
211         self.dW = None
212         self.db = None
```

```
214     def forward(self, x):
215         FN, C, FH, FW = self.W.shape
216         N, C, H, W = x.shape
217         out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
218         out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
219
220         col = im2col(x, FH, FW, self.stride, self.pad)
221         col_W = self.W.reshape(FN, -1).T
222
223         out = np.dot(col, col_W) + self.b
224         out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
225
226         self.x = x
227         self.col = col
228         self.col_W = col_W
229
230         return out
231
232     def backward(self, dout):
233         FN, C, FH, FW = self.W.shape
234         dout = dout.transpose(0,2,3,1).reshape(-1, FN)
235
236         self.db = np.sum(dout, axis=0)
237         self.dW = np.dot(self.col.T, dout)
238         self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)
239
240         dcol = np.dot(dout, self.col_W.T)
241         dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)
242
243         return dx
```

Filter의 관점에서 정리

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

input

1	2	3
4	5	6
7	8	9

filter

x = 3
y = 3

1	2
3	4

output

```
col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]
```

y:y_max:stride -> y 부터 y_max 까지 stride 간격

Filter의 관점에서 정리

input

1	2	3	4
7	8	9	10
13	14	15	16
19	20	21	22

filter

1	2	3
4	5	6
7	8	9

output

1	2
3	4

Filter 1의 관점 : 이동하면서
겹치는 영역은 4영역임

1	2
7	8

Filter 4의 관점 : 이동하면서
겹치는 영역은 4영역임

7	8
13	14

Filter 7의 관점 : 이동하면서
겹치는 영역은 4영역임

13	14
19	20

Filter 2의 관점 : 이동하면서
겹치는 영역은 4영역임

2	3
8	9

Filter 5의 관점 : 이동하면서
겹치는 영역은 4영역임

8	9
14	15

Filter 8의 관점 : 이동하면서
겹치는 영역은 4영역임

14	15
20	21

Filter 3의 관점 : 이동하면서
겹치는 영역은 4영역임

3	4
9	10

Filter 6의 관점 : 이동하면서
겹치는 영역은 4영역임

9	10
15	16

Filter 9의 관점 : 이동하면서
겹치는 영역은 4영역임

15	16
21	22

Output의 관점에서 정리

input

1	2	3	4
7	8	9	10
13	14	15	16
19	20	21	22

filter

1	2	3
4	5	6
7	8	9

output

1	2
3	4

output 1의 관점 : 이동하면서
겹치는 영역은 9영역임

1	2	3
7	8	9
13	14	15

output 2의 관점 : 이동하면서
겹치는 영역은 9영역임

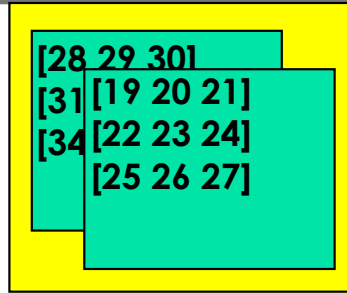
2	3	4
8	9	10
14	15	16

output 3의 관점 : 이동하면서
겹치는 영역은 9영역임

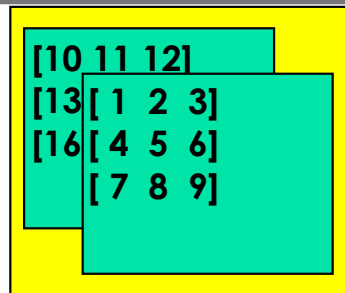
7	8	9
13	14	15
19	20	21

output 4의 관점 : 이동하면서
겹치는 영역은 9영역임

8	9	10
14	15	16
20	21	22



1



0



1

3



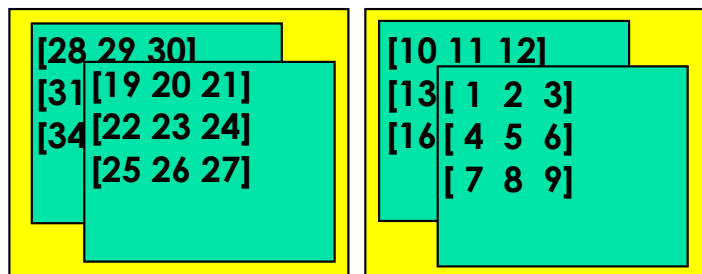
0

2

1

3

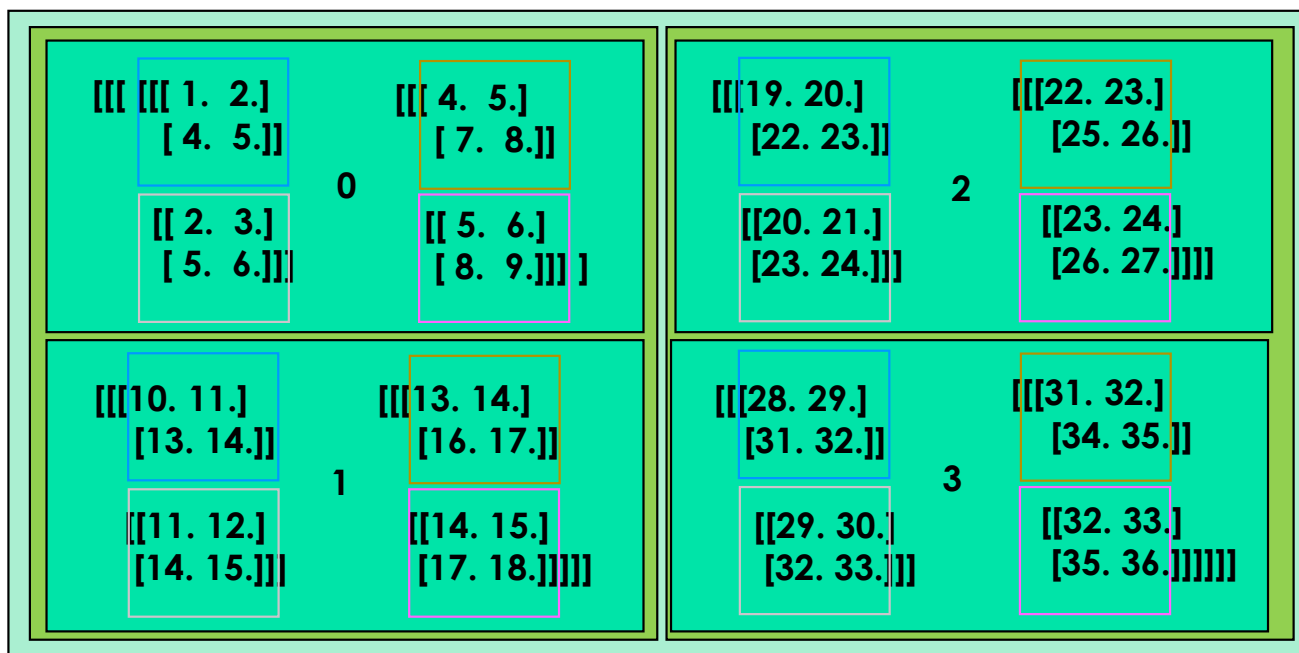
```
[
[ 1.  2.  4.  5. 10. 11. 13. 14.]
[ 2.  3.  5.  6. 11. 12. 14. 15.]
[ 4.  5.  7.  8. 13. 14. 16. 17.]
[ 5.  6.  8.  9. 14. 15. 17. 18.]
[19. 20. 22. 23. 28. 29. 31. 32.]
[20. 21. 23. 24. 29. 30. 32. 33.]
[22. 23. 25. 26. 31. 32. 34. 35.]
[23. 24. 26. 27. 32. 33. 35. 36.]
]
```



1

0

`x1 = np.arange(1, 37).reshape(2, 2, 3, 3) # 4차원 배열`



```
for y in range(filter_h):
    y_max = y + stride*out_h
    for x in range(filter_w):
        x_max = x + stride*out_w
        col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]
        print("start", y, x)
        print(col)
```



```

[
 [ 1.  2.  4.  5. 10. 11. 13. 14.]
 [ 2.  3.  5.  6. 11. 12. 14. 15.]
 [ 4.  5.  7.  8. 13. 14. 16. 17.]
 [ 5.  6.  8.  9. 14. 15. 17. 18.]
 [19. 20. 22. 23. 28. 29. 31. 32.]
 [20. 21. 23. 24. 29. 30. 32. 33.]
 [22. 23. 25. 26. 31. 32. 34. 35.]
 [23. 24. 26. 27. 32. 33. 35. 36.]
]

```

```
col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N*out_h*out_w, -1)
```

Filter의 관점에서 정리

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

input

1	2	3
4	5	6
7	8	9

filter

X = 3
Y = 3

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

output

```
col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]
```

`y:y_max:stride` -> y 부터 y_max 까지 stride 간격

Filter의 관점에서 정리

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

input

1	2	3
4	5	6
7	8	9

filter

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

output

Filter 1의 관점 : 이동하면서
겹치는 영역은 16영역임

1	2	3	4
7	8	9	10
13	14	15	16
19	20	21	22

Filter 2의 관점 : 이동하면서
겹치는 영역은 16영역임

2	3	4	5
8	9	10	11
14	15	16	17
20	21	22	23

Filter 2의 관점 : 이동하면서
겹치는 영역은 16영역임

3	4	5	6
9	10	11	12
15	16	17	18
21	22	23	24

Filter의 관점에서 정리

Filter 4의 관점 : 이동하면서
겹치는 영역은 16영역임

7	8	9	10
13	14	15	16
19	20	21	22
25	26	27	28

Filter 5의 관점 : 이동하면서
겹치는 영역은 16영역임

8	9	10	11
14	15	16	17
20	21	22	23
26	27	28	29

Filter 6의 관점 : 이동하면서
겹치는 영역은 16영역임

9	10	11	12
15	16	17	18
21	22	23	24
27	28	29	30

Filter 7의 관점 : 이동하면서
겹치는 영역은 16영역임

13	14	15	16
19	20	21	22
25	26	27	28
31	32	33	34

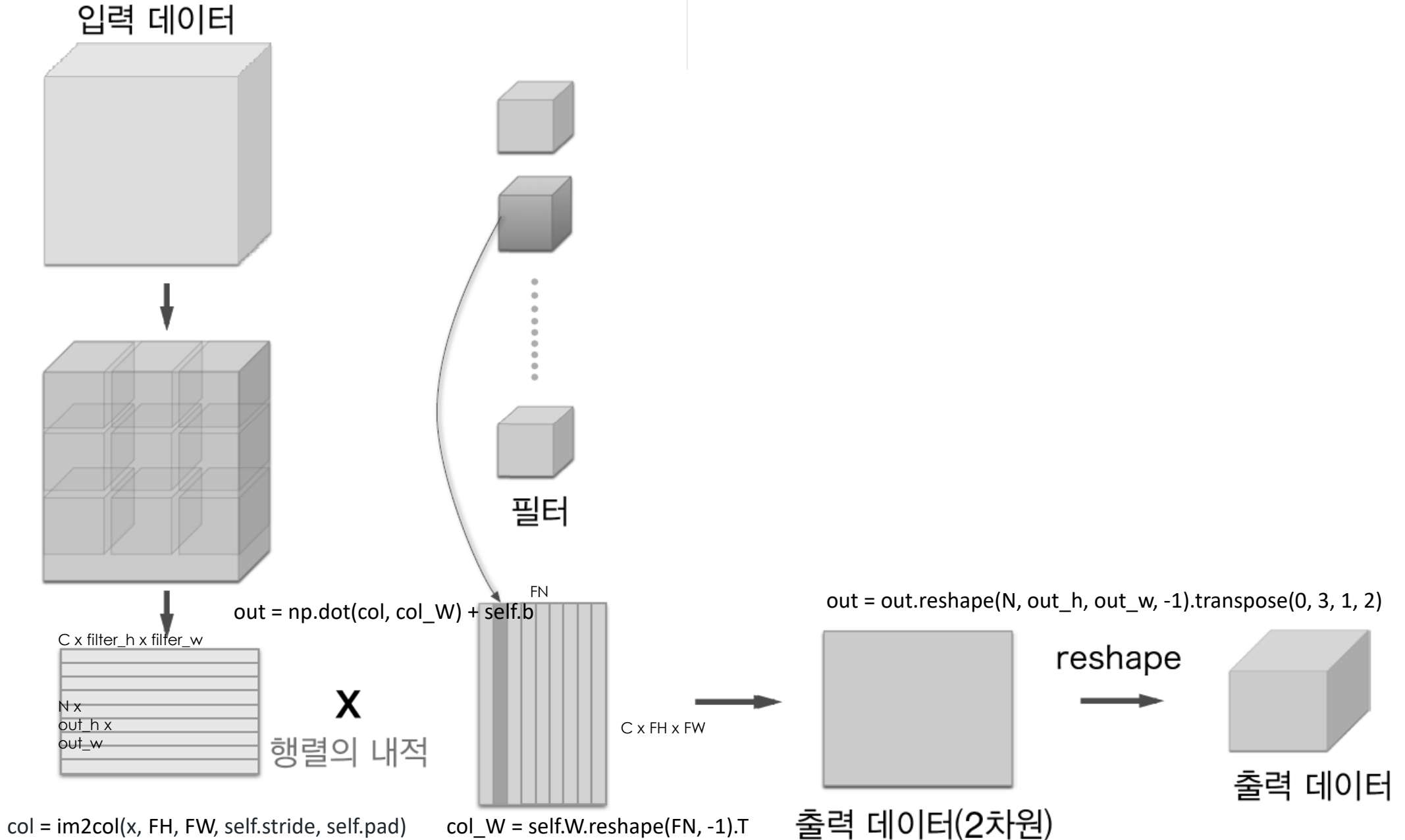
Filter 8의 관점 : 이동하면서
겹치는 영역은 16영역임

14	15	16	17
20	21	22	23
26	27	28	29
32	33	34	35

Filter 9의 관점 : 이동하면서
겹치는 영역은 16영역임

15	16	17	18
21	22	23	24
27	28	29	30
33	34	35	36

Convolution Class



/common/layers.py

```
246 class Pooling:
247     def __init__(self, pool_h, pool_w, stride=1, pad=0):
248         self.pool_h = pool_h
249         self.pool_w = pool_w
250         self.stride = stride
251         self.pad = pad
252
253         self.x = None
254         self.arg_max = None
255
256     def forward(self, x):
257         N, C, H, W = x.shape
258         out_h = int(1 + (H - self.pool_h) / self.stride)
259         out_w = int(1 + (W - self.pool_w) / self.stride)
260
261         col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
262         col = col.reshape(-1, self.pool_h*self.pool_w)
263
264         arg_max = np.argmax(col, axis=1)
265         out = np.max(col, axis=1)
266         out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)
267
268         self.x = x
269         self.arg_max = arg_max
270
271         return out
272
273     def backward(self, dout):
274         dout = dout.transpose(0, 2, 3, 1)
275
276         pool_size = self.pool_h * self.pool_w
277         dmax = np.zeros((dout.size, pool_size))
278         dmax[np.arange(self.arg_max.size), self.arg_max.flatten()] = dout.flatten()
279         dmax = dmax.reshape(dout.shape + (pool_size,))
280
281         dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
282         dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride, self.pad)
283
284         return dx
```


/common/multi_layer_net.py

```
10 class MultiLayerNet:
11     """완전연결 다층 신경망
12
13     Parameters
14     -----
15     input_size : 입력 크기 (MNIST의 경우엔 784)
16     hidden_size_list : 각 은닉층의 뉴런 수를 담은 리스트 (e.g. [100, 100, 100])
17     output_size : 출력 크기 (MNIST의 경우엔 10)
18     activation : 활성화 함수 - 'relu' 혹은 'sigmoid'
19     weight_init_std : 가중치의 표준편차 지정 (e.g. 0.01)
20         'relu'나 'he'로 지정하면 'He 초기값'으로 설정
21         'sigmoid'나 'xavier'로 지정하면 'Xavier 초기값'으로 설정
22     weight_decay_lambda : 가중치 감소(L2 법칙)의 세기
23     """
24     def __init__(self, input_size, hidden_size_list, output_size,
25                 activation='relu', weight_init_std='relu', weight_decay_lambda=0):
26         self.input_size = input_size          33         # 가중치 초기화
27         self.output_size = output_size        34         self.__init_weight(weight_init_std)
28         self.hidden_size_list = hidden_size_list 35
29         self.hidden_layer_num = len(hidden_size_list) 36         # 계층 생성
30         self.weight_decay_lambda = weight_decay_lambda 37         activation_layer = {'sigmoid': Sigmoid, 'relu': Relu}
31         self.params = {}                      38         self.layers = OrderedDict()
32                                             39         for idx in range(1, self.hidden_layer_num+1):
33                                             40             self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
34                                                         self.params['b' + str(idx)])
35                                             41
36                                             42             self.layers['Activation_function' + str(idx)] = activation_layer[activation]()
37                                             43
38                                             44             idx = self.hidden_layer_num + 1
39                                             45             self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
40                                                         self.params['b' + str(idx)])
41                                             46
42                                             47
43                                             48             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).
129
130     Parameters
131     -----
132     x : 입력 데이터
133     t : 정답 레이블
134
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
158
159     return grads
```


/common/multi_layer_net_extend.py

```
1  # coding: utf-8
2  import sys, os
3  sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
4  import numpy as np
5  from collections import OrderedDict
6  from common.layers import *
7  from common.gradient import numerical_gradient
8
9  class MultiLayerNetExtend:
10     """완전 연결 다층 신경망(확장판)
11     가중치 감소, 드롭아웃, 배치 정규화 구현
12
13     Parameters
14     -----
15     input_size : 입력 크기 (MNIST의 경우엔 784)
16     hidden_size_list : 각 은닉층의 뉴런 수를 담은 리스트 (e.g. [100, 100, 100])
17     output_size : 출력 크기 (MNIST의 경우엔 10)
18     activation : 활성화 함수 - 'relu' 혹은 'sigmoid'
19     weight_init_std : 가중치의 표준편차 지정 (e.g. 0.01)
20         'relu'나 'he'로 지정하면 'He 초기값'으로 설정
21         'sigmoid'나 'xavier'로 지정하면 'Xavier 초기값'으로 설정
22     weight_decay_lambda : 가중치 감소(L2 법칙)의 세기
23     use_dropout : 드롭아웃 사용 여부
24     dropout_ratio : 드롭아웃 비율
25     use_batchNorm : 배치 정규화 사용 여부
26     """
27
28     def __init__(self, input_size, hidden_size_list, output_size,
29                 activation='relu', weight_init_std='relu', weight_decay_lambda=0,
30                 use_dropout = False, dropout_ratio = 0.5, use_batchnorm=False):
31         self.input_size = input_size
32         self.output_size = output_size
33         self.hidden_size_list = hidden_size_list
34         self.hidden_layer_num = len(hidden_size_list)
35         self.use_dropout = use_dropout
36         self.weight_decay_lambda = weight_decay_lambda
37         self.use_batchnorm = use_batchnorm
38         self.params = {}
39
40         # 가중치 초기화
41         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
49         if self.use_batchnorm:
50             self.params['gamma' + str(idx)] = np.ones(hidden_size_list[idx-1])
51             self.params['beta' + str(idx)] = np.zeros(hidden_size_list[idx-1])
52             self.layers['BatchNorm' + str(idx)] = BatchNormalization(self.params['gamma' + str(idx)], self.params['beta' + str(idx)])
53
54         self.layers['Activation_function' + str(idx)] = activation_layer[activation]()
55
56         if self.use_dropout:
57             self.layers['Dropout' + str(idx)] = Dropout(dropout_ratio)
58
59     idx = self.hidden_layer_num + 1
60     self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)], self.params['b' + str(idx)])
61
62     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 초기값'으로 설정
70             'sigmoid'나 'xavier'로 지정하면 'Xavier 초기값'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

```

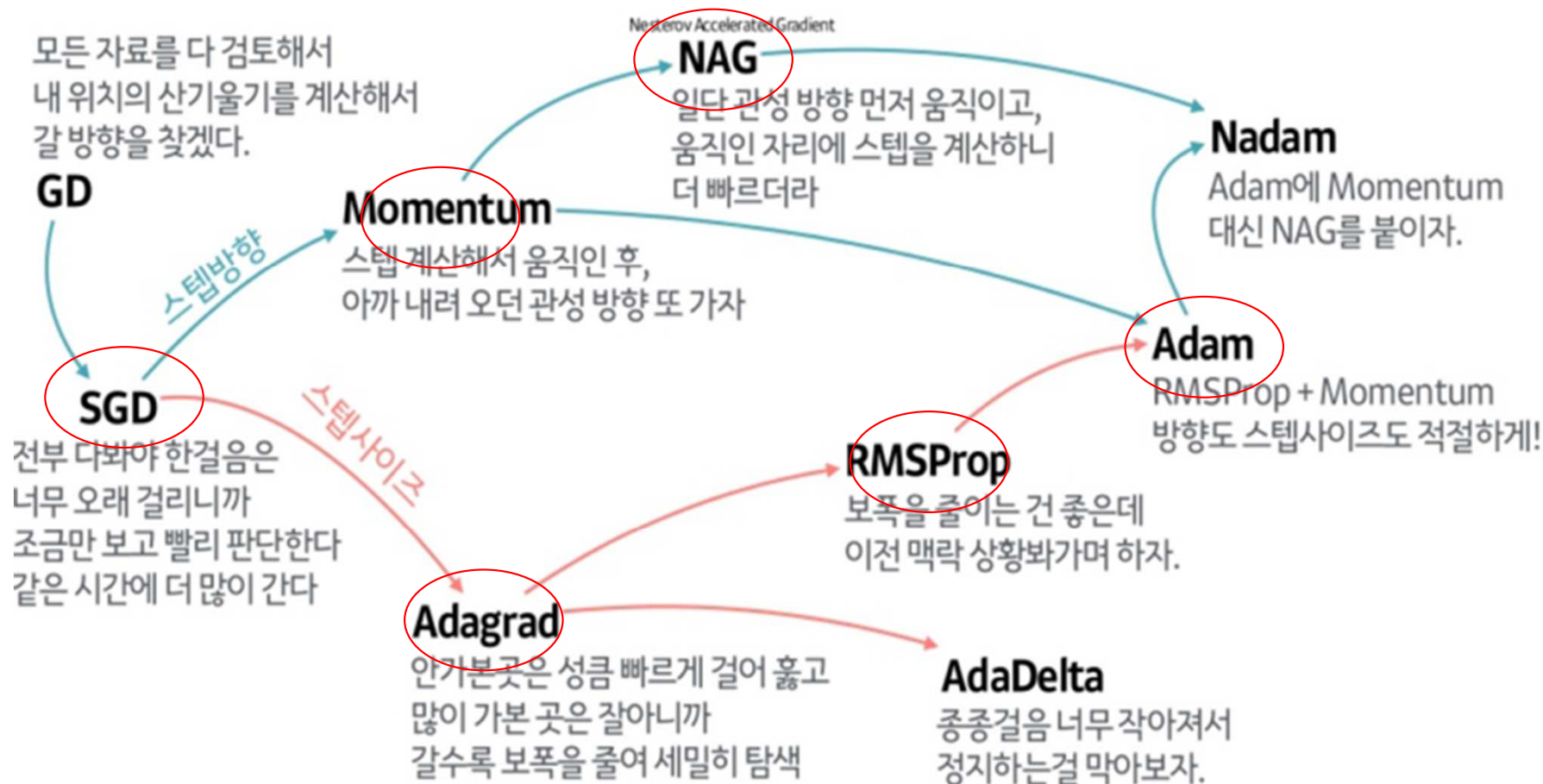
/common/multi_layer_net_extend.py

```
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
115
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
```

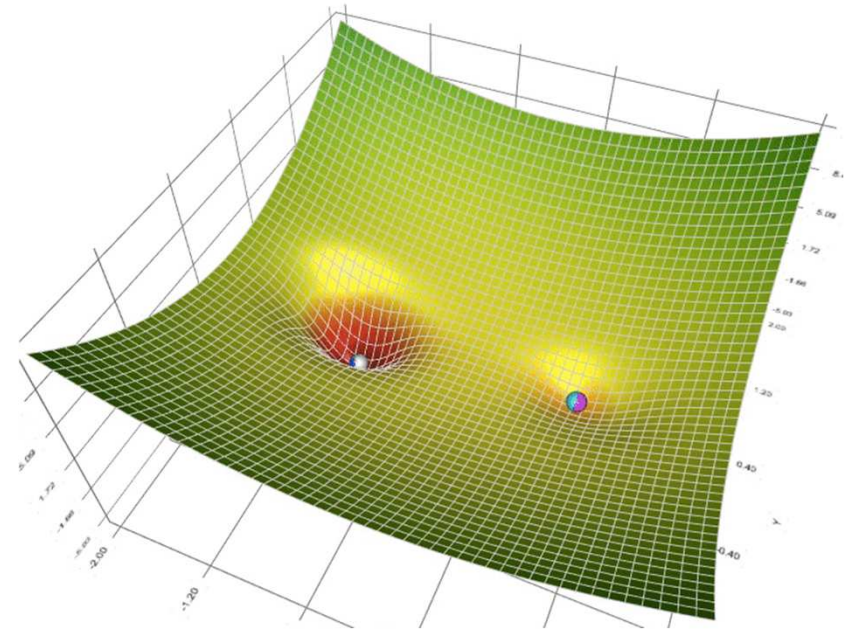
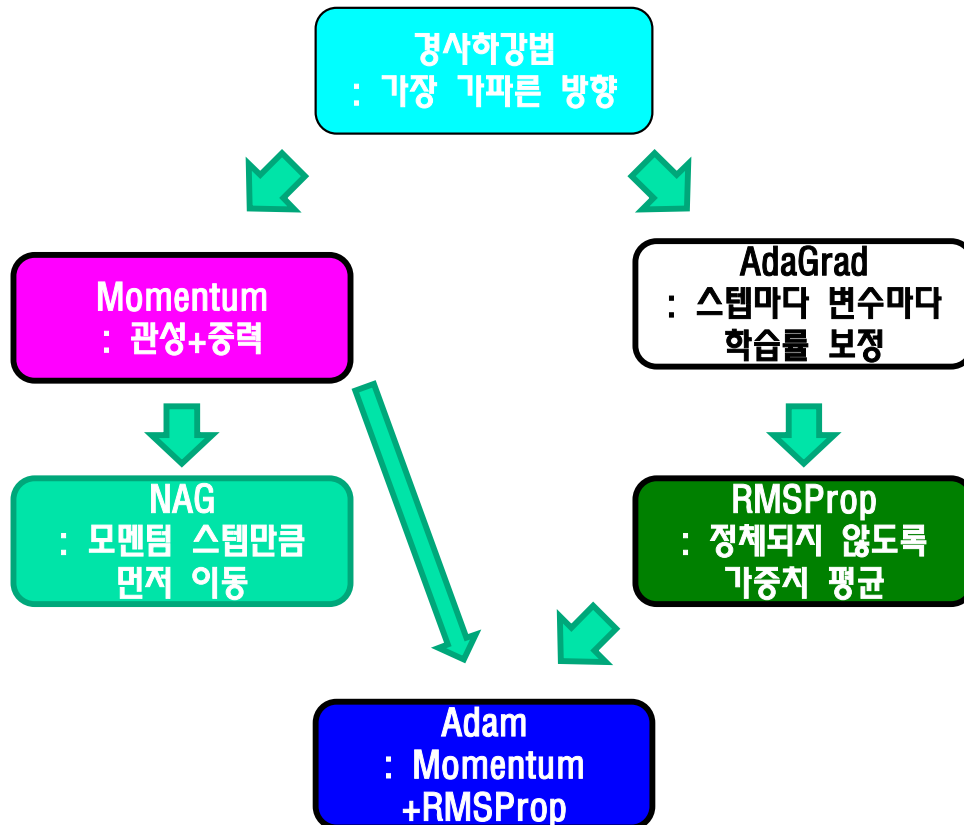
/common/multi_layer_net_extend.py

```
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



옵티마이저 계보



```
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]
```

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

```
16 class Momentum:    스텝 계산해서 움직인 후, 아까 내려 오던 관성 방향 또 가자
```

```
17
18     """모멘텀 SGD"""
19
20     def __init__(self, lr=0.01, momentum=0.9):
21         self.lr = lr
22         self.momentum = momentum
23         self.v = None
24
25     def update(self, params, grads):
26         if self.v is None:
27             self.v = {}
28             for key, val in params.items():
29                 self.v[key] = np.zeros_like(val)
30
31         for key in params.keys():
32             self.v[key] = self.momentum*self.v[key] - self.lr*grads[key]
33             params[key] += self.v[key]
```

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

/common/optimizer.py

36 `class Nesterov:` 일단 관성 방향 먼저 움직이고, 움직인 자리에 스텝을 계산하니 더 빠르더라

37

38 `"""Nesterov's Accelerated Gradient (http://arxiv.org/abs/1212.0901)"""`

39 `# NAG는 모멘텀에서 한 단계 발전한 방법이다. (http://newsight.tistory.com/224)`

40

41 `def __init__(self, lr=0.01, momentum=0.9):`

42 `self.lr = lr`

43 `self.momentum = momentum`

44 `self.v = None`

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

46 `def update(self, params, grads):`

47 `if self.v is None:`

48 `self.v = {}`

49 `for key, val in params.items():`

50 `self.v[key] = np.zeros_like(val)`

51

52 `for key in params.keys():`

53 `self.v[key] *= self.momentum`

54 `self.v[key] -= self.lr * grads[key]`

55 `params[key] += self.momentum * self.momentum * self.v[key]`

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

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

/common/optimizer.py

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

```
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)
```

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

/common/optimizer.py

78 `class RMSprop:` 보폭을 줄이는 건 좋은데 이전 맥락 상황 봐가며 가자.
79

80 `"""RMSprop"""`

81
82 `def __init__(self, lr=0.01, decay_rate = 0.99):`

83 `self.lr = lr`

84 `self.decay_rate = decay_rate`

85 `self.h = None`

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

86
87 `def update(self, params, grads):`

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

88 `if self.h is None:`

89 `self.h = {}`

90 `for key, val in params.items():`

91 `self.h[key] = np.zeros_like(val)`

92
93 `for key in params.keys():`

94 `self.h[key] *= self.decay_rate`

95 `self.h[key] += (1 - self.decay_rate) * grads[key] * grads[key]`

96 `params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)`

/common/optimizer.py

```
99 class Adam:          방향도 스텝사이즈도 적절하게
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
```

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

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

```
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)
```

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

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

```
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)
```

/common/trainer.py

```
7 class Trainer:
8     """신경망 훈련을 대신 해주는 클래스
9     """
10    def __init__(self, network, x_train, t_train, x_test, t_test,
11                  epochs=20, mini_batch_size=100,
12                  optimizer='SGD', optimizer_param={'lr':0.01},
13                  evaluate_sample_num_per_epoch=None, verbose=True):
14        self.network = network
15        self.verbose = verbose
16        self.x_train = x_train
17        self.t_train = t_train
18        self.x_test = x_test
19        self.t_test = t_test
20        self.epochs = epochs
21        self.batch_size = mini_batch_size
22        self.evaluate_sample_num_per_epoch = evaluate_sample_num_per_epoch
23
24        # optimizer
25        optimizer_class_dict = {'sgd':SGD, 'momentum':Momentum, 'nesterov':Nesterov,
26                                'adagrad':AdaGrad, 'rmsprop':RMSprop, 'adam':Adam}
27        self.optimizer = optimizer_class_dict[optimizer.lower()](**optimizer_param)
28
29        self.train_size = x_train.shape[0]
30        self.iter_per_epoch = max(self.train_size / mini_batch_size, 1)
31        self.max_iter = int(epochs * self.iter_per_epoch)
32        self.current_iter = 0
33        self.current_epoch = 0
34
35        self.train_loss_list = []
36        self.train_acc_list = []
37        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))
```



```
[[[ 1 2 3]
 [ 4 5 6]
 [ 7 8 9]]
```

```
[[10 11 12]
 [13 14 15]
 [16 17 18]]]
```

```
[[[19 20 21]
 [22 23 24]
 [25 26 27]]
```

```
[[28 29 30]
 [31 32 33]
 [34 35 36]]]]
```

```
[28 29 30]
[31
[34 [19 20 21]
      [22 23 24]
      [25 26 27]]
```

```
[10 11 12]
[13
[16 [ 1 2 3]
      [ 4 5 6]
      [ 7 8 9]]
```

```
[[[[[ 1. 2.] [ 4. 5.]]
 [[ 2. 3.] [[10. 11.]
 [ 5. 6.]] [13. 14.]]]
 [[ 4. 5.] [[ 2. 3.]
 [ 7. 8.]] [ 5. 6.]]
 [[ 5. 6.] [[11. 12.]
 [ 8. 9.]]] [14. 15.]]]
 [[[[10. 11.] [[ 4. 5.]
 [13. 14.]] [ 7. 8.]]
 [[11. 12.] [[13. 14.]
 [14. 15.]]] [16. 17.]]]
 [[[[13. 14.] [[ 5. 6.]
 [16. 17.]] [ 8. 9.]]
 [[14. 15.] [[14. 15.]
 [17. 18.]]]]] [17. 18.]]]]]
 [[[[[19. 20.] [[[[19. 20.]
 [22. 23.]] [22. 23.]]
 [[20. 21.] [[28. 29.]
 [23. 24.]]] [31. 32.]]]
 [[[[22. 23.] [[[[20. 21.]
 [25. 26.]] [23. 24.]]
 [[23. 24.] [[29. 30.]
 [26. 27.]]]]] [32. 33.]]]]]
 [[[[[28. 29.] [[[[22. 23.]
 [31. 32.]] [25. 26.]]
 [[29. 30.] [[31. 32.]
 [32. 33.]]] [34. 35.]]]
 [[[[31. 32.] [[[[23. 24.]
 [34. 35.]] [26. 27.]]
 [[32. 33.] [[32. 33.]
 [35. 36.]]]]]]] [35. 36.]]]]]]]
```

<https://cs231n.github.io/assets/conv-demo/index.html>

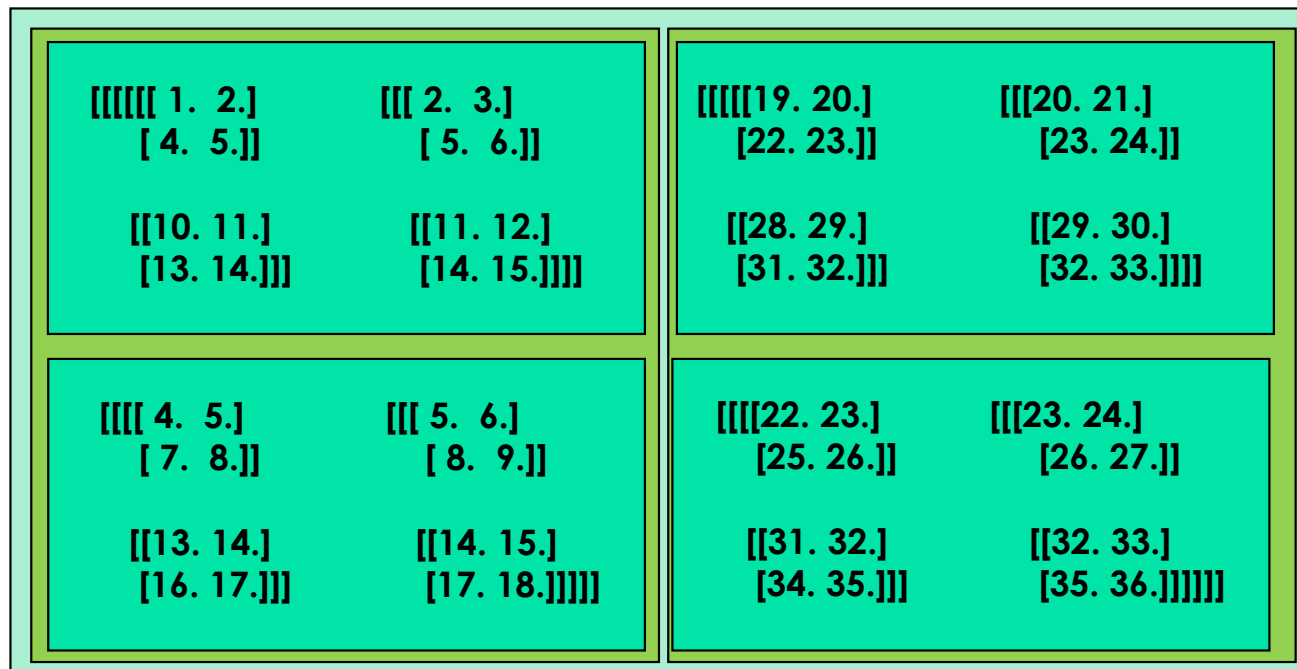
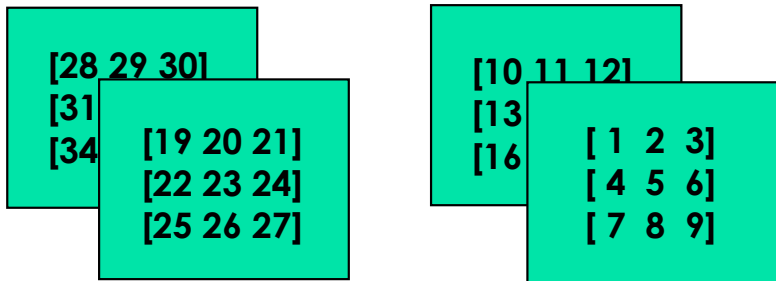


Image data

D0	D1	D2
D3	D4	D5
D6	D7	D8

$D[0,0,:,:]$

D0	D1	D2
D3	D4	D5
D6	D7	D8

$D[0,1,:,:]$

D0	D1	D2
D3	D4	D5
D6	D7	D8

$D[0,2,:,:]$

Filter data

F0	F1
F2	F3

$F[0,:,:,:]$

F0	F1
F2	F3

$F[1,:,:,:]$

$N = 1$
 $C = 3$
 $H = 3$
 $W = 3$
 $K = 2$
 $R = 2$
 $S = 2$
 $u=v = 1$
 $pad_h = 0$
 $pad_w = 0$

D4	D5	D7	D8
D3	D4	D6	D7
D1	D2	D4	D5
D0	D1	D3	D4
D4	D5	D7	D8
D3	D4	D6	D7
D1	D2	D4	D5
D0	D1	D3	D4
D4	D5	D7	D8
D3	D4	D6	D7
D1	D2	D4	D5
D0	D1	D3	D4

F0	F1	F2	F3	F0	F1	F2	F3	F0	F1	F2	F3
G0	G1	G2	G3	G0	G1	G2	G3	G0	G1	G2	G3

F_m

O_m

Convolution 역전파

합성곱층 역전파

합성곱층은 다음과 같이 이루어진다.

im2col → **Affine** 변환 → **Reshape**

이것의 역전파는

im2col의 역전파 ← **Affine**층 역전파 ← **Reshape**의 역전파

Affine층 역전파는 잘 이해하고 있다.

reshape의 역전파와 im2col의 역전파인 col2im에 대해 알아보자.

재배열의 역전파

재배열의 역전파는 역재배열이다.

데이터를 **im2col** 할 때, 아래와 같은 순전파 과정이 이루어진다.

순전파 : **reshape(N, OH, OW, FN) → transpose(0, 3, 1, 2)**

역전파는 아래와 같다.

역전파 : **reshape(-1, FN) ← transpose(0, 2, 3, 1)**

reshape(N, OH, OW, FN)을 하기 전의 **shape**은 **(N, FN)**이다.

transpose(0, 2, 3, 1)을 취하게 되면 순전파를 하기 전의 상태로 역재배열 할 수 있다.

필터를 **im2col** 할 때, 아래와 같은 순전파 과정이 이루어진다.

순전파 : **reshape(FN, -1) → transpose(1, 0)**

역전파는 아래와 같다.

역전파 : **reshape(FN, C, FH, FW) ← transpose(1, 0)**

col2im

im2col의 역전파를 col2im이라고 한다.

데이터에 대한 im2col은 두 가지 경우로 나뉜다.

원도우가 겹치지 않을 경우 : 단순한 재배열 → 역재배열을 이용한 역전파

원도우가 겹칠 경우 : repeat노드 등장 → sum노드를 이용한 역전파

예시)

데이터 x

$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$$

FH x FW = 2 x 2 . padding = 0. stride = 1로 im2col 적용

$$\text{col} = \begin{pmatrix} x_{11} & x_{12} & x_{21} & x_{22} \\ x_{12} & x_{13} & x_{22} & x_{23} \\ x_{21} & x_{22} & x_{31} & x_{32} \\ x_{22} & x_{23} & x_{32} & x_{33} \end{pmatrix}$$

x12, x21, x22, x23, x32는 copy가 일어난다.

im2col로 흘러들어온 미분이 아래와 같다.

$$D = \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{pmatrix}$$

재배열의 역전파는 역재배열, copy의 역전파는 sum 이기 때문에 역전파의 결과는 아래와 같다.

$$= \frac{\partial L}{\partial X} = \begin{pmatrix} d_{11} & d_{12} + d_{21} & d_{22} \\ d_{13} + d_{31} & d_{14} + d_{23} + d_{32} + d_{41} & d_{24} + d_{42} \\ d_{33} & d_{34} + d_{43} & d_{44} \end{pmatrix}$$

<https://pintelligence.tistory.com/33>



```
class Convolution:
```

```
...
```

```
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
```

흘러들어온 미분에 대하여 $\text{transpose}(0, 2, 3, 1) \rightarrow \text{reshape}(-1, \text{FN})$

db : repeat노드의 역전파 sum을 이용

dW : Affine 역전파 $\rightarrow \text{transpose}(1, 0) \rightarrow \text{reshape}(\text{FN}, \text{C}, \text{FH}, \text{FW})$

dx : Affine 역전파 $\rightarrow \text{col2im}$