**개별 프로젝트 과제 (중간시험 대체)**

제출기한 : 21년 12월 22일 (수) 23:59분

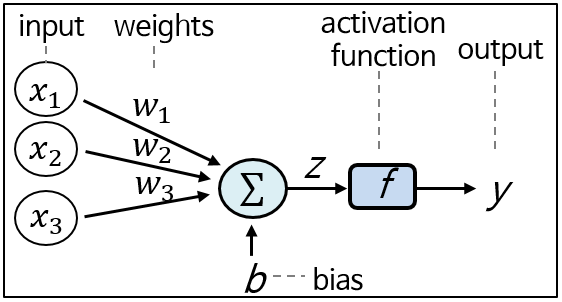
작성한 프로그램 코드 및 실행 결과물을 포함하여 word로 작성해주세요. (**레포트 반드시 첨부, 누락시 0점**)  
**작성한 Programming 코드 (.py 또는 .ipynb) 를 꼭 함께 제출 하세요.  
제출 code에 단순 warning이 아닌 수정 불가능한 error 발생 시 채점 되지 않습니다.**본인이 작성한 코드에 대해서는 주석을 통해 설명을 달아주세요.  
파일이 2개 이상 일 경우 하나의 .zip파일로 업로드 하세요.   
**＃ Copy 엄금, 적발 시 0점, 딜레이는 받지 않습니다 (기간 엄수)**

**[참고 1]**데이터 로드 및 평면 출력

|  |
| --- |
| %matplotlib notebook  from mpl\_toolkits.mplot3d import Axes3D  import matplotlib.pyplot as plt  from matplotlib import cm  import numpy as np  import pandas as pd  data = pd.read\_csv("p1\_training\_data.csv") # 데이터 읽기  np\_data = np.array(data)  fig = plt.figure()  ax = fig.add\_subplot(111, projection='3d')  ## positive samples  x\_1 = np\_data[0:50,0]  y\_1 = np\_data[0:50,1]  z\_1 = np\_data[0:50,2]  ## negative samples  x\_0 = np\_data[50:,0]  y\_0 = np\_data[50:,1]  z\_0 = np\_data[50:,2]  ## Generation 1의 fittest gene  w1 = 0.28645574  w2 = -0.43628723  w3 = 0.30481866  b = -14.39337271  ax.plot(x\_1, y\_1, z\_1, linestyle="none", marker="o", mfc="none", markeredgecolor="b") #샘플 출력  ax.plot(x\_0, y\_0, z\_0, linestyle="none", marker="o", mfc="none", markeredgecolor="r") #샘플 출력  X = np.arange(0, 2, 0.1)\*100  Y = np.arange(0, 2, 0.1)\*100  X, Y = np.meshgrid(X, Y)  Z = (-float(w1)/w3 \* X) + (-float(w2)/w3 \* Y) - float(b)/w3 # 평면의 방정식  ax.plot\_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.4, cmap=cm.Blues) # 평면 출력  plt.show() |

**Genetic Algorithm**

**1 유전 알고리즘을 통해 주어진 데이터를 선형 분류하는 single-layer neural network를 학습하라. [전체: 50점]**아래 세부 사항을 확인하여 문제를 해결 하시오.

* 학습 데이터: p1\_training\_data.csv (100개x4차원데이터, 1~3차원 데이터는 입력데이터 (x1,x2,x3), 4번째 데이터는 클래스 y(1 또는 0))
* 학습 파라미터: **w** (3차원), b (1차원) [아래 그림 참조]  
  
* Activation function: Sigmoid function 을 사용할 것.
* Population수는 100 이하로 설정하라.
* 위 조건 이외의 유전알고리즘 동작을 위한 모든 설계 조건은 자율적으로 설계 할 것 (**단, 레포트에 설계 내용을 모두 기입**)
* (fitness계산방법, Selection방법, crossover방법, mutation방법, 유전자(학습 파라미터)초기화 방법, 알고리즘 종료 조건 등)
  1. 1세대 유전자를 초기화 하라. 어떤 방식으로 유전자를 초기화 하였는가? Population은 얼마로 세팅하였는가? [5점]  
     (hint. 각각의 weight는 -1~1사이로, bias는 -100~100 사이로 초기화 한다.)

**Python 의 random 모듈을 사용해, 초기화 하였습니다. random\_weight\_list(), random\_bias\_list() 를 만들어서 초기화 했습니다. Population은 90으로 설정하였습니다.**

from mpl\_toolkits.mplot3d import Axes3D  
import matplotlib.pyplot as plt  
from matplotlib import cm  
import numpy as np  
import pandas as pd  
import random  
  
# 첫 세대의 가중치를 랜덤으로 생성하고, 배열로 반환.  
def random\_weight\_list():  
 weight\_list = list() # 리스트 선언  
 for i in range(70):  
 weight\_list.append(random.uniform(-1,1)) # uniform 함수 이용해 랜덤으로 값 추가.  
 return weight\_list  
  
# 첫 세대의 bias 를 랜덤으로 생성하고, 배열로 반환.  
def random\_bias\_list():  
 bias\_list = list()  
 for i in range(70):  
 bias\_list.append(random.uniform(-100,100)) # uniform 함수 이용해 랜덤으로 값 추가.  
 return bias\_list  
  
data = pd.read\_csv("p1\_training\_data.csv") # 데이터 읽기  
np\_data = np.array(data) # np Array 로 변경  
  
fig = plt.figure() # 새로운 figure 생성  
ax = fig.add\_subplot(111, projection='3d')  
  
## positive samples  
x\_1 = np\_data[0:50,0]  
y\_1 = np\_data[0:50,1]  
z\_1 = np\_data[0:50,2]  
postive\_label\_sample = np\_data[0:50,3]  
  
## negative samples  
x\_0 = np\_data[50:,0]  
y\_0 = np\_data[50:,1]  
z\_0 = np\_data[50:,2]  
  
## Generation 1의 fittest gene  
w1 = 0.28645574  
w2 = -0.43628723  
w3 = 0.30481866  
b = -14.39337271  
  
# numpy 배열로 변경  
w1\_population = np.array(random\_weight\_list())  
w2\_population = np.array(random\_weight\_list())  
w3\_population = np.array(random\_weight\_list())  
b\_population = np.array(random\_bias\_list())  
  
# population 출력  
print("--- w1 ---")  
print(w1\_population)  
print("--- w2 ---")  
print(w2\_population)  
print("--- w3 ---")  
print(w3\_population)  
print("--- b ----")  
print(b\_population)

실행결과

Text

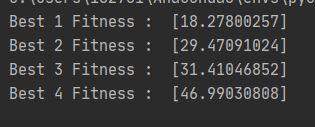
Description automatically generated

이는 무작위로 추출한 것이므로, **프로그램을 실행할 때마다 바뀝니다**.

* 1. 각 유전자에 대한 fitness 계산법을 설계하고 가장 fitness가 높은 유전자들을 선별하라. [10점]  
     fitness 계산은 어떻게 하였는가? -> **아래에 답 기재함.**1세대에서 가장 fitness가 높은 상위 4개의 유전자에 대해 fitness score를 적고, 분류 평면을 도시화 하라 [실행 예 참고]

|  |  |  |  |
| --- | --- | --- | --- |
| Ranking1, Fitness score: 18.28 | Ranking2, Fitness score: 29.47 | Ranking3, Fitness score: 31.41 | Ranking4, Fitness score: 46.99 |

표1. [실행 예] 1세대 유전자 (Rank1~4)



fitness score 구하는 방법 : (실제 label 값 - 모델이 구한 label 값)2 의 합

fitness score 가 작을수록, 분류가 잘된 모델입니다.

선택된 유전자들에 대해 Crossover와 mutation을 수행하여 2세대 유전자들을 생성하시오. 이때 생성된 자식세대의 population 수는   
1세대와 동일하게 설계 하시오.  
  
Crossover는 어떻게 수행하였는가? 상세히 작성하시오. [10점]  
교배방식은 다음과 같습니다. population number를 90으로 설정하고, 각각 개체마다 모든 개체와 교배할 수 있도록 합니다. 예를 들어, 개체가 3마리라면 (1,2) (1,3) (2,3) 이렇게 3가지 경우가 나옵니다.

따라서 Combination을 이용해서 , selection number를 10으로 설정한다면, 10C2 = 45로 Combination하는 방법을 두번 고려한다면, 해당 population number의 다음 세대의 개체수를 얻어낼 수 있습니다.

앞에서 든 예시를 살펴보며 제 프로젝트에 교배방식을 설명하겠습니다. 예시로 (1,2), (1,3) , (2,3) 이라했는데, 앞에 있는 개체를 **‘부’**, 뒤에있는 개체를 \*\*‘모’\*\*라고 칭하겠습니다.

상위 개체를 포함해서, 소수부 뒷 5자리는 분류하는 평면의 이동에 드라마틱한 영향을 주지 않습니다. 따라서, 소수부 앞자리를 전달하기 위해 고민을 한 결과, 다음 두 방법으로 교배했습니다.

어떻게 보면 비트를 섞는 것도 좋은 방법이긴 하겠지만, 좋은 부모끼리의 형질이 균등하게 분배되면 더 좋은 세대를 낳지 않을까의 생각해서 다음과 같이 생각했습니다.

방법 1)

가중치 : 부의 앞자리 유전자 + 모의 뒷자리 유전자

bias : 부모의 평균 값

방법 2)

가중치 : 부모의 앞자리 유전자의 평균값

bias : 부모의 평균 값

* 1. Mutation은 어떻게 수행하였는가? 상세히 작성하시오. [10점]

다음 개체로의 다양성을 부여하기 위해 변이가 필요합니다. 따라서 개체당 일어날 확률은 1프로로 지정하였습니다.

변이는 다음과 같이 이루어집니다.

w1 = w1 + 0.2

w2 = w2 - 0.2

w3 = w3 / 2

b = b - 2

아래에 Crossover, mutation 에 대한 코드를 적습니다.

from mpl\_toolkits.mplot3d import Axes3D  
import matplotlib.pyplot as plt  
from matplotlib import cm  
import numpy as np  
import pandas as pd  
import random  
import math  
  
max\_generation\_number = 40 # 최대로 학습할 세대  
population\_number = 90 # 세대 당 개체 수  
select\_number = 10 # 선택할 개체의 갯수  
# 첫 세대의 가중치를 랜덤으로 생성하고, 배열로 반환.  
def random\_weight\_list():  
 weight\_list = list() # 리스트 선언  
 for i in range(population\_number):  
 weight\_list.append(random.uniform(-1,1)) # uniform 함수 이용해 랜덤으로 값 추가.  
 return weight\_list  
  
# 첫 세대의 bias 를 랜덤으로 생성하고, 배열로 반환.  
def random\_bias\_list():  
 bias\_list = list()  
 for i in range(population\_number):  
 bias\_list.append(random.uniform(-100,100)) # uniform 함수 이용해 랜덤으로 값 추가.  
 return bias\_list  
  
# sigmoid 함수  
def sigmoid(x):  
 # math.exp(x) = e^x 와 같다.  
 y = 1 / (1 + math.exp(-x))  
 return y  
  
# zeta 함수 값을 sigmoid 처리한 label 들을 반환하는 함수  
def sigmoid\_list(zeta\_list):  
 y = np.zeros((len(zeta\_list), 1)) # 행렬 생성  
 # zeta\_list 의 개별 원소를 sigmoid 함수에 넣어서, y에 값 넣기  
 for i in range(0, len(zeta\_list)):  
 y[i] = sigmoid(zeta\_list[i])  
 # y 반환  
 return y  
  
# zeta 함수  
def zeta(w1, w2, w3, x1, x2, x3, b):  
 z = np.zeros((100, 1))  
  
 for i in range(100):  
 z[i] = w1\*x1[i] + w2\*x2[i] + w3\*x3[i] + b  
 return z  
  
# 유전자 교배  
def crossover(w1\_list, w2\_list, w3\_list, b\_list):  
 # 다음 세대로 전달될 w1, w2, w3, b  
 w1\_next\_generation = list()  
 w2\_next\_generation = list()  
 w3\_next\_generation = list()  
 b\_next\_generation = list()  
  
 # 유전자를 임시로 저장할 리스트  
 w1\_gene\_list = list()  
 w2\_gene\_list = list()  
 w3\_gene\_list = list()  
 b\_gene\_list = list()  
 # 유전자 생성. 가중치 유전자는 각각 앞의 소수 5자리와 뒷소수 5자리로 나누어진다. bias 는 부모의 평균값을 가져옴.  
 # 자세한 사항은 레포트에 기재  
 for i in range(select\_number):  
 w1\_gene\_list.append(weight\_get\_gene(w1\_list[i]))  
 w2\_gene\_list.append(weight\_get\_gene(w2\_list[i]))  
 w3\_gene\_list.append(weight\_get\_gene(w3\_list[i]))  
 b\_gene\_list.append(b\_list[i])  
  
 # 유전자 교배. 앞자리 + 뒷자리, b는 부모의 평균으로 구해온다.  
 for i in range(select\_number):  
 for j in range(i+1, select\_number):  
 w1\_next\_generation.append(w1\_gene\_list[i][0] + w1\_gene\_list[j][1])  
 w2\_next\_generation.append(w2\_gene\_list[i][0] + w2\_gene\_list[j][1])  
 w3\_next\_generation.append(w3\_gene\_list[i][0] + w3\_gene\_list[j][1])  
 b\_next\_generation.append((b\_gene\_list[i] + b\_gene\_list[j]) / 2)  
  
  
 # 유전자 교배. 부모의 앞자리를 더한 평균 값. b도 마찬가지로 부모의 평균으로 구해온다.  
 for i in range(select\_number):  
 for j in range(i+1, select\_number):  
 w1\_next\_generation.append((w1\_gene\_list[i][0] + w1\_gene\_list[j][0])/2)  
 w2\_next\_generation.append((w2\_gene\_list[i][0] + w2\_gene\_list[j][0])/2)  
 w3\_next\_generation.append((w3\_gene\_list[i][0] + w3\_gene\_list[j][0])/2)  
 b\_next\_generation.append((b\_gene\_list[i] + b\_gene\_list[j]) / 2)  
  
 # 혹시나 세대수를 변경할 시, 필요 이상의 유전자가 만들어지면, 임의로 제거한다.  
 for i in range(population\_number - len(w1\_next\_generation)):  
 kill\_index = int(random.random() % 100)  
 del w1\_next\_generation[kill\_index]  
 del w2\_next\_generation[kill\_index]  
 del w3\_next\_generation[kill\_index]  
 del b\_next\_generation[kill\_index]  
  
 # 리스트형으로 반환  
 return [w1\_next\_generation, w2\_next\_generation, w3\_next\_generation, b\_next\_generation]  
  
# 가중치 유전자를 tuple로 생성후 반환한다.  
# 유전자 생성은 a 는 소수점 5자리까지, b 는 소수점 5자리 뒤부터 이다.  
def weight\_get\_gene(source):  
 gene\_a = ((source\*100000) // 1) / 100000  
 gene\_b = source % 0.00001  
 return gene\_a, gene\_b  
  
# 출력해주는 함수  
def visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1, w2, w3, b, ax):  
 ax.plot(x\_0, y\_0, z\_0, linestyle="none", marker="o", mfc="none", markeredgecolor="r") # 샘플 출력  
 ax.plot(x\_1, y\_1, z\_1, linestyle="none", marker="o", mfc="none", markeredgecolor="b") # 샘플 출력  
  
 X = np.arange(0, 2, 0.1) \* 100  
 Y = np.arange(0, 2, 0.1) \* 100  
 X, Y = np.meshgrid(X, Y)  
  
 Z = (-float(w1) / w3 \* X) + (-float(w2) / w3 \* Y) - float(b) / w3 # 평면의 방정식  
  
 ax.plot\_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.4, cmap=cm.Blues) # 평면 출력  
  
  
data = pd.read\_csv("p1\_training\_data.csv") # 데이터 읽기  
np\_data = np.array(data) # np Array 로 변경  
  
fig = plt.figure() # 새로운 figure 생성  
fig2 = plt.figure()  
fig3 = plt.figure()  
fig4 = plt.figure()  
  
ax = fig.add\_subplot(111, projection='3d')  
ax2 = fig2.add\_subplot(111, projection='3d')  
ax3 = fig3.add\_subplot(111, projection='3d')  
ax4 = fig4.add\_subplot(111, projection='3d')  
  
## positive samples  
x\_1 = np\_data[0:50,0]  
y\_1 = np\_data[0:50,1]  
z\_1 = np\_data[0:50,2]  
  
## negative samples  
x\_0 = np\_data[50:,0]  
y\_0 = np\_data[50:,1]  
z\_0 = np\_data[50:,2]  
  
# 전체 data  
x1 = np\_data[0:,0]  
x2 = np\_data[0:,1]  
x3 = np\_data[0:,2]  
label = np\_data[0:,3]  
  
  
# numpy 배열로 변경  
w1\_population = np.array(random\_weight\_list())  
w2\_population = np.array(random\_weight\_list())  
w3\_population = np.array(random\_weight\_list())  
b\_population = np.array(random\_bias\_list())  
  
  
# 세대 반복  
for generation in range(max\_generation\_number):  
 # population 을 한 녀석씩 봄.  
 # index별로 fitness를 저장.  
 error\_sum\_list = list() # 오차 합을 (인덱스, 오차 합) 의 튜플 형식으로 저장한다.  
 for learn\_idx in range(population\_number):  
 # zeta 함수와 sigmoid 함수 이용해, 인공신경망이 산출해 낸 label 값 가져옴.  
 zeta\_list = zeta(w1\_population[learn\_idx], w2\_population[learn\_idx], w3\_population[learn\_idx], x1, x2, x3, b\_population[learn\_idx])  
 learning\_label\_list = sigmoid\_list(zeta\_list)  
 # fitness 검사과정  
 error\_sum = 0 # 임시로 오차 합을 저장할 변수  
  
 # 오차합을 구하는 과정.  
 for fit\_idx in range(0, 100):  
 error\_sum += pow(learning\_label\_list[fit\_idx] - label[fit\_idx], 2) # 오차합은 제곱을 해준다.  
  
 error\_sum\_list.append((learn\_idx, error\_sum)) # 구한 오차합을, 현재 population 의 idx와 함께 저장해준다.  
  
  
 error\_sum\_list.sort(key=lambda x: x[1]) # 튜플을 두번째 원소를 이용해, 오름차순 정렬  
  
 # 부모 생성해 교배 시키기. 해당 리스트에는, 상위 14% 가 가지는 원소가 들어간다.  
 parent\_w1 = list()  
 parent\_w2 = list()  
 parent\_w3 = list()  
 parent\_b = list()  
  
 # 유전자 추가. 상위 14프로만 사용. 14프로만 사용하는 이유는 세대수 90과 교배규칙으로 나온 형질의 수를 맞추기 위해서  
 for i in range(select\_number):  
 parent\_w1.append(w1\_population[error\_sum\_list[i][0]])  
 parent\_w2.append(w2\_population[error\_sum\_list[i][0]])  
 parent\_w3.append(w3\_population[error\_sum\_list[i][0]])  
 parent\_b.append(b\_population[error\_sum\_list[i][0]])  
  
 # 유전자 교배시키기.  
 next\_population = crossover(parent\_w1, parent\_w2, parent\_w3, parent\_b)  
 # 새로운 세대교체  
 w1\_population = next\_population[0]  
 w2\_population = next\_population[1]  
 w3\_population = next\_population[2]  
 b\_population = next\_population[3]  
  
 # 모든 개체는 1퍼센트의 확률로 변이할 수도 있다.  
 for i in range(population\_number):  
 randomidx = random.randrange(0, 100)  
 if randomidx==1:  
 print("mutation occur") # mutation이 발생할 경우 알려주는 문구.  
 # 임의의 변이 규칙에 의해 개체에 변이가 일어난다.  
 w1\_population[i] = w1\_population[i] + 0.2  
 w2\_population[i] = w2\_population[i] - 0.2  
 w3\_population[i] = w3\_population[i] / 2  
 b\_population[i] = b\_population[i] - 2  
 print("--- %d 세대 의 fitness score ---" %(generation+1))  
 print("1st fitness score =", error\_sum\_list[0][1])  
 print("2nd fitness score =", error\_sum\_list[1][1])  
 print("3rd fitness score =", error\_sum\_list[2][1])  
 print("4th fitness score =", error\_sum\_list[3][1])  
  
 # 10번째 개체의 fitness 가 0.0001이하로 떨어지면, 종료시킨다.  
 if error\_sum\_list[9][1] < 0.0001 : break  
  
 # 그래프 각각의 fig에 추가함.  
visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[0][0]], w2\_population[error\_sum\_list[0][0]], w3\_population[error\_sum\_list[0][0]], b\_population[error\_sum\_list[0][0]], ax)  
visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[1][0]], w2\_population[error\_sum\_list[1][0]], w3\_population[error\_sum\_list[1][0]], b\_population[error\_sum\_list[1][0]], ax2)  
visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[2][0]], w2\_population[error\_sum\_list[2][0]], w3\_population[error\_sum\_list[2][0]], b\_population[error\_sum\_list[2][0]], ax3)  
visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[3][0]], w2\_population[error\_sum\_list[3][0]], w3\_population[error\_sum\_list[3][0]], b\_population[error\_sum\_list[3][0]], ax4)  
  
plt.show()

* 1. (2,3)과정을 반복하여 2세대, 3세대에서 가장 fitness가 높은 상위 4개의 유전자에 대해 fitness score를 적고, 분류 평면을 도시화 하라. [10점]

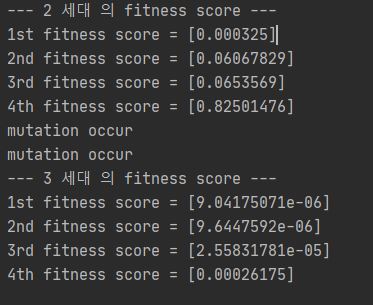
|  |  |  |  |
| --- | --- | --- | --- |
| Ranking1, Fitness score: 0.000325 | Ranking2, Fitness score: 0.06067829 | Ranking3, Fitness score: 0.0653569 | Ranking4, Fitness score: 0.82501476 |

표2. 2세대 유전자 (Rank1~4)

|  |  |  |  |
| --- | --- | --- | --- |
| Ranking1, Fitness score: 9.04175071e-06 | Ranking2, Fitness score: 9.6447592e-06 | Ranking3, Fitness score: 2.55831781e-05 | Ranking4, Fitness score: 0.00026175 |

표3. 3세대 유전자 (Rank1~4)

실행결과

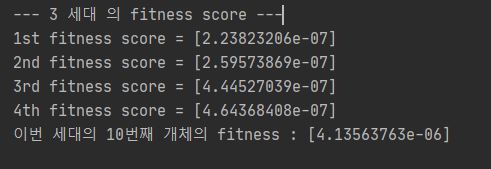


from mpl\_toolkits.mplot3d import Axes3D  
import matplotlib.pyplot as plt  
from matplotlib import cm  
import numpy as np  
import pandas as pd  
import random  
import math  
  
max\_generation\_number = 3 # 최대로 학습할 세대  
population\_number = 90 # 세대 당 개체 수  
select\_number = 10 # 선택할 개체의 갯수  
# 첫 세대의 가중치를 랜덤으로 생성하고, 배열로 반환.  
def random\_weight\_list():  
 weight\_list = list() # 리스트 선언  
 for i in range(population\_number):  
 weight\_list.append(random.uniform(-1,1)) # uniform 함수 이용해 랜덤으로 값 추가.  
 return weight\_list  
  
# 첫 세대의 bias 를 랜덤으로 생성하고, 배열로 반환.  
def random\_bias\_list():  
 bias\_list = list()  
 for i in range(population\_number):  
 bias\_list.append(random.uniform(-100,100)) # uniform 함수 이용해 랜덤으로 값 추가.  
 return bias\_list  
  
# sigmoid 함수  
def sigmoid(x):  
 # math.exp(x) = e^x 와 같다.  
 y = 1 / (1 + math.exp(-x))  
 return y  
  
# zeta 함수 값을 sigmoid 처리한 label 들을 반환하는 함수  
def sigmoid\_list(zeta\_list):  
 y = np.zeros((len(zeta\_list), 1)) # 행렬 생성  
 # zeta\_list 의 개별 원소를 sigmoid 함수에 넣어서, y에 값 넣기  
 for i in range(0, len(zeta\_list)):  
 y[i] = sigmoid(zeta\_list[i])  
 # y 반환  
 return y  
  
# zeta 함수  
def zeta(w1, w2, w3, x1, x2, x3, b):  
 z = np.zeros((100, 1))  
  
 for i in range(100):  
 z[i] = w1\*x1[i] + w2\*x2[i] + w3\*x3[i] + b  
 return z  
  
# 유전자 교배  
def crossover(w1\_list, w2\_list, w3\_list, b\_list):  
 # 다음 세대로 전달될 w1, w2, w3, b  
 w1\_next\_generation = list()  
 w2\_next\_generation = list()  
 w3\_next\_generation = list()  
 b\_next\_generation = list()  
  
 # 유전자를 임시로 저장할 리스트  
 w1\_gene\_list = list()  
 w2\_gene\_list = list()  
 w3\_gene\_list = list()  
 b\_gene\_list = list()  
 # 유전자 생성. 가중치 유전자는 각각 앞의 소수 5자리와 뒷소수 5자리로 나누어진다. bias 는 부모의 평균값을 가져옴.  
 # 자세한 사항은 레포트에 기재  
 for i in range(select\_number):  
 w1\_gene\_list.append(weight\_get\_gene(w1\_list[i]))  
 w2\_gene\_list.append(weight\_get\_gene(w2\_list[i]))  
 w3\_gene\_list.append(weight\_get\_gene(w3\_list[i]))  
 b\_gene\_list.append(b\_list[i])  
  
 # 유전자 교배. 앞자리 + 뒷자리, b는 부모의 평균으로 구해온다.  
 for i in range(select\_number):  
 for j in range(i+1, select\_number):  
 w1\_next\_generation.append(w1\_gene\_list[i][0] + w1\_gene\_list[j][1])  
 w2\_next\_generation.append(w2\_gene\_list[i][0] + w2\_gene\_list[j][1])  
 w3\_next\_generation.append(w3\_gene\_list[i][0] + w3\_gene\_list[j][1])  
 b\_next\_generation.append((b\_gene\_list[i] + b\_gene\_list[j]) / 2)  
  
  
 # 유전자 교배. 부모의 앞자리를 더한 평균 값. b도 마찬가지로 부모의 평균으로 구해온다.  
 for i in range(select\_number):  
 for j in range(i+1, select\_number):  
 w1\_next\_generation.append((w1\_gene\_list[i][0] + w1\_gene\_list[j][0])/2)  
 w2\_next\_generation.append((w2\_gene\_list[i][0] + w2\_gene\_list[j][0])/2)  
 w3\_next\_generation.append((w3\_gene\_list[i][0] + w3\_gene\_list[j][0])/2)  
 b\_next\_generation.append((b\_gene\_list[i] + b\_gene\_list[j]) / 2)  
  
 # 혹시나 세대수를 변경할 시, 필요 이상의 유전자가 만들어지면, 임의로 제거한다.  
 for i in range(population\_number - len(w1\_next\_generation)):  
 kill\_index = int(random.random() % 100)  
 del w1\_next\_generation[kill\_index]  
 del w2\_next\_generation[kill\_index]  
 del w3\_next\_generation[kill\_index]  
 del b\_next\_generation[kill\_index]  
  
 # 리스트형으로 반환  
 return [w1\_next\_generation, w2\_next\_generation, w3\_next\_generation, b\_next\_generation]  
  
# 가중치 유전자를 tuple로 생성후 반환한다.  
# 유전자 생성은 a 는 소수점 5자리까지, b 는 소수점 5자리 뒤부터 이다.  
def weight\_get\_gene(source):  
 gene\_a = ((source\*100000) // 1) / 100000  
 gene\_b = source % 0.00001  
 return gene\_a, gene\_b  
  
# 출력해주는 함수  
def visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1, w2, w3, b, ax):  
 ax.plot(x\_0, y\_0, z\_0, linestyle="none", marker="o", mfc="none", markeredgecolor="r") # 샘플 출력  
 ax.plot(x\_1, y\_1, z\_1, linestyle="none", marker="o", mfc="none", markeredgecolor="b") # 샘플 출력  
  
 X = np.arange(0, 2, 0.1) \* 100  
 Y = np.arange(0, 2, 0.1) \* 100  
 X, Y = np.meshgrid(X, Y)  
  
 Z = (-float(w1) / w3 \* X) + (-float(w2) / w3 \* Y) - float(b) / w3 # 평면의 방정식  
  
 ax.plot\_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.4, cmap=cm.Blues) # 평면 출력  
  
  
data = pd.read\_csv("p1\_training\_data.csv") # 데이터 읽기  
np\_data = np.array(data) # np Array 로 변경  
  
fig\_list = list()  
for i in range(8):  
 fig\_list.append(plt.figure())  
  
ax\_list = list()  
for i in range(8):  
 ax\_list.append(fig\_list[i].add\_subplot(111, projection="3d"))  
  
## positive samples  
x\_1 = np\_data[0:50,0]  
y\_1 = np\_data[0:50,1]  
z\_1 = np\_data[0:50,2]  
  
## negative samples  
x\_0 = np\_data[50:,0]  
y\_0 = np\_data[50:,1]  
z\_0 = np\_data[50:,2]  
  
# 전체 data  
x1 = np\_data[0:,0]  
x2 = np\_data[0:,1]  
x3 = np\_data[0:,2]  
label = np\_data[0:,3]  
  
  
# numpy 배열로 변경  
w1\_population = np.array(random\_weight\_list())  
w2\_population = np.array(random\_weight\_list())  
w3\_population = np.array(random\_weight\_list())  
b\_population = np.array(random\_bias\_list())  
  
  
# 세대 반복  
for generation in range(max\_generation\_number):  
 # population 을 한 녀석씩 봄.  
 # index별로 fitness를 저장.  
 error\_sum\_list = list() # 오차 합을 (인덱스, 오차 합) 의 튜플 형식으로 저장한다.  
 for learn\_idx in range(population\_number):  
 # zeta 함수와 sigmoid 함수 이용해, 인공신경망이 산출해 낸 label 값 가져옴.  
 zeta\_list = zeta(w1\_population[learn\_idx], w2\_population[learn\_idx], w3\_population[learn\_idx], x1, x2, x3, b\_population[learn\_idx])  
 learning\_label\_list = sigmoid\_list(zeta\_list)  
 # fitness 검사과정  
 error\_sum = 0 # 임시로 오차 합을 저장할 변수  
  
 # 오차합을 구하는 과정.  
 for fit\_idx in range(0, 100):  
 error\_sum += pow(learning\_label\_list[fit\_idx] - label[fit\_idx], 2) # 오차합은 제곱을 해준다.  
  
 error\_sum\_list.append((learn\_idx, error\_sum)) # 구한 오차합을, 현재 population 의 idx와 함께 저장해준다.  
  
  
 error\_sum\_list.sort(key=lambda x: x[1]) # 튜플을 두번째 원소를 이용해, 오름차순 정렬  
  
 # 부모 생성해 교배 시키기. 해당 리스트에는, 상위 14% 가 가지는 원소가 들어간다.  
 parent\_w1 = list()  
 parent\_w2 = list()  
 parent\_w3 = list()  
 parent\_b = list()  
  
 # 유전자 추가. 상위 14프로만 사용. 14프로만 사용하는 이유는 세대수 90과 교배규칙으로 나온 형질의 수를 맞추기 위해서  
 for i in range(select\_number):  
 parent\_w1.append(w1\_population[error\_sum\_list[i][0]])  
 parent\_w2.append(w2\_population[error\_sum\_list[i][0]])  
 parent\_w3.append(w3\_population[error\_sum\_list[i][0]])  
 parent\_b.append(b\_population[error\_sum\_list[i][0]])  
  
 # 유전자 교배시키기.  
 next\_population = crossover(parent\_w1, parent\_w2, parent\_w3, parent\_b)  
 # 새로운 세대교체  
 w1\_population = next\_population[0]  
 w2\_population = next\_population[1]  
 w3\_population = next\_population[2]  
 b\_population = next\_population[3]  
  
 # 모든 개체는 1퍼센트의 확률로 변이할 수도 있다.  
 for i in range(population\_number):  
 randomidx = random.randrange(0, 100)  
 if randomidx==1:  
 print("mutation occur") # mutation이 발생할 경우 알려주는 문구.  
 # 임의의 변이 규칙에 의해 개체에 변이가 일어난다.  
 w1\_population[i] = w1\_population[i] + 0.2  
 w2\_population[i] = w2\_population[i] - 0.2  
 w3\_population[i] = w3\_population[i] / 2  
 b\_population[i] = b\_population[i] - 2  
 print("--- %d 세대 의 fitness score ---" %(generation+1))  
 print("1st fitness score =", error\_sum\_list[0][1])  
 print("2nd fitness score =", error\_sum\_list[1][1])  
 print("3rd fitness score =", error\_sum\_list[2][1])  
 print("4th fitness score =", error\_sum\_list[3][1])  
 # 각각 2, 3 세대의 값 시각화.  
 if generation == 1:  
 visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[0][0]],  
 w2\_population[error\_sum\_list[0][0]], w3\_population[error\_sum\_list[0][0]],  
 b\_population[error\_sum\_list[0][0]], ax\_list[0])  
 visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[1][0]],  
 w2\_population[error\_sum\_list[1][0]], w3\_population[error\_sum\_list[1][0]],  
 b\_population[error\_sum\_list[1][0]], ax\_list[1])  
 visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[2][0]],  
 w2\_population[error\_sum\_list[2][0]], w3\_population[error\_sum\_list[2][0]],  
 b\_population[error\_sum\_list[2][0]], ax\_list[2])  
 visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[3][0]],  
 w2\_population[error\_sum\_list[3][0]], w3\_population[error\_sum\_list[3][0]],  
 b\_population[error\_sum\_list[3][0]], ax\_list[3])  
 elif generation == 2:  
 visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[0][0]],  
 w2\_population[error\_sum\_list[0][0]], w3\_population[error\_sum\_list[0][0]],  
 b\_population[error\_sum\_list[0][0]], ax\_list[4])  
 visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[1][0]],  
 w2\_population[error\_sum\_list[1][0]], w3\_population[error\_sum\_list[1][0]],  
 b\_population[error\_sum\_list[1][0]], ax\_list[5])  
 visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[2][0]],  
 w2\_population[error\_sum\_list[2][0]], w3\_population[error\_sum\_list[2][0]],  
 b\_population[error\_sum\_list[2][0]], ax\_list[6])  
 visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[3][0]],  
 w2\_population[error\_sum\_list[3][0]], w3\_population[error\_sum\_list[3][0]],  
 b\_population[error\_sum\_list[3][0]], ax\_list[7])  
  
  
plt.show()

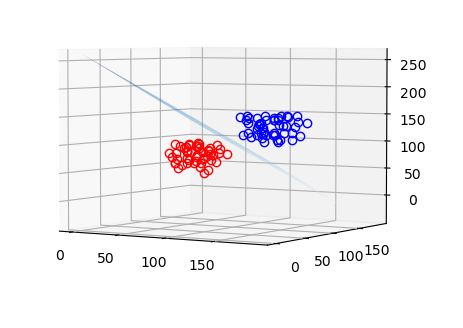
* 1. 최종적으로 유전알고리즘을 통해 얻어진 유전자중 가장 fitness가 높은 유전 자에 대해 분류 평면을 도시화 하라. 또한 어떤 조건으로 유전 알고리즘을 종료 하였는지 작성하라. [5점]

각 세대에서 선택될 개체들 중에서, 가장 fitness가 낮은 10번째 개체의 fitness의 값이 0.0001보다 작으면, 정확하게 분류했다 판단하고 종료합니다.

실행결과



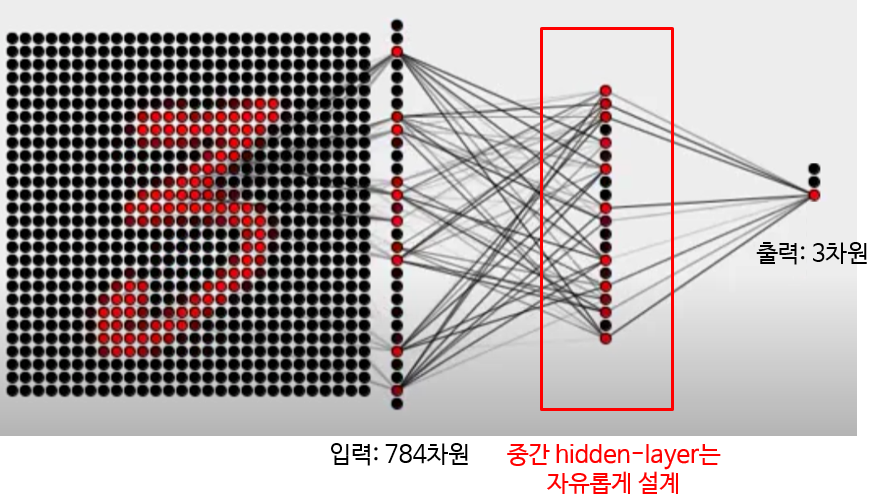
3세대 fittest gene. Fitness score : 4.13563763e-06



**w1, w2, w3, b 는 랜덤으로 만들어집니다. 1번 소주제의 모든 문제의 초기 세대는 각각 모두 다름을 다시 한번 알려드립니다.**

from mpl\_toolkits.mplot3d import Axes3D  
import matplotlib.pyplot as plt  
from matplotlib import cm  
import numpy as np  
import pandas as pd  
import random  
import math  
  
max\_generation\_number = 40 # 최대로 학습할 세대  
population\_number = 90 # 세대 당 개체 수  
select\_number = 10 # 선택할 개체의 갯수  
# 첫 세대의 가중치를 랜덤으로 생성하고, 배열로 반환.  
def random\_weight\_list():  
 weight\_list = list() # 리스트 선언  
 for i in range(population\_number):  
 weight\_list.append(random.uniform(-1,1)) # uniform 함수 이용해 랜덤으로 값 추가.  
 return weight\_list  
  
# 첫 세대의 bias 를 랜덤으로 생성하고, 배열로 반환.  
def random\_bias\_list():  
 bias\_list = list()  
 for i in range(population\_number):  
 bias\_list.append(random.uniform(-100,100)) # uniform 함수 이용해 랜덤으로 값 추가.  
 return bias\_list  
  
# sigmoid 함수  
def sigmoid(x):  
 # math.exp(x) = e^x 와 같다.  
 y = 1 / (1 + math.exp(-x))  
 return y  
  
# zeta 함수 값을 sigmoid 처리한 label 들을 반환하는 함수  
def sigmoid\_list(zeta\_list):  
 y = np.zeros((len(zeta\_list), 1)) # 행렬 생성  
 # zeta\_list 의 개별 원소를 sigmoid 함수에 넣어서, y에 값 넣기  
 for i in range(0, len(zeta\_list)):  
 y[i] = sigmoid(zeta\_list[i])  
 # y 반환  
 return y  
  
# zeta 함수  
def zeta(w1, w2, w3, x1, x2, x3, b):  
 z = np.zeros((100, 1))  
  
 for i in range(100):  
 z[i] = w1\*x1[i] + w2\*x2[i] + w3\*x3[i] + b  
 return z  
  
# 유전자 교배  
def crossover(w1\_list, w2\_list, w3\_list, b\_list):  
 # 다음 세대로 전달될 w1, w2, w3, b  
 w1\_next\_generation = list()  
 w2\_next\_generation = list()  
 w3\_next\_generation = list()  
 b\_next\_generation = list()  
  
 # 유전자를 임시로 저장할 리스트  
 w1\_gene\_list = list()  
 w2\_gene\_list = list()  
 w3\_gene\_list = list()  
 b\_gene\_list = list()  
 # 유전자 생성. 가중치 유전자는 각각 앞의 소수 5자리와 뒷소수 5자리로 나누어진다. bias 는 부모의 평균값을 가져옴.  
 # 자세한 사항은 레포트에 기재  
 for i in range(select\_number):  
 w1\_gene\_list.append(weight\_get\_gene(w1\_list[i]))  
 w2\_gene\_list.append(weight\_get\_gene(w2\_list[i]))  
 w3\_gene\_list.append(weight\_get\_gene(w3\_list[i]))  
 b\_gene\_list.append(b\_list[i])  
  
 # 유전자 교배. 앞자리 + 뒷자리, b는 부모의 평균으로 구해온다.  
 for i in range(select\_number):  
 for j in range(i+1, select\_number):  
 w1\_next\_generation.append(w1\_gene\_list[i][0] + w1\_gene\_list[j][1])  
 w2\_next\_generation.append(w2\_gene\_list[i][0] + w2\_gene\_list[j][1])  
 w3\_next\_generation.append(w3\_gene\_list[i][0] + w3\_gene\_list[j][1])  
 b\_next\_generation.append((b\_gene\_list[i] + b\_gene\_list[j]) / 2)  
  
  
 # 유전자 교배. 부모의 앞자리를 더한 평균 값. b도 마찬가지로 부모의 평균으로 구해온다.  
 for i in range(select\_number):  
 for j in range(i+1, select\_number):  
 w1\_next\_generation.append((w1\_gene\_list[i][0] + w1\_gene\_list[j][0])/2)  
 w2\_next\_generation.append((w2\_gene\_list[i][0] + w2\_gene\_list[j][0])/2)  
 w3\_next\_generation.append((w3\_gene\_list[i][0] + w3\_gene\_list[j][0])/2)  
 b\_next\_generation.append((b\_gene\_list[i] + b\_gene\_list[j]) / 2)  
  
 # 혹시나 세대수를 변경할 시, 필요 이상의 유전자가 만들어지면, 임의로 제거한다.  
 for i in range(population\_number - len(w1\_next\_generation)):  
 kill\_index = int(random.random() % 100)  
 del w1\_next\_generation[kill\_index]  
 del w2\_next\_generation[kill\_index]  
 del w3\_next\_generation[kill\_index]  
 del b\_next\_generation[kill\_index]  
  
 # 리스트형으로 반환  
 return [w1\_next\_generation, w2\_next\_generation, w3\_next\_generation, b\_next\_generation]  
  
# 가중치 유전자를 tuple로 생성후 반환한다.  
# 유전자 생성은 a 는 소수점 5자리까지, b 는 소수점 5자리 뒤부터 이다.  
def weight\_get\_gene(source):  
 gene\_a = ((source\*100000) // 1) / 100000  
 gene\_b = source % 0.00001  
 return gene\_a, gene\_b  
  
# 출력해주는 함수  
def visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1, w2, w3, b, ax):  
 ax.plot(x\_0, y\_0, z\_0, linestyle="none", marker="o", mfc="none", markeredgecolor="r") # 샘플 출력  
 ax.plot(x\_1, y\_1, z\_1, linestyle="none", marker="o", mfc="none", markeredgecolor="b") # 샘플 출력  
  
 X = np.arange(0, 2, 0.1) \* 100  
 Y = np.arange(0, 2, 0.1) \* 100  
 X, Y = np.meshgrid(X, Y)  
  
 Z = (-float(w1) / w3 \* X) + (-float(w2) / w3 \* Y) - float(b) / w3 # 평면의 방정식  
  
 ax.plot\_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.4, cmap=cm.Blues) # 평면 출력  
  
  
data = pd.read\_csv("p1\_training\_data.csv") # 데이터 읽기  
np\_data = np.array(data) # np Array 로 변경  
  
fig\_list = list()  
for i in range(1):  
 fig\_list.append(plt.figure())  
  
ax\_list = list()  
for i in range(1):  
 ax\_list.append(fig\_list[i].add\_subplot(111, projection="3d"))  
  
## positive samples  
x\_1 = np\_data[0:50,0]  
y\_1 = np\_data[0:50,1]  
z\_1 = np\_data[0:50,2]  
  
## negative samples  
x\_0 = np\_data[50:,0]  
y\_0 = np\_data[50:,1]  
z\_0 = np\_data[50:,2]  
  
# 전체 data  
x1 = np\_data[0:,0]  
x2 = np\_data[0:,1]  
x3 = np\_data[0:,2]  
label = np\_data[0:,3]  
  
  
# numpy 배열로 변경  
w1\_population = np.array(random\_weight\_list())  
w2\_population = np.array(random\_weight\_list())  
w3\_population = np.array(random\_weight\_list())  
b\_population = np.array(random\_bias\_list())  
  
  
# 세대 반복  
for generation in range(max\_generation\_number):  
 # population 을 한 녀석씩 봄.  
 # index별로 fitness를 저장.  
 error\_sum\_list = list() # 오차 합을 (인덱스, 오차 합) 의 튜플 형식으로 저장한다.  
 for learn\_idx in range(population\_number):  
 # zeta 함수와 sigmoid 함수 이용해, 인공신경망이 산출해 낸 label 값 가져옴.  
 zeta\_list = zeta(w1\_population[learn\_idx], w2\_population[learn\_idx], w3\_population[learn\_idx], x1, x2, x3, b\_population[learn\_idx])  
 learning\_label\_list = sigmoid\_list(zeta\_list)  
 # fitness 검사과정  
 error\_sum = 0 # 임시로 오차 합을 저장할 변수  
  
 # 오차합을 구하는 과정.  
 for fit\_idx in range(0, 100):  
 error\_sum += pow(learning\_label\_list[fit\_idx] - label[fit\_idx], 2) # 오차합은 제곱을 해준다.  
  
 error\_sum\_list.append((learn\_idx, error\_sum)) # 구한 오차합을, 현재 population 의 idx와 함께 저장해준다.  
  
  
 error\_sum\_list.sort(key=lambda x: x[1]) # 튜플을 두번째 원소를 이용해, 오름차순 정렬  
  
 # 부모 생성해 교배 시키기. 해당 리스트에는, 상위 14% 가 가지는 원소가 들어간다.  
 parent\_w1 = list()  
 parent\_w2 = list()  
 parent\_w3 = list()  
 parent\_b = list()  
  
 # 유전자 추가. 상위 14프로만 사용. 14프로만 사용하는 이유는 세대수 90과 교배규칙으로 나온 형질의 수를 맞추기 위해서  
 for i in range(select\_number):  
 parent\_w1.append(w1\_population[error\_sum\_list[i][0]])  
 parent\_w2.append(w2\_population[error\_sum\_list[i][0]])  
 parent\_w3.append(w3\_population[error\_sum\_list[i][0]])  
 parent\_b.append(b\_population[error\_sum\_list[i][0]])  
  
 # 유전자 교배시키기.  
 next\_population = crossover(parent\_w1, parent\_w2, parent\_w3, parent\_b)  
 # 새로운 세대교체  
 w1\_population = next\_population[0]  
 w2\_population = next\_population[1]  
 w3\_population = next\_population[2]  
 b\_population = next\_population[3]  
  
 # 모든 개체는 1퍼센트의 확률로 변이할 수도 있다.  
 for i in range(population\_number):  
 randomidx = random.randrange(0, 100)  
 if randomidx==1:  
 print("mutation occur") # mutation이 발생할 경우 알려주는 문구.  
 # 임의의 변이 규칙에 의해 개체에 변이가 일어난다.  
 w1\_population[i] = w1\_population[i] + 0.2  
 w2\_population[i] = w2\_population[i] - 0.2  
 w3\_population[i] = w3\_population[i] / 2  
 b\_population[i] = b\_population[i] - 2  
 print("--- %d 세대 의 fitness score ---" %(generation+1))  
 print("1st fitness score =", error\_sum\_list[0][1])  
 print("2nd fitness score =", error\_sum\_list[1][1])  
 print("3rd fitness score =", error\_sum\_list[2][1])  
 print("4th fitness score =", error\_sum\_list[3][1])  
  
 # 각 세대에서 선택될 개체들 중에서, 가장 fitness가 낮은 10번째 개체의 fitness의 값이 0.0001보다 작으면, 정확하게 분류했다 판단하고 종료합니다.  
 if error\_sum\_list[9][1] < 0.0001:  
 print("이번 세대의 10번째 개체의 fitness :", error\_sum\_list[9][1])  
 break  
  
  
  
# best fitness 를 가지는 것 시각화.  
visualize\_grid(x\_0, y\_0, z\_0, x\_1, y\_1, z\_1, w1\_population[error\_sum\_list[0][0]],  
 w2\_population[error\_sum\_list[0][0]], w3\_population[error\_sum\_list[0][0]],  
 b\_population[error\_sum\_list[0][0]], ax\_list[0])  
  
plt.show()

**2. 유전 알고리즘을 통해 주어진 데이터를 분류하는 Multi-layer neural network를 학습하라. [전체: 50점]**아래 세부 사항을 확인하여 문제를 해결 하시오.

* [학습 데이터]  
  이미지 데이터: p2\_training\_data\_mnist.csv (600개x784차원데이터)  
  1~3 숫자 600개데이터가 저장 되어있다. 하나의 숫자는 28x28=784차원의 데이터이다.  
    
  Label 데이터: p2\_training\_data\_label.csv (600개x3차원데이터)  
  3차원으로 표현되며 숫자 1은 [1,0,0], 2는 [0,1,0], 3은 [0,0,1]의 형태로 저장 되어있다.
* 숫자 이미지 1,2,3을 분류할 수 있는   
  Multi-layer neural network의 구조는 자유롭게 설계할 것.   
  (단, 입력은 784차원, 출력은 3차원으로 설계. 중간 layer의 node 수 및 층 수는 자유롭게 설계)  
  [아래 그림 참조]  
  
* Activation function: Sigmoid function 을 사용할 것.
* 유전알고리즘 동작을 위한 모든 설계 조건은 자율적으로 설계 할 것.
* (Population수, fitness계산방법, Selection방법, crossover방법, mutation방법, 유전자(학습 파라미터)초기화 방법, 알고리즘 종료 조건 등)

2-1) 설계한 네트워크의 구조는 어떻게 되는가? (입력과 출력층 사이에 적어도 1개 이상의 층을 반드시 설계하라.) [10점]  
 (ex. Layer1(input): 784nodes / Layer2: 18nodes / Layer3(output): 3nodes)

2-2) 유전알고리즘을 통해 2-1)에서 설계한 네트워크를 학습하시오. 알고리즘이 종료될때까지, 각 세대별로 가장 높은 fitness 를 출력하시오. [20점]  
[실행 예]  
Generation 1 best fitness : 580.3221  
Generation 2 best fitness : 215.1078  
Generation 3 best fitness : 126.6617  
Generation 4 best fitness : 150.3162  
…  
Generation 10 best fitness : 0.0122

2-3) 학습한 최종 네트워크(가장 높은 fitness)로 p2\_test\_data\_mnist.csv (30개x784차원데이터) 데이터에 대한 test를 수행하라. [20점]  
[실행 예]  
Test 1 : [1, 0, 0]  
Test 2 : [1, 0, 0]  
Test 3 : [1, 0, 0]  
…  
Test 30 : [0, 0, 1]  
  
[참고]  
Test 결과는 아래 예시와 같이 0~1사이의 확률 값으로 출력이 될 수 있다.   
예시) [0.991, 0.011, 0.021] 이 경우 아래와 같이 값이 가장 큰 결과를 1로, 나머지는 0으로 변환하여 최종 결과를 얻는다.  
예시) 네트워크 출력 값: [0.991, 0.011, 0.021] → 최종 숫자 판별 값 [1, 0, 0]