메디치소프트 기술연구소

2018.08.06

딥러닝 5일

정규방정식(Normal Equation)- 경사하강법(Gradient Descent) 비교

- **필요성**

- 경사하강법: 미분을 통해 비용함수(cost)를 최소화하는 θ 값 을 찾는 방법
- 정규방정식(normal equation)은 미분을 수행하지 않고 빠르게 θ 값을 찾을 수 있다는 장점 존재

비교	경사하강법	정규방정식
공식	$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$	$\theta = (X^T X)^{-1} X^T y$
장점	X(features)의 수가 많을 수록 유리	반복하여 수행하지 않는다
단점	미분을 계속 해야하기 때문에 수많은 반복 수행이 필요	X(features)의 수가 많을수록 불리
선택 기준	Andrew 교수님 “n이 10000개(즉 행렬X가 m x 10001 Marix를 초과하기 시작하면 경사하강법사용”	

차원축소 or 정규화(Regualrization)or Scailing

정규방정식(Normal Equation)- 예제(2배열)

```
import numpy as np
from sklearn.datasets import fetch_california_housing
import tensorflow as tf
housing = fetch_california_housing()
m, n = housing.data.shape
housing_data_plus_bias = np.c_[np.ones((m, 1)), housing.data] #편향에 대한 입력 특성(X0=1)을 추가
X = tf.constant(housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
XT = tf.transpose(X) #전치 행렬
theta = tf.matmul(tf.matmul(tf.matrix_inverse(tf.matmul(XT, X)), XT), y) #inverse 역행렬
print(X)
print(y)
print(XT)
print(theta)
with tf.Session() as sess:
    theta_value = theta.eval()
    print(theta_value)
```

```
Tensor("x:0", shape=(20640, 9), dtype=float32)
Tensor("y:0", shape=(20640, 1), dtype=float32)
Tensor("transpose:0", shape=(9, 20640), dtype=float32)
Tensor("MatMul_2:0", shape=(9, 1), dtype=float32)
```

결과값:

```
[[-3.7185181e+01]
 [ 4.3633747e-01]
 [ 9.3952334e-03]
 [-1.0711310e-01]
 [ 6.4479220e-01]
 [-4.0338000e-06]
 [-3.7813708e-03]
 [-4.2348403e-01]
 [-4.3721911e-01]]
```

경사하강법(Gradient Descent)- 예제(2배열)

```
import numpy as np
from sklearn.datasets import fetch_california_housing
import tensorflow as tf
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()
m, n = housing.data.shape
scaler = StandardScaler()
scaled_housing_data = scaler.fit_transform(housing.data)

n_epochs = 1000
learning_rate = 0.01
scaled_housing_data_plus_bias = np.c_[np.ones((m, 1)), scaled_housing_data]
X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42), name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
```

스케일링:
전체 자료의 분포를 평균 0, 분산 1이 되도록 만드는 과정

Scaler 클래스의 사용 방법은 다음과 같다.
1.클래스 객체 생성
2.fit() 메서드와 트레이닝 데이터를 사용하여 변환 계수 추정
3.transform() 메서드를 사용하여 실제로 자료를 변환

경사하강법(Gradient Descent)- 예제(2배열)

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("에포크", epoch, "MSE =", mse.eval())
        sess.run(training_op)

    best_theta = theta.eval()

print("best_theta:")
print(best_theta)
```

. 반복 루프: 훈련 단계를 반복해서 실행(n_ecpochs)
-100번 반복마다 현재의 평균 제곱에러(mse)출력
. MSE는 매 반복에서 값이 줄어들어야 함

결과값

```
에포크 0 MSE = 2.7544262
에포크 100 MSE = 0.632222
에포크 200 MSE = 0.5727805
에포크 300 MSE = 0.5585007
에포크 400 MSE = 0.54907
에포크 500 MSE = 0.54228795
에포크 600 MSE = 0.5373789
에포크 700 MSE = 0.533822
에포크 800 MSE = 0.5312425
에포크 900 MSE = 0.5293704
```

```
best_theta:
[[ 2.06855249e+00]
 [ 7.74078071e-01]
 [ 1.31192386e-01]
 [-1.17845066e-01]
 [ 1.64778143e-01]
 [ 7.44078017e-04]
 [-3.91945094e-02]
 [-8.61356676e-01]
 [-8.23479772e-01]]
```

Placeholder

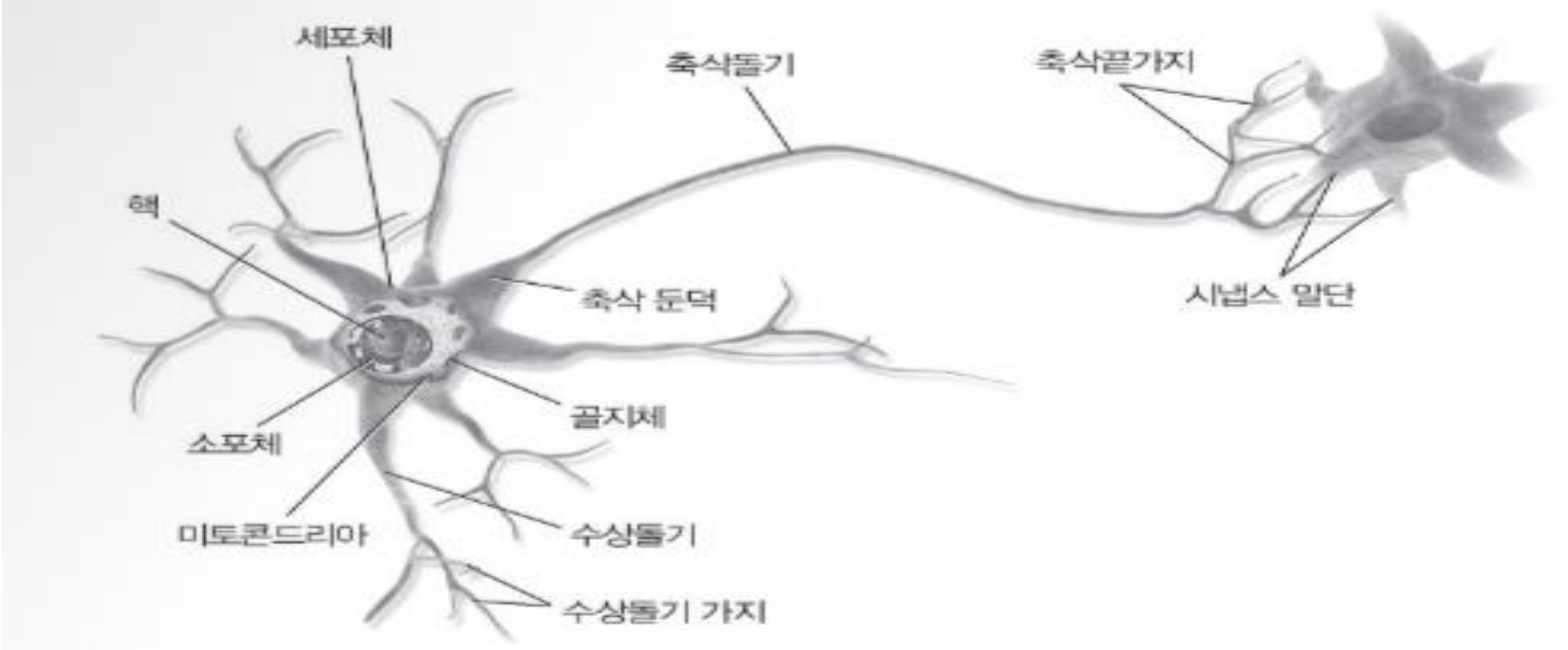
- 실제로 아무 계산도 하지 않는 특수한 노드, 실행시 데이터를 출력
- 훈련을 하는 동안 tensor에 훈련 데이터를 전달하기 위해 사용
  (실행시 placeholder에 값을 지정하지 않으면 예외 발생)

```
import tensorflow as tf
A = tf.placeholder(tf.float32, shape=(None, 3))
B = A + 5
with tf.Session() as sess:
    B_val_1 = B.eval(feed_dict={A: [[1, 2, 3]]})
    B_val_2 = B.eval(feed_dict={A: [[4, 5, 6], [7, 8, 9]]})
print(B_val_1)
print(B_val_2)
```

결과값

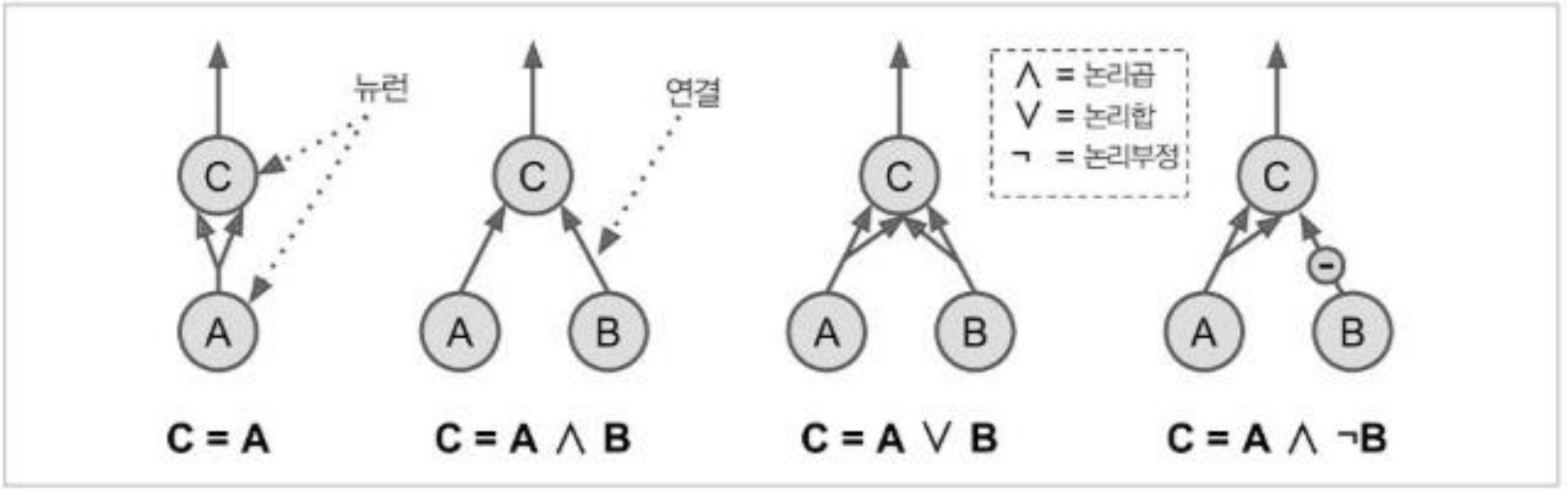
```
[[6. 7. 8.]]
[[ 9. 10. 11.]
 [12. 13. 14.]]
```

생물학적 뉴런



개개의 뉴런은 단순하게 동작하지만 수십억 개의 뉴런으로 구성된 거대한 네트워크 조직 구성, 각 뉴런은 수천개의 다른 뉴런 연결

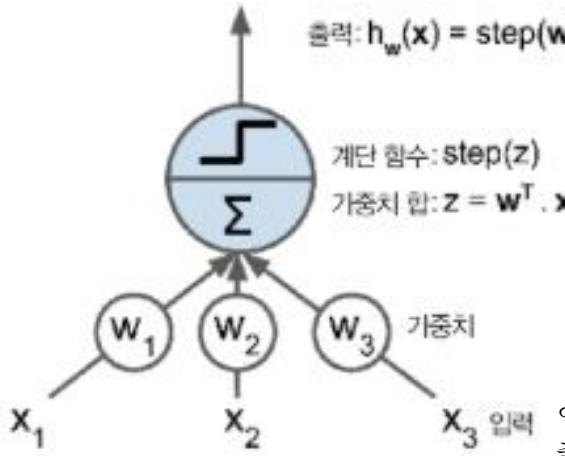
인공 뉴런



. 항등 함수: 뉴런 A가 활성화되면 뉴런 C도 활성화
. 논리곱 연산: 뉴런 A와 B가 모두 활성화될 때만 뉴런 C가 활성화
. 논리합 연산: 뉴런 A와 B 중 하나가 활성화되면 뉴런 C가 활성화
. 노리부정 :뉴런 A가 활성화되고 뉴런 B가 비활성화될 때 뉴런 C가 활성화

퍼셉트론

. 가장 단순한 인공 신경망 구조
. 1957년 프랑크 로젠블라트가 제안
. TLU(threshold logic unit)라는 조금 다른 형태의 인공 뉴런을 기반
. 입력과 출력이 (이진 on/off) 어떤 숫자이며 입력 연결은 가중치와 연관
. 로지스틱 회귀 분류기, 선형 SVM , 임계값을 넘어서면 양성 클래스출력



$$heaviside(z) = \begin{cases} 0 & z < 0\text{일 때} \\ 1 & z \geq 0\text{일 때} \end{cases} \qquad sgn(z) = \begin{cases} -1 & z < 0\text{일 때} \\ 0 & z = 0\text{일 때} \\ +1 & z > 0\text{일 때} \end{cases}$$

입력층: d+1(d:특징 벡터의 차원, 1:bias(항상 1)
출력층: 1개 노드

TLU

※계단 함수: 헤비 사이드 계단함수

퍼셉트론(헤브의 규칙 훈련알고리즘)

. 시에그리드 로웰 "서로 활성화하는 세포가 서로 연결된다"
. 두 뉴런이 동일한 출력을 낼 때마다 그들 사이의 연결 가중치가 증가
. 네트워크가 만드는 에러를 반영하도록 조금 변형된 규칙을 사용하여 훈련하며 잘못된 출력을 만드는 연결은 강화시키지 않는다.
. 한 번에 한 개의 샘플이 주입되면 각 샘플에 대해 예측이 만들어진다.
. 잘못된 예측을 하는 모든 출력 뉴런에 대해 올바른 예측을 만들 수 있으며 입력에 연결된 가중치 강화

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i$$

퍼셉트론 학습 규칙(가중치 업데이트)

- $w_{i,j}$는 i번째 입력 뉴런과 j번째 출력 뉴런 사이를 연결하는 가중치입니다.
- x_i는 현재 훈련 샘플의 i번째 뉴런의 입력값입니다.
- $\hat{y}_j$는 현재 훈련 샘플의 j번째 출력 뉴런의 출력값입니다.
- y_j는 현재 훈련 샘플의 j번째 출력 뉴런의 타깃값입니다.
- η는 학습률입니다.

퍼셉트론(예제-Iris 데이터 머신러닝)

```
# Load required libraries
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Load the iris dataset
iris = datasets.load_iris()
# Create our X and y data
X = iris.data
y = iris.target
# View the first five observations of our y data
print(y[:5])
# Split the data into 70% training data and 30% test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# Train the scaler, which standarizes all the features to have mean=0 and unit variance
sc = StandardScaler()
sc.fit(X_train)
# Apply the scaler to the X training data
X_train_std = sc.transform(X_train)
# Apply the SAME scaler to the X test data
X_test_std = sc.transform(X_test)
```

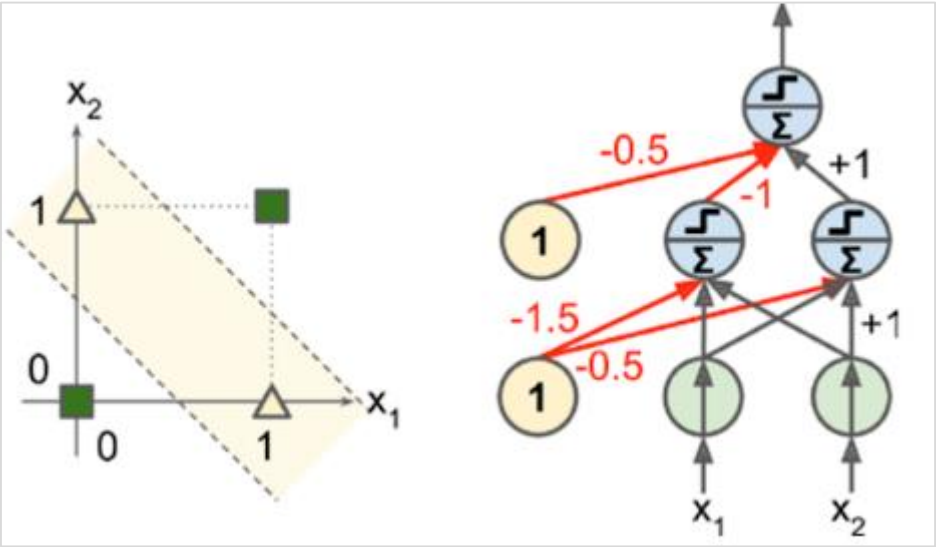
퍼셉트론(예제-Iris 데이터 머신러닝)

```
# Create a perceptron object with the parameters:
# 40 iterations (epochs) over the data, and a learning rate of 0.1
ppn = Perceptron(max_iter=40, eta0=0.1, random_state=0)
# Train the perceptron
ppn.fit(X_train_std, y_train)
# Apply the trained perceptron on the X data to make predicts for the y test data
y_pred = ppn.predict(X_test_std)
# View the predicted y test data
print(y_pred)
# View the true y test data
print(y_test)
# View the accuracy of the model, which is: 1 - (observations predicted wrong / total observations)
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

참고: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

다층 퍼셉트론(Multi-Layer Perceptron)

- 퍼셉트론(마빈 민스키와 시모어 페퍼트)의 배타적 논리합(XOR)분류 문제 해결

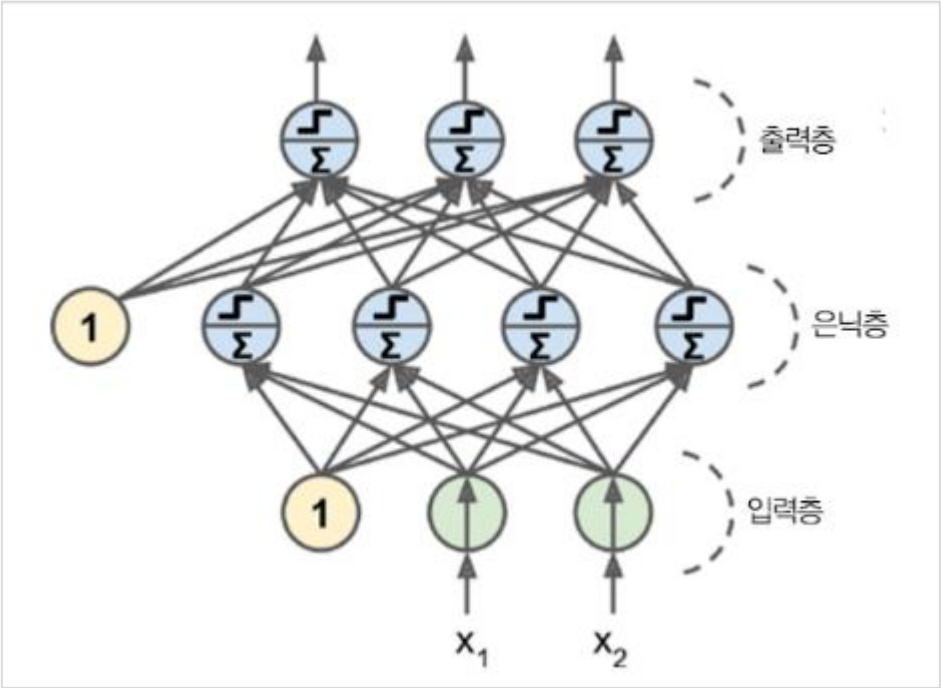


X1	X2	XOR	Y1	Y2	Y
0	0	0 (-)	0(-8)	1(3)	0(-5)
0	1	1 (+)	0(-3)	0(-4)	1(6)
1	0	1 (-)	0(-3)	0(-4)	1(6)
1	1	0 (+)	1(2)	0(-11)	0(-5)

입력(0,0)이나 (1,1) => 출력 0
입력(0,1)이나 (1,0) => 출력 1

다층 퍼셉트론와 역전파

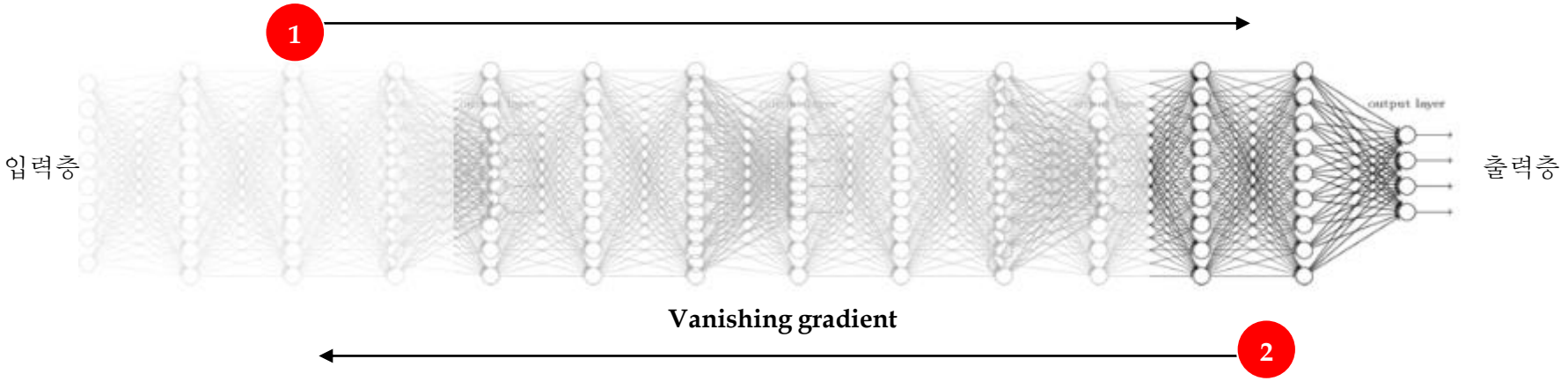
. 입력층 1개와 은닉층이라 불리는 1개이상의 TLU층과 마지막층인 출력층으로 구성
. 출력층을 제외하고 모든 층은 편향 뉴런을 포함하며 다음층과 완전히 연결
. 인공신경망의 은닉층이 2개 이상일 때 => 심층 신경망(DNN: Deep Neural Network)



다층 퍼센트론 훈련을 시킬 방법 => 실패

역전파(backpropagation)

역전파



1. 알고리즘이 각 훈련 샘플을 네트워크에 주입, 연속되는 각 층의 뉴런마다 출력 계산 (정방향 계산) => 오차측정
2. 역방향으로 각 층을 거치면서 각 연결이 오차에 기여한 정도 측정(역방향계산)
3. 이 오차가 감소하도록 가중치를 조금씩 조정

↓

뉴런이 (S자 모양의)시그모이드 활성화 함수를 구현 => ReLU함수가 인공신경망에서 더 잘 작동

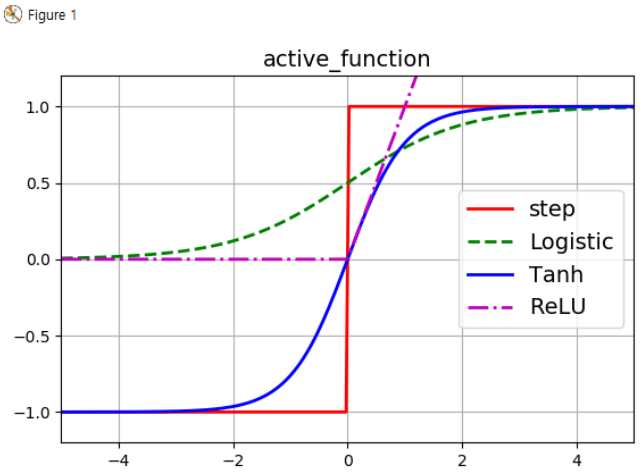
활성화 함수

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import os
PROJECT_ROOT_DIR = "." # 그림을 저장할 폴더
CHAPTER_ID = "ann"
def save_fig(fig_id, tight_layout=True):
    path = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID, fig_id + ".png")
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format='png', dpi=300)
def logit(z):
    return 1 / (1 + np.exp(-z))
def relu(z):
    return np.maximum(0, z)
def derivative(f, z, eps=0.000001):
    return (f(z + eps) - f(z - eps))/(2 * eps)
```

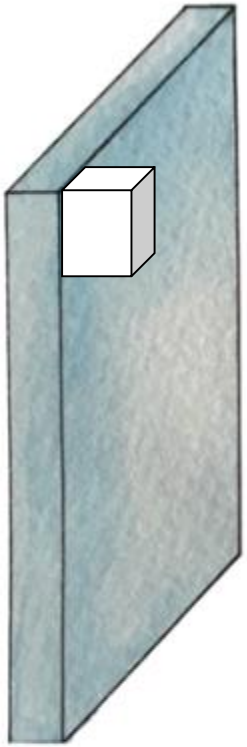
활성화 함수

```
z = np.linspace(-5, 5, 200)
plt.figure(figsize=(11,4))
plt.subplot(121)
plt.plot(z, np.sign(z), "r-", linewidth=2, label="step")
plt.plot(z, logit(z), "g--", linewidth=2, label="Logistic")
plt.plot(z, np.tanh(z), "b-", linewidth=2, label="Tanh")
plt.plot(z, relu(z), "m-.", linewidth=2, label="ReLU")
plt.grid(True)
plt.legend(loc="center right", fontsize=14)
plt.title("active_function", fontsize=14)
plt.axis([-5, 5, -1.2, 1.2])
save_fig("activation_functions_plot")
plt.show()
```

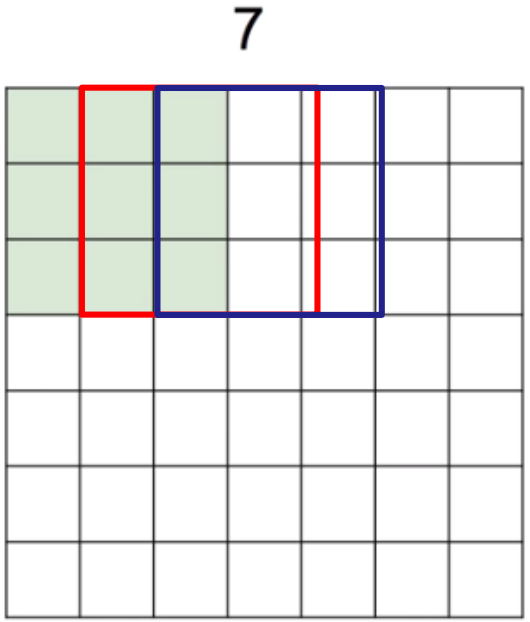
출력값



. 고양이에게 그림을 읽어 들이는 뉴런이 이미지의 형태에 따라 반응 실험 시작



. 필터: 이미지의 일부분 (5X5X3 filter) : 한점으로 표현
. ReLU(Wx+b)
. 같은 W를 가지는 필터로 다른 부분(이동)의 점 수집



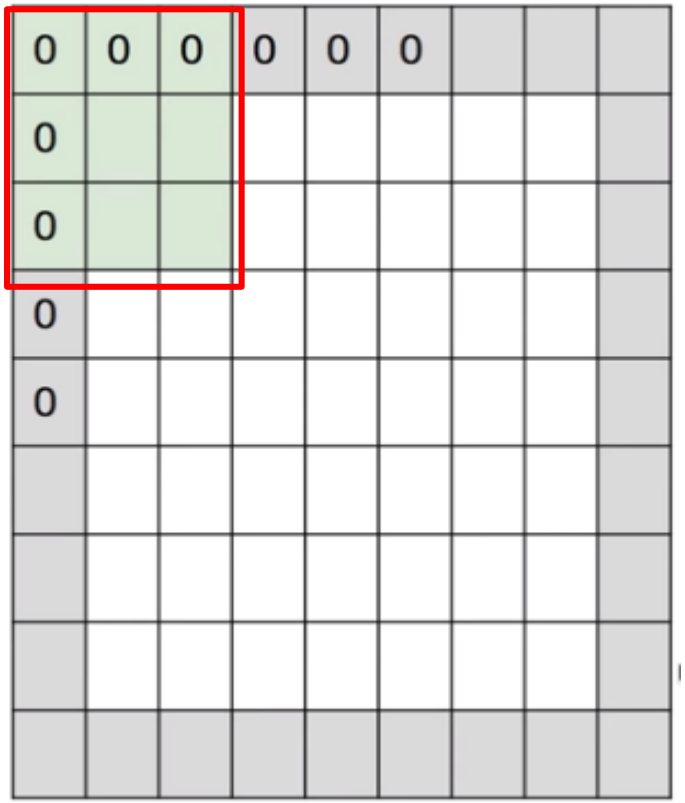
. 7 X 7 input (이미지 전체사이즈)
. 3 X 3 filter

. 5 X 5 output Stride(1)
. 3 X 3 output Stride(2)
=>(input-filter)/stride +1

예를 들어
Stride 1=> (7-3)/1 +1 =5
Stride 2=> (7-3)/2 +1 =3

CNN – padding

. 테두리에 0으로 입력(그림 사이즈 축소 우려, 모서리 표시하기 위함)



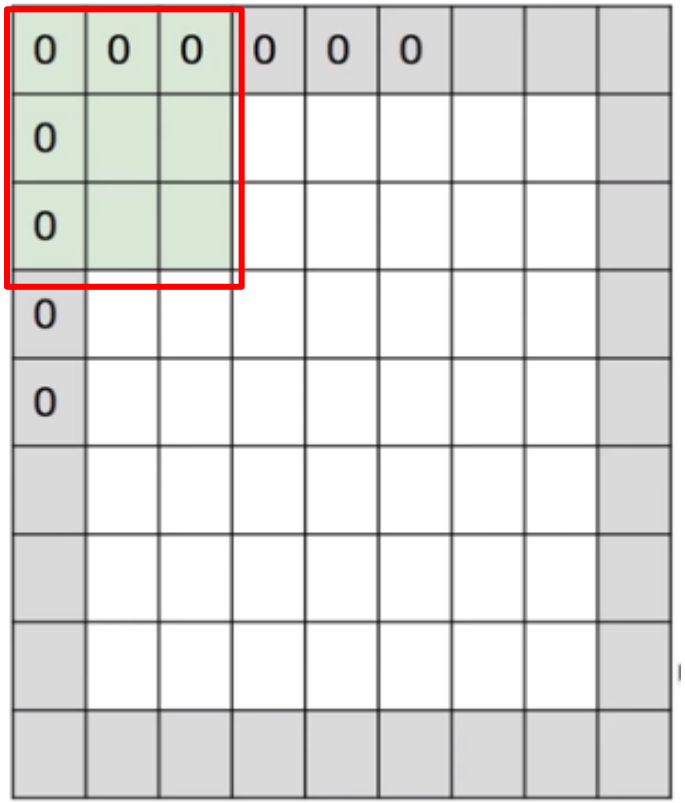
. 7 X 7 Input
. 3 X 3 filter
. 1 Stride
. 1 Padding

Output?
0으로 padding 9 X 9 (Stride 1)
(9 – 3) /1 +1 =7
(7 X 7 Output)

=> CONV layers는 paddin을 해서 input과 output사이즈가 같게 한다.

CNN – padding

. 테두리에 0으로 입력(그림 사이즈 축소 우려, 모서리 표시하기 위함)

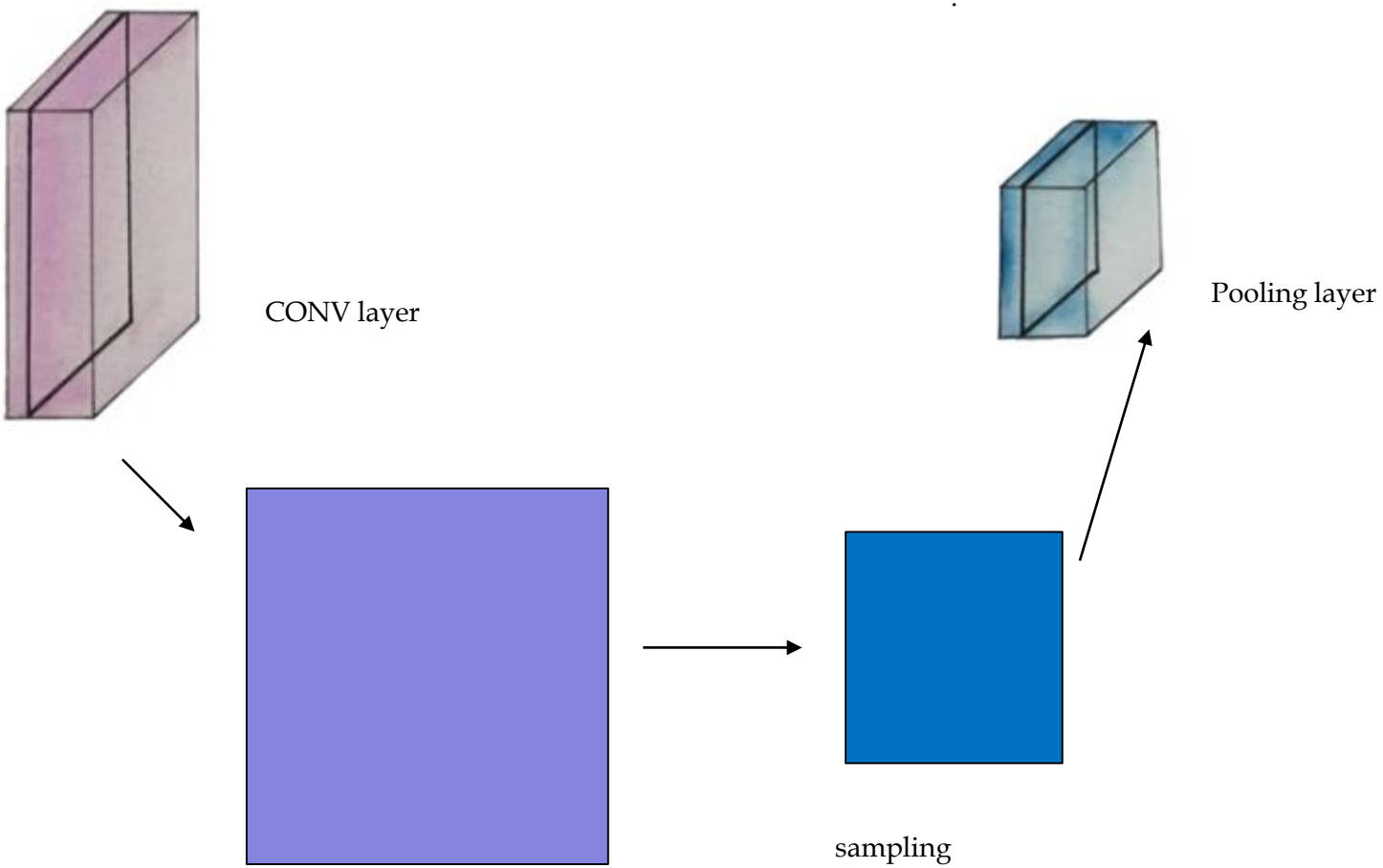


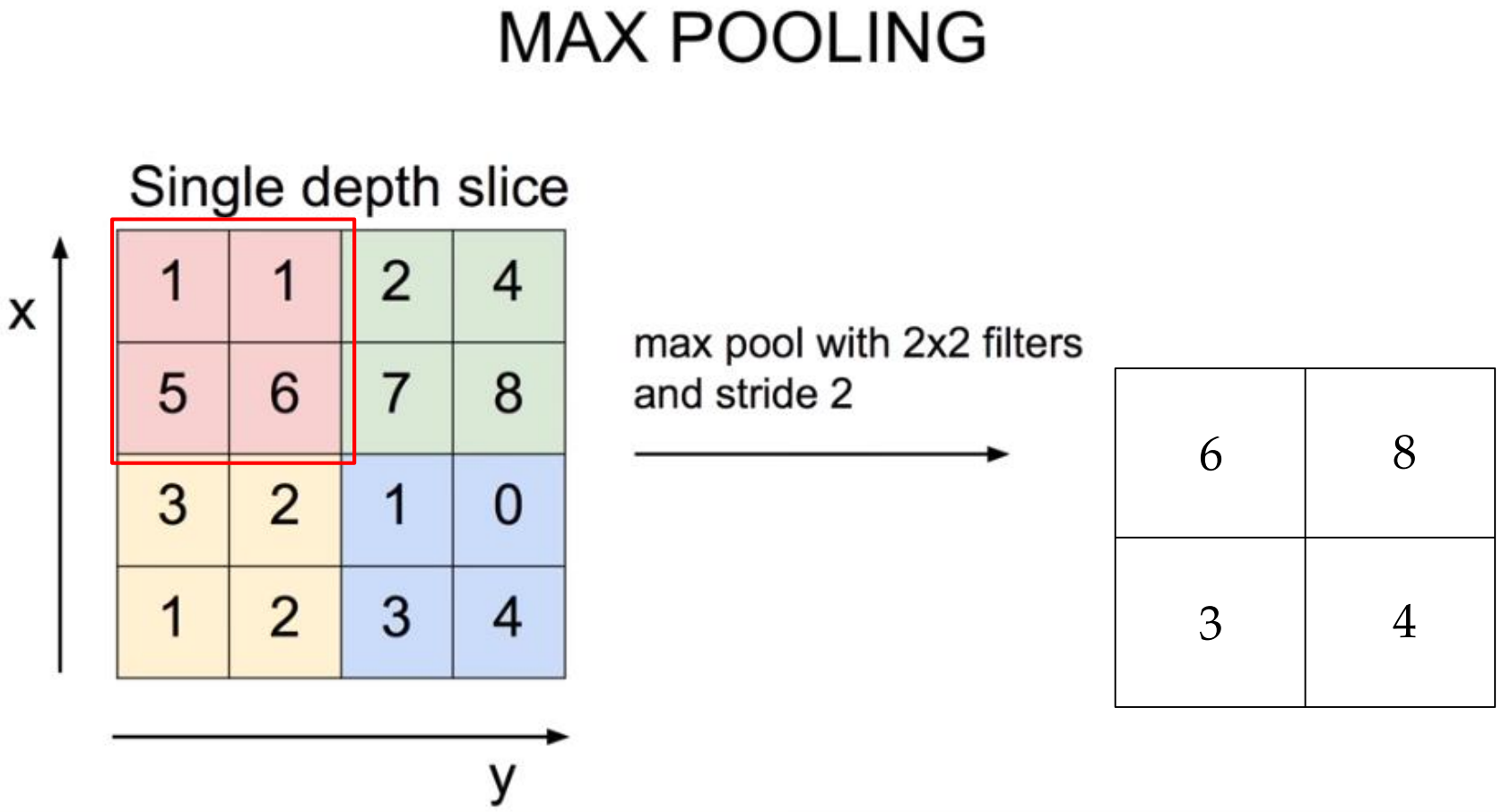
. 7 X 7 Input
. 3 X 3 filter
. 1 Stride
. 1 Padding

Output?
0으로 padding 9 X 9 (Stride 1)
(9 – 3) /1 +1 =7
(7 X 7 Output)

=> CONV layers는 paddin을 해서 input과 output사이즈가 같게 한다.

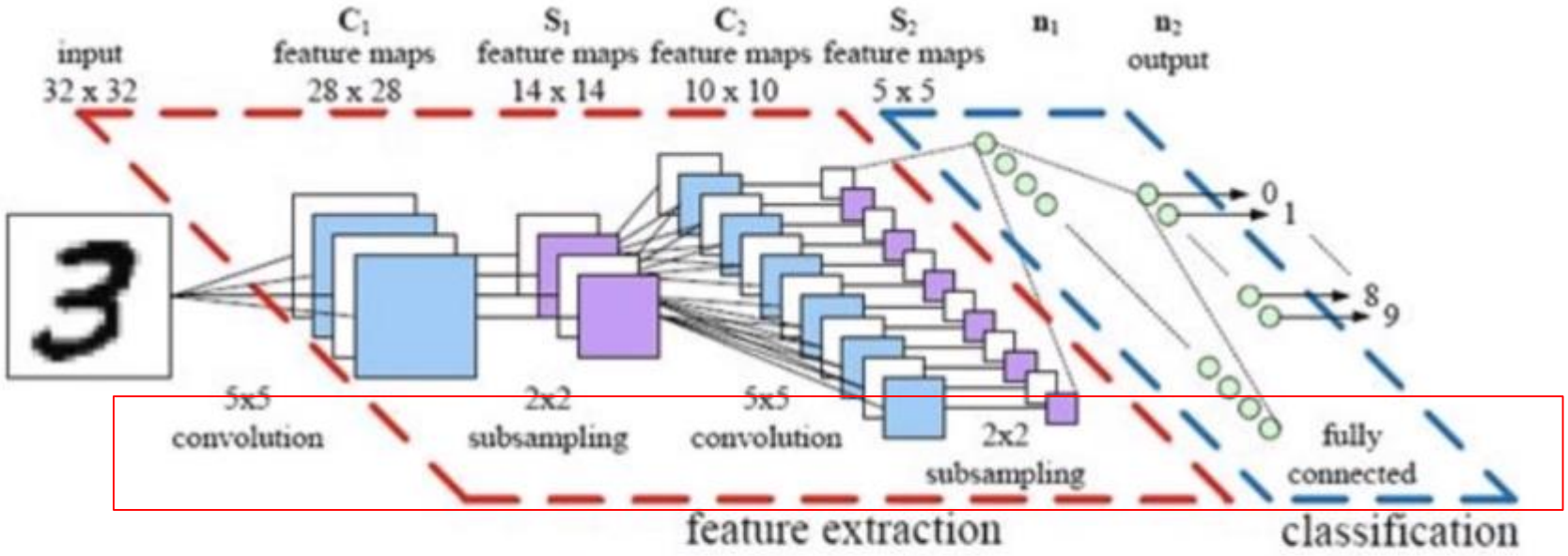
Pooling layer(sampling)




MAX POOLING
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4
Fei-Fei Li & Andrej Karpathy & Justin Johnson
Lecture 7 - 55
27 Jan 2016

CNN 예

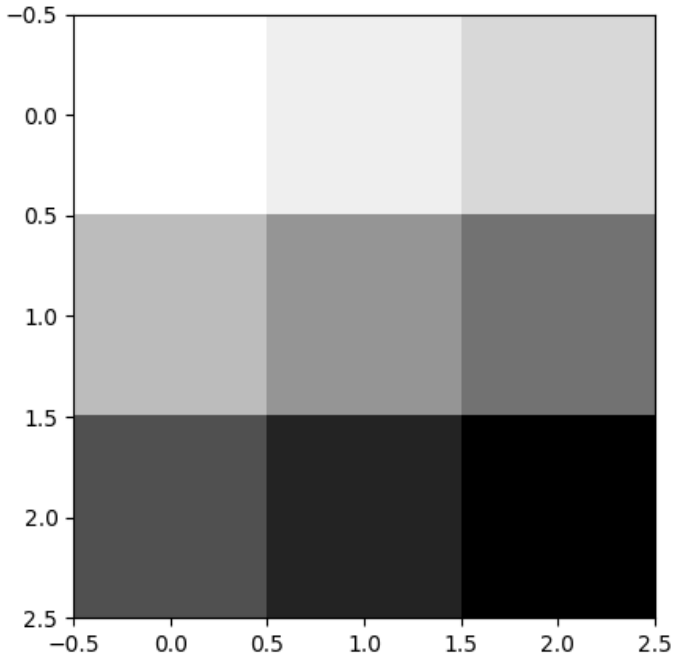
. ○



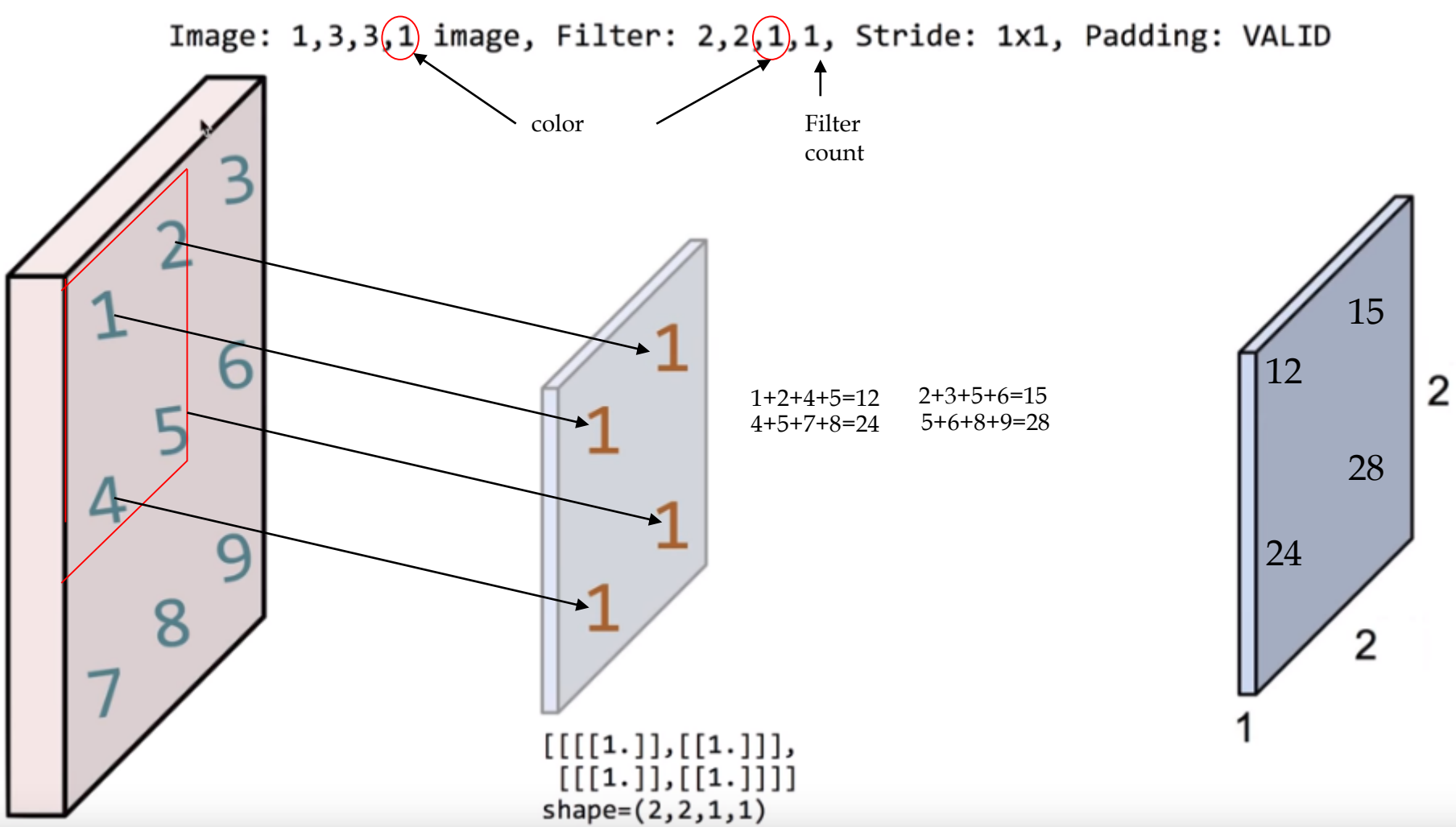
http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

CNN 예

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
sess = tf.InteractiveSession()
image = np.array([[[[1],[2],[3]],
                   [[4],[5],[6]],
                   [[7],[8],[9]]]], dtype=np.float32)
print(image.shape)
plt.imshow(image.reshape(3,3), cmap='Greys')
plt.show()
```

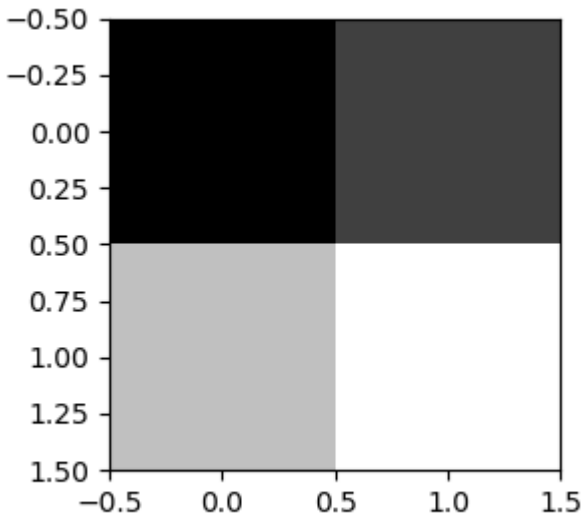


Simple convolution layer



CNN 합성곱

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
image = np.array([[[[1],[2],[3]],
                   [[4],[5],[6]],
                   [[7],[8],[9]]]], dtype=np.float32)
print("image.shape", image.shape)
weight = tf.constant([[[[1.]],[[1.]]],
                      [[[1.]],[[1.]]]])    #(2,2,1,1)
print("weight.shape", weight.shape)
sess = tf.InteractiveSession()
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='VALID')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(2,2))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(2,2), cmap='gray')
```



CNN 합성곱

. X는 입력의 4D텐서
. Filters 적용될 일련의 필터 4D텐서
. Strides는 4개 원소를 갖는 1D배열 => 가운데 2개는 수직, 수평 스트라이드
. Padding은 “valid”와 “same”중 하나를 지정
 - VALID: 합성곱층에 제로 패딩을 사용하지 않는다.
 - SAME : 합성곱층에 제로패팅 사용

Padding

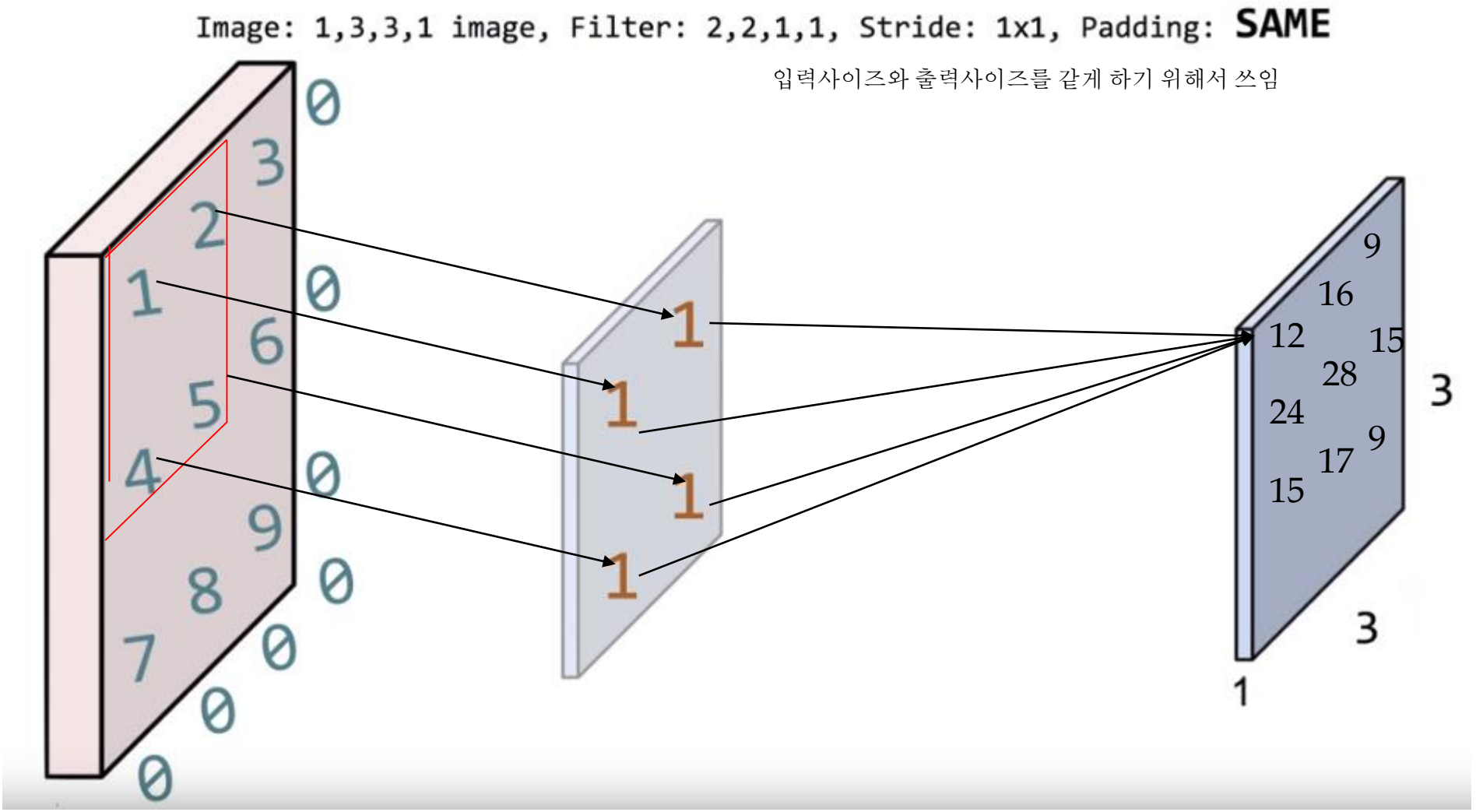
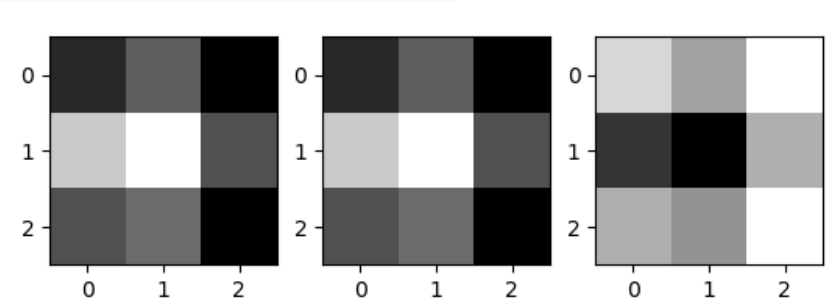


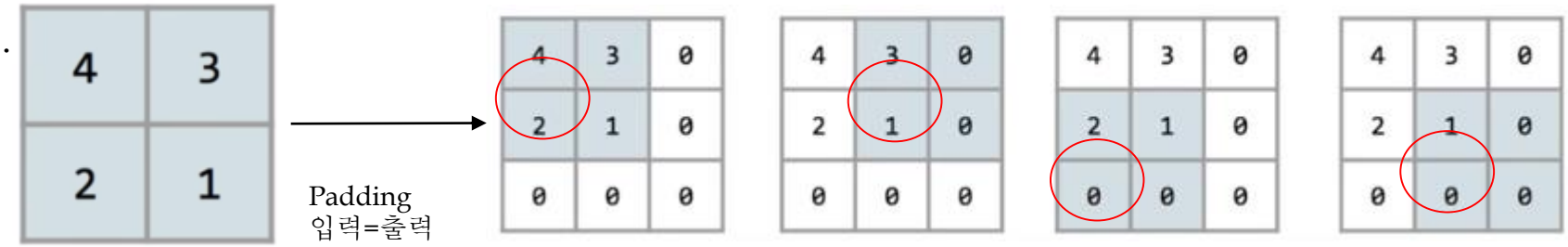
Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: SAME
입력사이즈와 출력사이즈를 같게 하기 위해서 쓰임
0
3
2
1
0
6
5
4
0
9
8
0
7
0
0
0
1
1
1
1
9
16
12
15
28
24
9
17
15
3
3
1

Padding 예

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
sess = tf.InteractiveSession()
image = np.array([[[[1], [2], [3]],
                  [[4], [5], [6]],
                  [[7], [8], [9]]]], dtype=np.float32)
print("image.shape", image.shape)
weight = tf.constant([[[[1.,10.,-1.]],[[1.,10.,-1.]]],    #(2,2,1,3)
                      [[[1.,10.,-1.]],[[1.,10.,-1.]]]])
print("weight.shape", weight.shape)
conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
conv2d_img = conv2d.eval()
print("conv2d_img.shape", conv2d_img.shape)
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(3,3))
    plt.subplot(1,3,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
plt.show()
```



Max Pooling(Sampling)



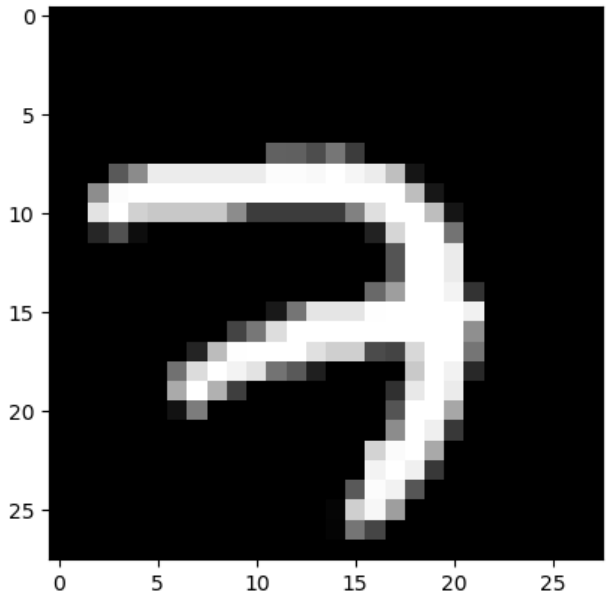

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
sess = tf.InteractiveSession()
image = np.array([[[[4],[3]],
                   [[2],[1]]]], dtype=np.float32)
pool = tf.nn.max_pool(image, ksize=[1, 2, 2, 1], strides=[1, 1, 1, 1], padding='SAME')
print(pool.shape)
print(pool.eval())
```

```
(1, 2, 2, 1)
[[[[4.]
   [3.]]

  [[2.]
   [1.]]]]
```

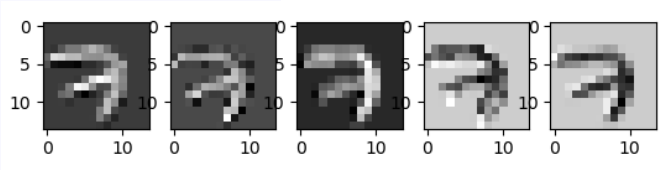
MNIST image loading

```
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
img = mnist.train.images[0].reshape(28,28)
plt.imshow(img, cmap='gray')
plt.show()
```



MNIST Convolution layer

```
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt
import numpy as np
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
import tensorflow as tf
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
img = mnist.train.images[0].reshape(28,28)
sess = tf.InteractiveSession()
img = img.reshape(-1,28,28,1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 5], stddev=0.01))
conv2d = tf.nn.conv2d(img, W1, strides=[1, 2, 2, 1], padding='SAME')
print(conv2d)
sess.run(tf.global_variables_initializer())
conv2d_img = conv2d.eval()
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(14,14), cmap='gray')
plt.show()
```



#3 X3 1개 color 5개 filter

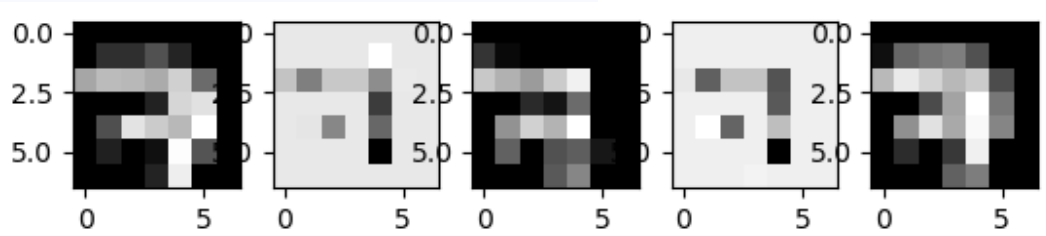
#2칸씩 이미지 이동(14 X14)

MNIST Max pooling

```
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt
import numpy as np
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
import tensorflow as tf
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
img = mnist.train.images[0].reshape(28,28)
sess = tf.InteractiveSession()
img = img.reshape(-1,28,28,1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 5], stddev=0.01))
conv2d = tf.nn.conv2d(img, W1, strides=[1, 2, 2, 1], padding='SAME')
print(conv2d)
sess.run(tf.global_variables_initializer())
conv2d_img = conv2d.eval()
conv2d_img = np.swapaxes(conv2d_img, 0, 3)
for i, one_img in enumerate(conv2d_img):
    plt.subplot(1,5,i+1), plt.imshow(one_img.reshape(14,14), cmap='gray')
plt.show()
```

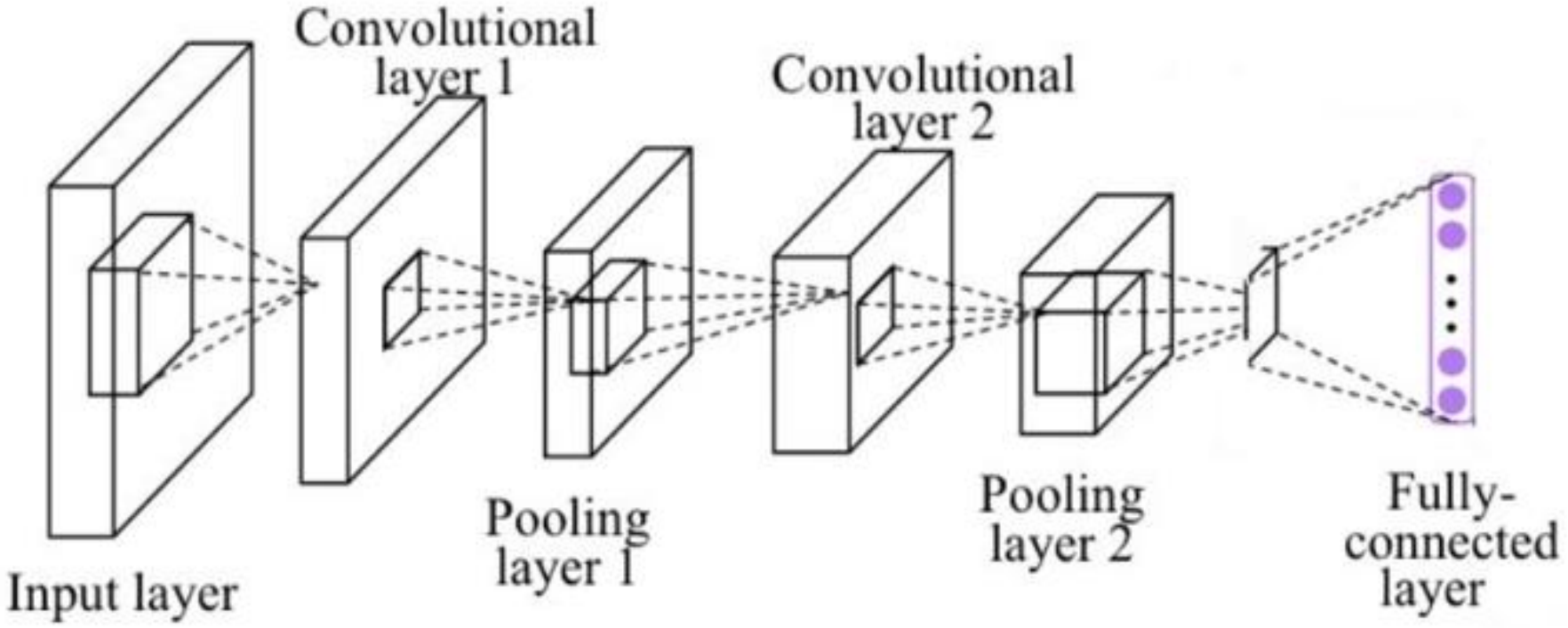
#3 X3 1개 color 5개 filter

#2칸씩 이미지 이동(7 X 7)



Simple CNN

.


Convolutional layer 1
Convolutional layer 2
Pooling layer 1
Pooling layer 2
Fully-connected layer
Input layer

Simple CNN-Mnist Conv layer1

```
# input placeholders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)
Y = tf.placeholder(tf.float32, [None, 10])
```

Filter의 수

```
# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
#    Conv     -> (?, 28, 28, 32)
#    Pool     -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
        strides=[1, 2, 2, 1], padding='SAME')
'''
```

```
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
'''
```

Simple CNN-Mnist Conv layer2

```
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
'''

# L2 ImgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
#    Conv      ->(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.reshape(L2, [-1, 7 * 7 * 64])
'''
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("Reshape_1:0", shape=(?, 3136), dtype=float32)
```

Filter의 수

Fully Connected(FC, Dense) layer

```
'''
Tensor("Conv2D_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("Relu_1:0", shape=(?, 14, 14, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 7, 7, 64), dtype=float32)
Tensor("Reshape_1:0", shape=(?, 3136), dtype=float32)
'''
L2 = tf.reshape(L2, [-1, 7 * 7 * 64])

# Final FC 7x7x64 inputs -> 10 outputs
W3 = tf.get_variable("W3", shape=[7 * 7 * 64, 10],
initializer=tf.contrib.layers.xavier_initializer())
b = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis,
labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

Training and Evaluation

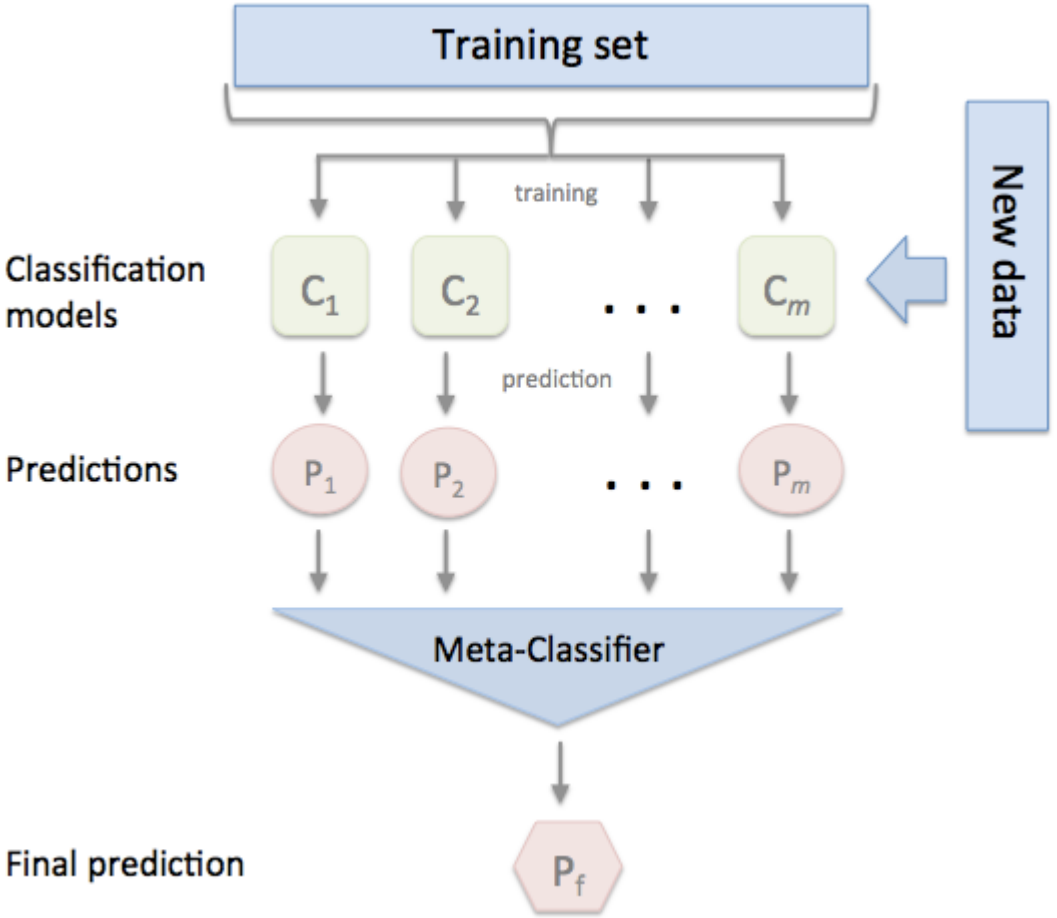
```
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
print('Learning stared. It takes sometime.')
for epoch in range(training_epochs):
  avg_cost = 0
  total_batch = int(mnist.train.num_examples / batch_size)
  for i in range(total_batch):
      batch_xs, batch_ys = mnist.train.next_batch(batch_size)
      feed_dict = {X: batch_xs, Y: batch_ys}
      c, _, = sess.run([cost, optimizer], feed_dict=feed_dict)
      avg_cost += c / total_batch
  print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y:
mnist.test.labels}))
```

Ensemble



여러 개의 모델을 학습 후 각각의 모델로 예측 후
예측한 결과를 조합하여 output
. Training set 같게 다르게 줄 수 있다.
. 전문가 100명에게 질문하여 효율적인 정답을 찾는 방법과 비슷하다.
. 2~4% 정확도 향상