1　プログラム

実行環境と用いた言語・ライブラリを以下の表 1 に示す。

表 1: プログラムの実行環境

OS	:	Microsoft Windows 10 Pro (64bit)
CPU	:	Intel(R) Core(TM) i5-4300U
RAM	:	4.00 GB
使用言語	:	Python3.6
可視化	:	matplotlib ライブラリ

Listings 1: neural_network.py

```
# -*- coding: utf-8 -*-

import numpy as np


def identity_function(x):
    return x


def deriv_identity_function(x):
    return np.ones_like(x)


def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))


def deriv_sigmoid(x):
    return sigmoid(x) * (1 - sigmoid(x))


class FC(object):
    def __init__(self, input_size, output_size, activate_func,
    activate_func_deriv,):
        self.W = np.random.rand(input_size, output_size).astype(float)
        self.b = np.zeros(output_size, dtype=float)

        self.dW = None
        self.db = None

        self.x = None
        self.u = None

```

```
        self.delta = None

        self.activate_func = activate_func
        self.activate_func_deriv = activate_func_deriv

    def __call__(self, x):
        self.x = x
        self.u = x.dot(self.W) + self.b
        h = self.activate_func(self.u)
        return h

    def back_prop(self, delta, W):
        self.delta = self.activate_func_deriv(self.u) * delta.dot(W.T)
        return self.delta

    def compute_grad(self):
        batch_size = self.delta.shape[0]
        self.dW = self.x.T.dot(self.delta) / batch_size
        self.db = self.delta.mean()


class Model(object):
    def __init__(self, layers):
        self.layers = layers

    def __call__(self, x, is_training=False):
        if is_training:
            for layer in self.layers:
                x = layer(x)
        else:
            for layer in self.layers:
                x = x.dot(layer.W) + layer.b
                x = layer.activate_func(x)
        return x

    def back_propagate(self, delta):
        W = None
        for i, layer in enumerate(self.layers[::-1]):
            if i == 0:
                layer.delta = delta
            else:
                delta = layer.back_prop(delta, W)
            layer.compute_grad()
            W = layer.W

    def update(self, lr):
        for layer in self.layers:
```

```
80              layer.W -= lr * layer.dW
81              layer.b -= lr * layer.db
82          return
```

Listings 2: assignment1.py

```
# -*- coding: utf-8 -*-


import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat

from neural_network import *


def load_data(path):
    data = loadmat(path)
    #print(data.keys())
    x = data['x'].T
    y = data['l'].T
    n = data['n'][0, 0]
    d = data['d'][0, 0]
    return x, y, n, d


def compute_loss(y_pred, y_true):
    loss = np.mean((y_pred - y_true)**2)
    return loss


def plot(model, x, l, xx, yy, axy):
    # settings
    plt.clf()
    plt.xlim([-1, 1])
    plt.ylim([-1, 1])

    # scatter
    plt.plot(x[np.where(l==1), 0], x[np.where(l==1), 1], 'bo')
    plt.plot(x[np.where(l==0), 0], x[np.where(l==0), 1], 'bx')

    # contour
    p = model(axy, is_training=False)  # compute classification results
    cs = plt.contour(
        xx, yy, np.reshape(p, xx.shape),
        levels=[-5, 0, 5],
        colors='g',
        )
    plt.clabel(cs)

    # show
    plt.show()
```

```
47      plt.pause(0.000001)
48
49
50  def train(model, x, y, lr, epochs, fig_path=None,):
51      n_sample = len(y)
52
53      xx, yy = np.meshgrid(np.linspace(-1,1), np.linspace(-1,1))
54      xf = xx.flatten()[:, np.newaxis]
55      yf = yy.flatten()[:, np.newaxis]
56      axy = np.concatenate((
57          xf,
58          yf
59          ),axis=1)
60      l = (y == 1)
61      plt.figure()
62      plt.ion()
63
64      for epoch in range(epochs):
65          # forward
66          y_pred = model(x, is_training=True)
67          loss = compute_loss(y, y_pred)
68
69          # backward
70          delta = y_pred - y
71          model.back_propagate(delta)
72          model.update(lr)
73
74          # result
75          pred = y_pred
76          pred[pred > 0] = 1
77          pred[pred < 0] = -1
78          n_correct = (pred == y).sum()
79          acc = n_correct / n_sample
80          print(f'Epoch: {epoch+1}\t#Loss: {loss:.4f}\t#Correct:
        {n_correct}\tAcc: {acc:.3f}')
81
82          # plot
83          #plot(model, x, l, xx, yy, axy)
84
85
86      # plot
87      plot(model, x, l, xx, yy, axy)
88      if fig_path:
89          plt.savefig(fig_path)
90      plt.show()
91
92      return model
```

```
93
94
95  def main():
96      # settings
97      #data_type = 'linear'
98      #data_type = 'nonlinear'
99      data_type = 'slinear'
100     data_path = f'../data/{data_type}-data.mat'
101     fig_path = f'../figures/assignment1_1_{data_type}_result.png'
102     np.random.seed(1)
103
104
105     # hyperparameters
106     epochs = 100
107     lr = 0.10
108
109
110     # model
111     input_size = 2
112     output_size = 1
113     hidden_size = 20
114     model = Model([
115         FC(input_size, hidden_size, sigmoid, deriv_sigmoid),
116         FC(hidden_size, output_size, identity_function,
        deriv_identity_function),
117         ])
118
119
120     # load data
121     x, y, n, d = load_data(data_path)
122     print(f'Data Type: {data_type}  #Sample: {n}  #Dim: {d}\n')
123
124
125     # train
126     model = train(model, x, y, lr, epochs, fig_path=fig_path,)
127
128
129 if __name__ == '__main__':
130     main()
```