20171664 소프트웨어학부 이범석

HW8 p269 ~ p321 In and Out Practice

실습 결과

1. 3클래스 분류 데이터와 동일한 데이터를 만들었습니다.

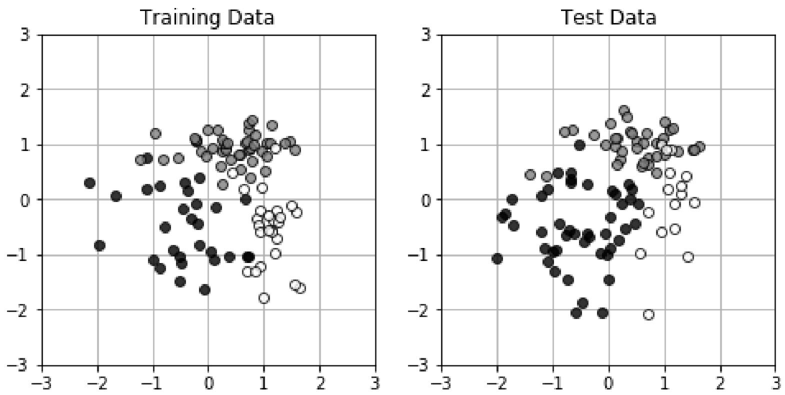
In [1]:
```
import numpy as np
np.random.seed(seed=1)
N = 200
K = 3
T = np.zeros((N,3), dtype=np.uint8)
X = np.zeros((N,2))
X_range0 = [-3, 3]
X_range1 = [-3, 3]
Mu = np.array([[-.5, -.5], [.5, 1.0], [1, -.5]])
Sig = np.array([[.7, .7], [.8, .3],[.3, .8]])
Pi = np.array([0.4, 0.8, 1])
for n in range(N):
    wk = np.random.rand()
    for k in range(K):
        if wk < Pi[k]:
            T[n, k] = 1
            break
    for k in range(2):
        X[n, k] = np.random.randn() * Sig[T[n, :] == 1, k] + \
        Mu[T[n, :] == 1, k]
```

2. 데이터를 훈련 데이터와 테스트 데이터로 나누고 데이터를 저장했습니다.

In [2]:
```
TestRatio = 0.5
X_n_training = int(N * TestRatio)
X_train = X[:X_n_training, :]
X_test = X[X_n_training:, :]
T_train = T[:X_n_training, :]
T_test = T[X_n_training:, :]

np.savez('class_data.npz', X_train = X_train, T_train=T_train,
         X_test = X_test, T_test = T_test,
         X_range0 = X_range0, X_range1 = X_range1)
```

3. 분할한 데이터를 출력했습니다.

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline

def Show_data(x, t):
    wk, n = t.shape
    c = [[0, 0, 0], [.5, .5, .5], [1, 1, 1]]
    for i in range(n):
        plt.plot(x[t[:, i] == 1, 0], x[t[:, i] == 1, 1],
                linestyle = 'none',
                marker='o', markeredgecolor='black',
                color=c[i], alpha=0.8)
    plt.grid(True)

plt.figure(1, figsize=(8, 3.7))
plt.subplot(1, 2, 1)
Show_data(X_train, T_train)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.title('Training Data')
plt.subplot(1, 2, 2)
Show_data(X_test, T_test)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.title('Test Data')
plt.show()
```



4. 시그모이드 함수를 정의하고 네트워크 프로그램을 작성해 중간층과 출력층의 정보를 출력했습니다.

In [4]:
```
def Sigmoid(x):
    y = 1 / (1+ np.exp(-x))
    return y

def FNN(wv, M, K, x):
    N, D = x.shape
    w = wv[:M*(D+1)]
    w = w.reshape(M, (D+1))
    v = wv[M*(D+1):]
    v = v.reshape((K, M+1))
    b = np.zeros((N, M+1))
    z = np.zeros((N, M+1))
    a = np.zeros((N,K))
    y = np.zeros((N,K))
    for n in range(N):
        for m in range(M):
            b[n, m] = np.dot(w[m, :], np.r_[x[n, :], 1])
            z[n, m] = Sigmoid(b[n, m])
        z[n, M] = 1
        wkz = 0
        for k in range(K):
            a[n, k] = np.dot(v[k, :], z[n, :])
            wkz = wkz + np.exp(a[n, k])
        for k in range(K):
            y[n, k] = np.exp(a[n, k]) / wkz
    return y, a, z, b

WV = np.ones(15)
M = 2
K = 3
FNN(WV, M, K, X_train[:2, :])
```

Out[4]:
```
(array([[0.33333333, 0.33333333, 0.33333333],
        [0.33333333, 0.33333333, 0.33333333]]),
 array([[2.6971835 , 2.6971835 , 2.6971835 ],
        [1.49172649, 1.49172649, 1.49172649]]),
 array([[0.84859175, 0.84859175, 1.        ],
        [0.24586324, 0.24586324, 1.        ]]),
 array([[ 1.72359839,  1.72359839,  0.        ],
        [-1.12079826, -1.12079826,  0.        ]]))
```

5. 평균 교차 엔트로피 오차를 구현했습니다.

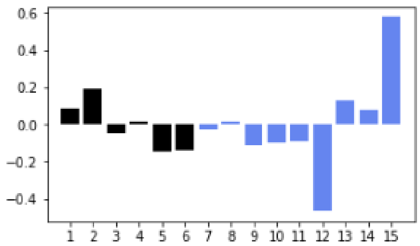
In [5]:
```
def CE_FNN(wv, M, K, x, t):
    N, D = x.shape
    y, a, z, b = FNN(wv, M, K, x)
    ce = -np.dot(np.log(y.reshape(-1)), t.reshape(-1)) / N
    return ce

WV = np.ones(15)
M = 2
K = 3
CE_FNN(WV, M, K, X_train[:2, :], T_train[:2, :])
```

Out[5]:
```
1.0986122886681098
```

6. 평균 교차 엔트로피 오차 함수의 수치 미분을 출력하는 함수를 만들고 출력했습니다.

In [6]:
```
def dCE_FNN_num(wv, M, K, x, t):
    epsilon = 0.001
    dwv = np.zeros_like(wv)
    for iwv in range(len(wv)):
        wv_modified = wv.copy()
        wv_modified[iwv] = wv[iwv] - epsilon
        mse1 = CE_FNN(wv_modified, M, K, x, t)
        wv_modified[iwv] = wv[iwv] + epsilon
        mse2 = CE_FNN(wv_modified, M, K, x, t)
        dwv[iwv] = (mse2 -mse1) / (2 * epsilon)
    return dwv

def Show_WV(wv, M):
    N = wv.shape[0]
    plt.bar(range(1, M*3 +1), wv[:M *3], align="center", color = 'black')
    plt.bar(range(M*3+1, N+1), wv[M*3:],
            align="center", color = 'cornflowerblue')
    plt.xticks(range(1, N+1))
    plt.xlim(0, N+1)


M = 2
K = 3
nWV = M*3 + K*(M+1)
np.random.seed(1)
WV = np.random.normal(0, 1, nWV)
dWV = dCE_FNN_num(WV, M, K, X_train[:2, :], T_train[:2, :])
print(dWV)
plt.figure(1, figsize=(5, 3))
Show_WV(dWV, M)
plt.show()
```

```
[ 0.0884813   0.19157999 -0.05139799  0.01281536 -0.14468029 -0.14242768
 -0.02992012  0.01351315 -0.11115648 -0.10104422 -0.09427964 -0.46855603
  0.13096434  0.08076649  0.57971252]
```



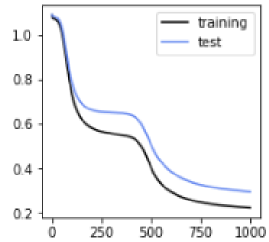
7. 수치 미분을 사용한 경사 하강법 함수를 구현하고 실행 시간을 표현했습니다.

In [7]:
```
import time

def Fit_FNN_num(wv_init, M, K, x_train, t_train, x_test, t_test, n, alpha):
    wvt = wv_init
    err_train = np.zeros(n)
    err_test = np.zeros(n)
    wv_hist = np.zeros((n, len(wv_init)))
    epsilon = 0.001
    for i in range(n):
        wvt = wvt -alpha*dCE_FNN_num(wvt, M, K, x_train, t_train)
        err_train[i] = CE_FNN(wvt, M, K, x_train, t_train)
        err_test[i] = CE_FNN(wvt, M, K, x_test, t_test)
        wv_hist[i, :] =wvt
    return wvt, wv_hist, err_train, err_test

startTime = time.time()
M = 2
K = 3
np.random.seed(1)
WV_init = np.random.normal(0, 0.01, M*3 + K*(M+1))
N_step = 1000
alpha = 0.5
WV, WV_hist, Err_train, Err_test = Fit_FNN_num(
WV_init, M, K, X_train, T_train, X_test, T_test, N_step, alpha)
calculation_time = time.time() -startTime
print("Calculation time:{0:.3f} sec".format(calculation_time))
```

```
Calculation time:209.817 sec
```

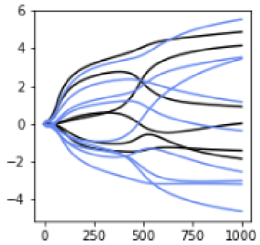
8. 오차 역전파법으로 구한 데이터를 그림으로 표현했습니다.

In [8]:
```
plt.figure(1, figsize =(3, 3))
plt.plot(Err_train, 'black', label = 'training')
plt.plot(Err_test, 'cornflowerblue', label = 'test')
plt.legend()
plt.show()
```



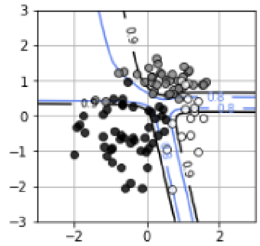
9. 가중치의 시간 변화를 그림으로 표현했습니다.

In [9]:
```
plt.figure(1, figsize = (3,3))
plt.plot(WV_hist[:, :M*3], 'black')
plt.plot(WV_hist[:, M*3:], 'cornflowerblue')
plt.show()
```



10. 수치 미분법에 의한 경사 하강법에서 얻은 클래스 간의 경계선을 그림으로 표현했습니다.

In [10]:
```
def show_FNN(wv, M, K):
    xn = 60
    x0 = np.linspace(X_range0[0], X_range0[1], xn)
    x1 = np.linspace(X_range1[0], X_range1[1], xn)
    xx0, xx1 = np.meshgrid(x0, x1)
    x = np.c_[np.reshape(xx0, xn * xn, 'F'), np.reshape(xx1, xn * xn, 'F')]
    y, a, z, b = FNN(wv, M, K, x)
    plt.figure(1, figsize=(4,4))
    for ic in range(K):
        f = y[:, ic]
        f = f.reshape(xn, xn)
        f = f.T
        cont = plt.contour(xx0, xx1, f, levels=[0.8, 0.9],
                           colors = ['cornflowerblue', 'black'])
        cont.clabel(fmt = '%1.1f', fontsize=9)
    plt.xlim(X_range0)
    plt.ylim(X_range1)


plt.figure(1, figsize=(3,3))
Show_data(X_test, T_test)
show_FNN(WV, M, K)
plt.show()
```

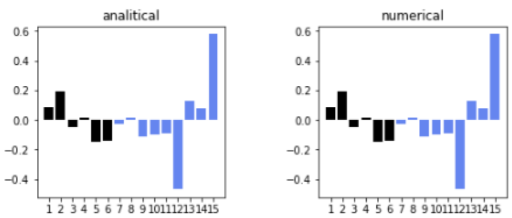


11. 오차 역전파법으로 $\partial E/\partial w$와 $\partial E/\partial v$를 구하는 프로그램을 만들고 그림으로 표현했습니다.

In [11]:

```python
def dCE_FNN(wv, M, K, x ,t):
    N, D = x.shape
    w = wv[:M * (D+1)]
    w = w.reshape(M, (D+1))
    v = wv[M*(D+1):]
    v = v.reshape((K, M+1))
    y, a, z, b = FNN(wv, M, K, x)
    dwv = np.zeros_like(wv)
    dw = np.zeros((M, D+1))
    dv = np.zeros((K, M+1))
    delta1 = np.zeros(M)
    delta2 = np.zeros(K)
    for n in range(N):
        for k in range(K):
            delta2[k] = (y[n, k] - t[n,k])
        for j in range(M):
            delta1[j] = z[n,j] * (1 - z[n, j])*np.dot(v[:, j], delta2)
        for k in range(K):
            dv[k, :] = dv[k, :] + delta2[k] * z[n, :] / N
        for j in range(M):
            dw[j,:] = dw[j, :] +delta1[j] * np.r_[x[n, :], 1] / N

    dwv = np.c_[dw.reshape((1, M*(D+1))), \
                dv.reshape((1, K*(M+1)))]
    dwv = dwv.reshape(-1)
    return dwv


def Show_dWV(wv, M):
    N = wv.shape[0]
    plt.bar(range(1, M*3 +1), wv[:M*3],
            align="center", color= 'black')
    plt.bar(range(M*3+1, N+1), wv[M*3:],
            align="center", color='cornflowerblue')
    plt.xticks(range(1, N+1))
    plt.xlim(0, N+1)


M = 2
K = 3
N = 2
nWV = M*3 + K*(M+1)
np.random.seed(1)
WV = np.random.normal(0, 1, nWV)

dWV_ana = dCE_FNN(WV, M, K, X_train[:N, :], T_train[:N, :])
print("analytical dWV")
print(dWV_ana)

dWV_num = dCE_FNN_num(WV, M, K, X_train[:N, :], T_train[:N, :])
print("numerical dWV")
print(dWV_num)

plt.figure(1, figsize=(8,3))
plt.subplots_adjust(wspace=0.5)
plt.subplot(1, 2, 1)
Show_dWV(dWV_ana, M)
plt.title('analitical')
plt.subplot(1, 2, 2)
Show_dWV(dWV_num, M)
plt.title('numerical')
plt.show()
```

```
analytical dWV
[ 0.08848131  0.19158    -0.051398    0.01281536 -0.14468029 -0.14242768
 -0.02992012  0.01351315 -0.11115649 -0.10104422 -0.09427964 -0.46855604
  0.13096434  0.08076649  0.57971253]
numerical dWV
[ 0.0884813   0.19157999 -0.05139799  0.01281536 -0.14468029 -0.14242768
 -0.02992012  0.01351315 -0.11115648 -0.10104422 -0.09427964 -0.46855603
  0.13096434  0.08076649  0.57971252]
```



12. 수치 미분으로 풀었던 분류 문제를 오차 역전파법으로 풀었습니다.

    수치 미분에 비해 훨씬 적은 실행 시간을 보였습니다.

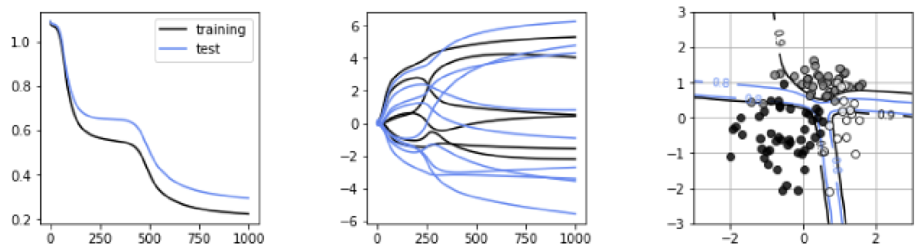
In [12]:
```
import time
def Fit_FNN(wv_init, M, K, x_train, t_train, x_test, t_test, n , alpha):
    wv = wv_init.copy()
    err_train = np.zeros(n)
    err_test = np.zeros(n)
    wv_hist = np.zeros((n, len(wv_init)))
    epsilon = 0.001
    for i in range(n):
        wv = wv- alpha*dCE_FNN(wv, M, K, x_train, t_train)
        err_train[i] = CE_FNN(wv, M, K, x_train, t_train)
        err_test[i] = CE_FNN(wv, M, K, x_test, t_test)
        wv_hist[i, :] = wv
    return wv, wv_hist, err_train, err_test

startTime = time.time()
M = 2
K = 3
np.random.seed(1)
WV_init = np.random.normal(0, 0.01, M*3 + K*(M+1))
N_step = 1000
alpha = 1
WV, WV_hist, ERR_train, ERR_test = Fit_FNN(
WV_init, M, K, X_train, T_train, X_test, T_test, N_step, alpha)
calculation_time = time.time() - startTime
print("Calculation time:{0:.3f} sec".format(calculation_time))
```

Calculation time:41.001 sec

13. 해석적 미분을 사용한 오차 역전파법의 실행 결과를 그림으로 표현했습니다.

수치 미분과 거의 같은 결과를 얻었습니다.

In [13]:
```
plt.figure(1, figsize=(12,3))
plt.subplots_adjust(wspace=0.5)
plt.subplot(1,3,1)
plt.plot(Err_train, 'black', label = 'training')
plt.plot(Err_test, 'cornflowerblue', label = 'test')
plt.legend()

plt.subplot(1,3,2)
plt.plot(WV_hist[:, :M*3], 'black')
plt.plot(WV_hist[:, M*3:], 'cornflowerblue')
plt.subplot(1, 3,3)
Show_data(X_test, T_test)
M = 2
K = 3
show_FNN(WV, M, K)
plt.show()
```

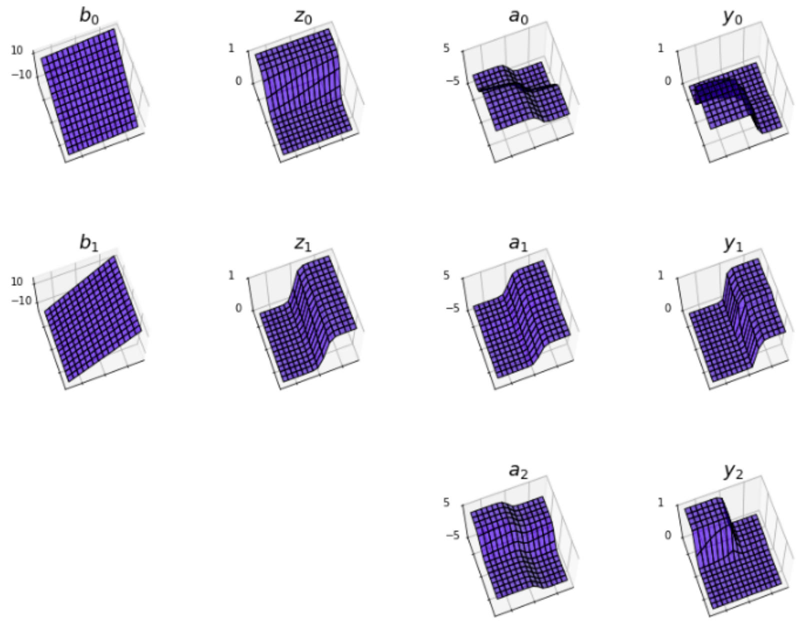


14. 각 뉴런의 특성을 그림으로 표현했습니다.

x0, x1의 쌍이 입력된 경우는 각 변수의 값을 나타냈습니다.

중간층 입력 총합은 선형의 합이므로 입출력 맵은 평면입니다.

```
In [14]: from mpl_toolkits.mplot3d import Axes3D

def show_activation3d(ax, v, v_ticks, title_str):
    f = v.copy()
    f = f.reshape(xn, xn)
    f = f.T
    ax.plot_surface(xx0, xx1, f, color= 'blue', edgecolor = 'black',
                rstride = 1, cstride = 1, alpha =0.5)
    ax.view_init(70, -110)
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.set_zticks(v_ticks)
    ax.set_title(title_str, fontsize=18)

M = 2
K = 3
xn = 15
x0 = np.linspace(X_range0[0], X_range0[1], xn)
x1 = np.linspace(X_range1[0], X_range1[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
x = np.c_[np.reshape(xx0, xn * xn, 'A'), np.reshape(xx1, xn*xn,'A')]
y, a, z, b = FNN(WV, M, K, x)

fig = plt.figure(1, figsize = (12, 9))
plt.subplots_adjust(left=0.075, bottom = 0.05, right = 0.95,
                top = 0.95, wspace = 0.4, hspace = 0.4)

for m in range(M):
    ax = fig.add_subplot(3, 4, 1 + m * 4, projection = '3d')
    show_activation3d(ax, b[:,m],[-10,10], '$b_{0:d}$'.format(m))
    ax = fig.add_subplot(3, 4, 2 + m * 4, projection = '3d')
    show_activation3d(ax, z[:, m], [0, 1], '$z_{0:d}$'.format(m))

for k in range(K):
    ax = fig.add_subplot(3, 4, 3+ k * 4, projection = '3d')
    show_activation3d(ax, a[:, k], [-5, 5], '$a_{0:d}$'.format(k))
    ax = fig.add_subplot(3, 4, 4+k*4, projection='3d')
    show_activation3d(ax, y[:,k],[0,1], '$y_{0:d}$'.format(k))

plt.show()
```



15. 메모리를 초기화했습니다.

```
In [15]: %reset
```

Once deleted, variables cannot be recovered. Proceed (y/[n])? y

16. 필요한 라이브러리를 import하고, 저장된 데이터를 불러왔습니다.

```
In [16]: import numpy as np
import matplotlib.pyplot as plt
import time
np.random.seed(1)
import keras.optimizers
from keras.models import Sequential
from keras.layers.core import Dense, Activation

outfile = np.load('class_data.npz')
X_train = outfile['X_train']
T_train = outfile['T_train']
X_test = outfile['X_test']
T_test = outfile['T_test']
X_range0 = outfile['X_range0']
X_range1 = outfile['X_range1']
```

Using TensorFlow backend.

17. 데이터를 그림으로 그리하는 함수를 재정의했습니다.

```
In [17]: def Show_data(x, t):
             wk, n = t.shape
             c = [[0, 0, 0], [.5, .5, .5], [1, 1, 1]]
             for i in range(n):
                 plt.plot(x[t[:, i] == 1, 0], x[t[:, i] == 1, 1],
                     linestyle = 'none', marker='o',
                     markeredgecolor = 'black',
                     color = c[i], alpha = 0.8)
             plt.grid(True)
```

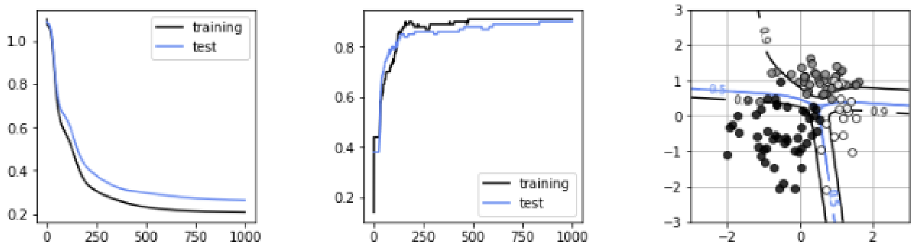
18. 케라스를 사용하여 2층 피드백 신경망 모델을 만들고 학습시켰습니다.

```
In [18]: np.random.seed(1)

         model = Sequential()
         model.add(Dense(2, input_dim = 2, activation = 'sigmoid',
                kernel_initializer = 'uniform'))
         model.add(Dense(3, activation = 'softmax',
                kernel_initializer = 'uniform'))
         sgd = keras.optimizers.SGD(lr = 1, momentum = 0.0,
                          decay = 0.0, nesterov = False)
         model.compile(optimizer = sgd, loss = 'categorical_crossentropy',
               metrics = ['accuracy'])

         startTime = time.time()
         history = model.fit(X_train, T_train, epochs= 1000, batch_size = 100,
                     verbose = 0, validation_data = (X_test, T_test))

         score = model.evaluate(X_test, T_test, verbose = 0)
         print('cross entropy {0:3.2f}, accuracy {1:3.2f}'.format(score[0], score[1]))
         calculation_time = time.time() - startTime
         print("Calculation time:{0:.3f} sec".format(calculation_time))
```

```
cross entropy 0.26, accuracy 0.90
Calculation time:7.282 sec
```

19. 케라스 사용의 흐름의 과정과 그 결과를 그래프로 표시했습니다.

```
In [19]: plt.figure(1, figsize = (12, 3))
         plt.subplots_adjust(wspace = 0.5)

         plt.subplot(1, 3, 1)
         plt.plot(history.history['loss'], 'black', label = 'training')
         plt.plot(history.history['val_loss'], 'cornflowerblue', label = 'test')
         plt.legend()

         plt.subplot(1, 3, 2)
         plt.plot(history.history['accuracy'], 'black', label = 'training')
         plt.plot(history.history['val_accuracy'], 'cornflowerblue', label='test')
         plt.legend()

         plt.subplot(1, 3, 3)
         Show_data(X_test, T_test)
         xn = 60
         x0 = np.linspace(X_range0[0], X_range0[1], xn)
         x1 = np.linspace(X_range1[0], X_range1[1], xn)

         xx0, xx1 = np.meshgrid(x0, x1)
         x = np.c_[np.reshape(xx0, xn*xn, 'F'), np.reshape(xx1, xn * xn , 'F')]
         y = model.predict(x)
         K = 3
         for ic in range(K):
             f = y[:, ic]
             f = f.reshape(xn, xn)
             f = f.T
             cont = plt.contour(xx0, xx1, f, levels = [0.5, 0.9], colors=[
                 'cornflowerblue', 'black'])
             cont.clabel(fmt='%1.1f', fontsize = 9)
             plt.xlim(X_range0)
             plt.ylim(X_range1)
         plt.show()
```



수행 소감

신경망 프로그램 구현과 라이브러리 이용으로 데이터 학습을 할 수 있다는 사실 자체로 학습 효과가 있었습니다.

더 복잡한 프로그램을 구현할 때는 효과적으로 원하는 근사 데이터를 얻을 수 있을 것 같습니다.

이범석

국민대학교 소프트웨어학부, 20171664

Mobile 010-6401-6042

qpwoeiru6486@gmail.com

ijkoo16@kookmin.ac.kr