Exercises 6.4 Calculate the gradient using a single hidden layer basis

When employing M single hidden layer basis features (using any activation $a(\cdot)$) the full gradient of a cost g (e.g., the softmax) is a vector of length $Q = M(N+2)+1$ containing the derivatives of the cost with respect to each variable,

$$\nabla g = \begin{bmatrix} \frac{\partial}{\partial b} g & \frac{\partial}{\partial w_1} g & \cdots & \frac{\partial}{\partial w_M} g & \frac{\partial}{\partial c_1} g \cdots & \frac{\partial}{\partial c_M} g & \nabla_{\mathbf{v}_1}^T g & \cdots & \nabla_{\mathbf{v}_M}^T g \end{bmatrix}^T, \quad (6.31)$$

where the derivatives are easily calculated using the chain rule.

a) Using the chain rule verify that the derivatives of this gradient (using the softmax cost) are given by

$$\begin{aligned} \frac{\partial}{\partial b} g &= -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m a\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right) y_p \\ \frac{\partial}{\partial w_n} g &= -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m a\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right) a\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right) y_p \\ \frac{\partial}{\partial c_n} g &= -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m a\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right) a'\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right) w_n y_p \\ \nabla_{\mathbf{v}_n} g &= -\sum_{p=1}^{P} \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m a\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right) a'\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right) \mathbf{x}_p w_n y_p. \end{aligned} \quad (6.32)$$

b) This gradient can be written more efficiently for programming languages like Python and MATLAB/OCTAVE that have especially good implementations of matrix/vector operations by writing it more compactly. Supposing that $a = \tanh(\cdot)$ is the activation function (meaning $a' = \text{sech}^2(\cdot)$ is the hyperbolic secant function squared), verify that the derivatives from part a) may be written more compactly as

$$\begin{aligned} \frac{\partial}{\partial b} g &= -\mathbf{1}_{P\times 1}^T \mathbf{q} \odot \mathbf{y} \\ \frac{\partial}{\partial w_n} g &= -\mathbf{1}_{P\times 1}^T (\mathbf{q} \odot \mathbf{t}_n \odot \mathbf{y}) \\ \frac{\partial}{\partial c_n} g &= -\mathbf{1}_{P\times 1}^T (\mathbf{q} \odot \mathbf{s}_n \odot \mathbf{y}) w_n \\ \nabla_{\mathbf{v}_n} g &= -\mathbf{X} \cdot \mathbf{q} \odot \mathbf{s}_n \odot \mathbf{y} w_n, \end{aligned} \quad (6.33)$$

where $\odot$ denotes the component-wise product and denoting $q_p = \sigma\left(-y_p\left(b + \sum_{m=1}^{M} w_m \tanh\left(c_m + \mathbf{x}_p^T \mathbf{v}_m\right)\right)\right)$, $t_{np} = \tanh\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right)$, $s_{np} = \text{sech}^2\left(c_n + \mathbf{x}_p^T \mathbf{v}_n\right)$, and $\mathbf{q}$, $\mathbf{t}_n$, and $\mathbf{s}_n$ the P length vectors containing these entries.

Exercises 6.5 **Code up gradient descent using single hidden layer bases**

In this exercise you will reproduce the classification result using a single hidden layer feature basis with tanh activation shown in the middle panel of Fig. 6.9.

a) Plug the gradient from Exercise 6.4 into the gradient descent function

$$\mathbf{T} = \text{tanh\_softmax}\,(\mathbf{X}, \mathbf{y}, M) \tag{6.34}$$

located in the wrapper *single_layer_classification_hw* and the dataset *genreg_data.csv*, both of which may be downloaded from the book website. Here $\mathbf{T}$ is the set of optimal weights learned via gradient descent, $\mathbf{X}$ is the input data matrix, $\mathbf{y}$ contains the associated labels, and M is the number of basis features to employ.

Almost all of this function has already been constructed for you, e.g., various initializations, step length, etc., and you need only enter the gradient of the associated cost function. All of the additional code necessary to generate the associated plot is already provided in the wrapper. Due to the non-convexity of the associated cost function when using neural network features, the wrapper will run gradient descent several times and plot the result of each run.

b) Try adjusting the number of basis features M in the wrapper and run it several times. Is there a value of M other than $M = 4$ that seems to produce a good fit to the underlying function?

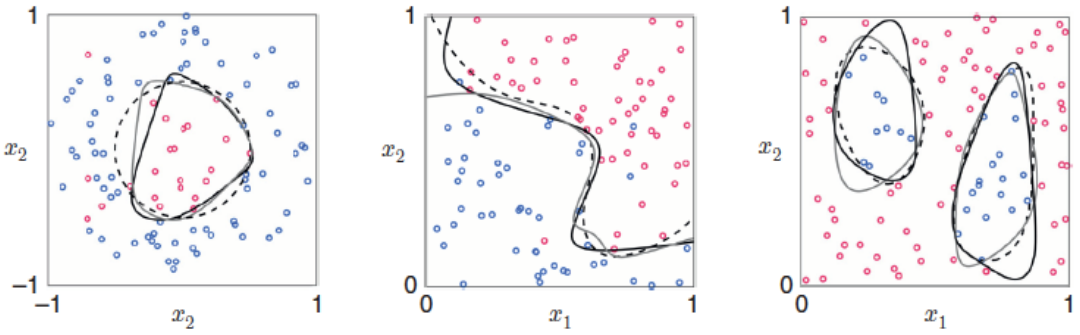


Fig. 6.9

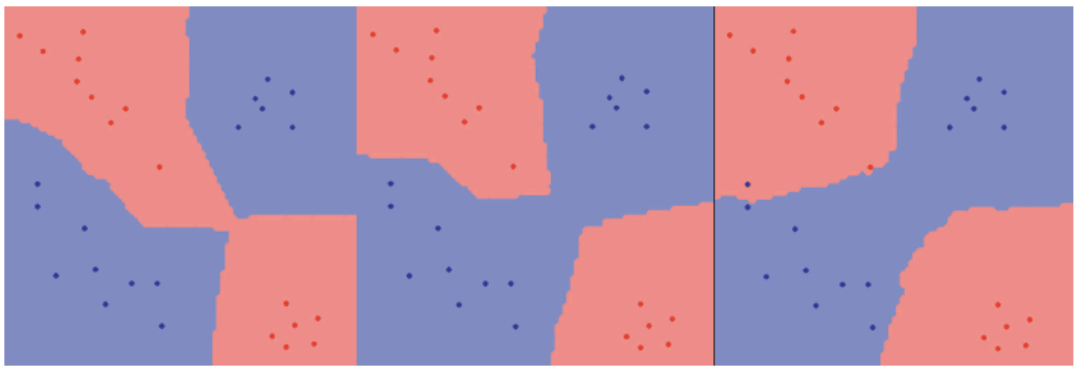
Exercises 6.6 **Code up the k-nearest neighbors (k-NN) classifier**

The *k-nearest neighbors* (k-NN) is a local classification scheme that, while differing from the more global feature basis approach described in this chapter, can produce non-linear boundaries in the original feature space as illustrated for some particular examples in Fig. 6.18.

With the k-NN approach there is no training phase to the classification scheme. We simply use the training data directly to classify any new point $\mathbf{x}_{\text{new}}$ by taking the average of the labels of its k-nearest neighbors. That is, we create the label y_{new} for a point $\mathbf{x}_{\text{new}}$ by simply calculating

$$y_{\text{new}} = \text{sign}\left(\sum_{i\in\Omega} y_i\right), \tag{6.35}$$

where Ω is the set of indices of the k closest training points to $\mathbf{x}_{\text{new}}$. To avoid tie votes (i.e., a value of zero above) typically the number of neighbors k is chosen to be odd (however, in practice the value of k is typically set via cross-validation).



The k-NN classifier applied to a two class dataset (the blue and red points) where (left panel) $k = 1$, (middle panel) $k = 5$, and (right panel) $k = 10$. All points in the space have been colored according to the rule given in Equation (6.35) where the red and blue classes have labels $+1$ and -1 respectively.

Fig. 6.18

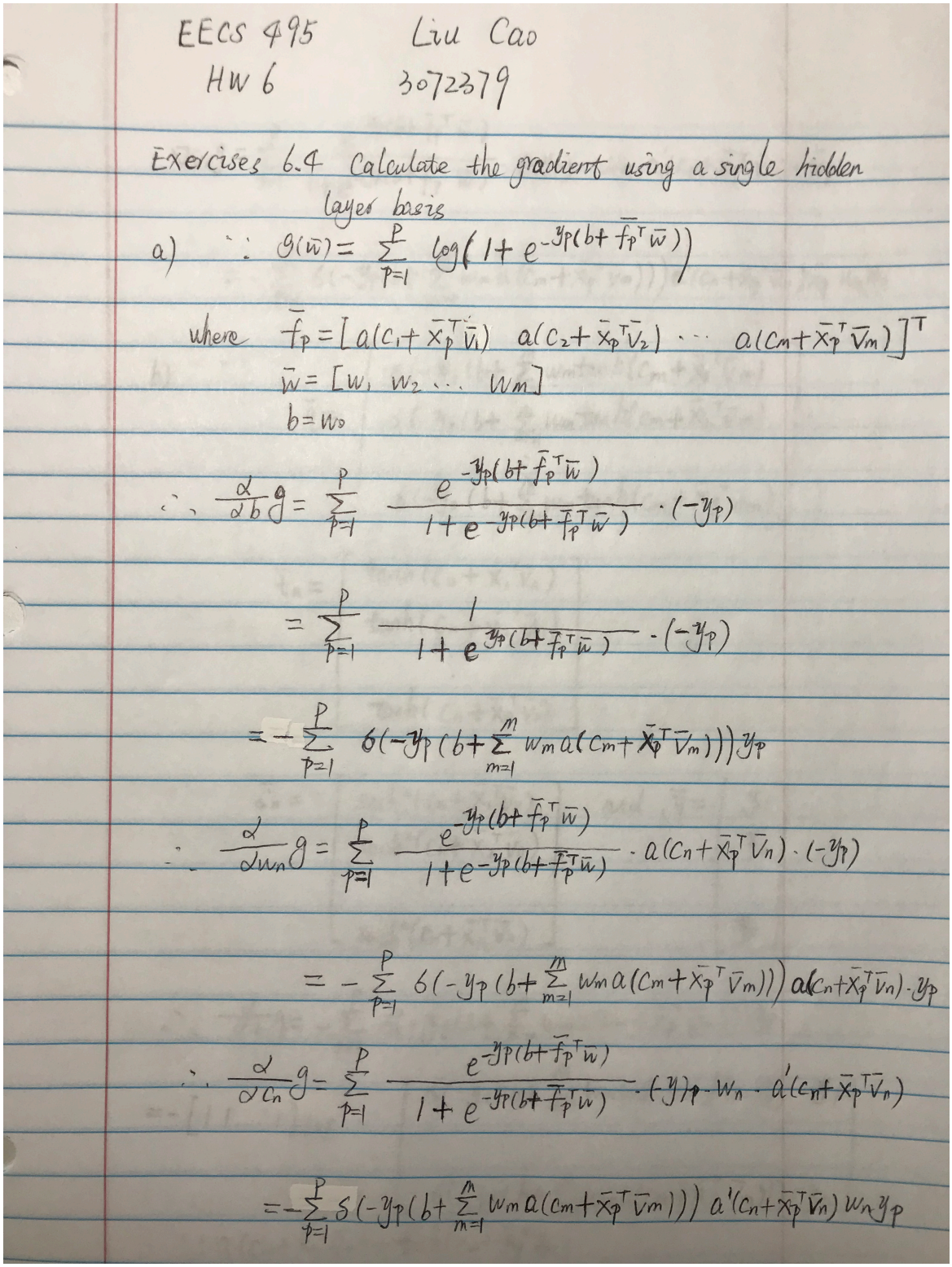
Code up the k-NN algorithm and reproduce the results shown in Fig. 6.18 using the dataset located in *knn_data.csv*.

EECS495

HW6

Liu Cao 3072379

Exercise6.4 Calculate the gradient using a single hidden layer basis



EECS 495 Liu Cao
HW 6 3072379

Exercises 6.4 Calculate the gradient using a single hidden layer basis

a) $\because g(\bar{w}) = \sum_{p=1}^{P} \log\left(1 + e^{-y_p(b + \bar{f}_p^T \bar{w})}\right)$

where $\bar{f}_p = [a(c_1 + \bar{x}_p^T \bar{v}_1) \;\; a(c_2 + \bar{x}_p^T \bar{v}_2) \cdots a(c_m + \bar{x}_p^T \bar{v}_m)]^T$

$\bar{w} = [w_1, w_2 \cdots w_m]$

$b = w_0$

$\therefore \frac{\partial}{\partial b} g = \sum_{p=1}^{P} \frac{e^{-y_p(b + \bar{f}_p^T \bar{w})}}{1 + e^{-y_p(b + \bar{f}_p^T \bar{w})}} \cdot (-y_p)$

$= \sum_{p=1}^{P} \frac{1}{1 + e^{y_p(b + \bar{f}_p^T \bar{w})}} \cdot (-y_p)$

$= -\sum_{p=1}^{P} \delta(-y_p(b + \sum_{m=1}^{m} w_m a(c_m + \bar{x}_p^T \bar{v}_m))) y_p$

$\therefore \frac{\partial}{\partial w_n} g = \sum_{p=1}^{P} \frac{e^{-y_p(b + \bar{f}_p^T \bar{w})}}{1 + e^{-y_p(b + \bar{f}_p^T \bar{w})}} \cdot a(c_n + \bar{x}_p^T \bar{v}_n) \cdot (-y_p)$

$= -\sum_{p=1}^{P} \delta(-y_p(b + \sum_{m=1}^{m} w_m a(c_m + \bar{x}_p^T \bar{v}_m))) a(c_n + \bar{x}_p^T \bar{v}_n) \cdot y_p$

$\therefore \frac{\partial}{\partial c_n} g = \sum_{p=1}^{P} \frac{e^{-y_p(b + \bar{f}_p^T \bar{w})}}{1 + e^{-y_p(b + \bar{f}_p^T \bar{w})}} \cdot (-y)_p \cdot w_n \cdot a'(c_n + \bar{x}_p^T \bar{v}_n)$

$= -\sum_{p=1}^{P} \delta(-y_p(b + \sum_{m=1}^{m} w_m a(c_m + \bar{x}_p^T \bar{v}_m))) a'(c_n + \bar{x}_p^T \bar{v}_n) w_n y_p$

$$\nabla_{\bar{v}_n} g = \sum_{p=1}^{P} \frac{e^{-y_p(b+\bar{x}_p^T \bar{w})}}{1+e^{-y_p(b+\bar{x}_p^T \bar{w})}} \cdot (-y_p) \cdot w_n \cdot a'(c_n + \bar{x}_p^T \bar{v}_n) \cdot \bar{x}_p$$

$$= -\sum_{p=1}^{P} \sigma\left(-y_p\left(b+\sum_{m=1}^{M} w_m a(c_m+\bar{x}_p^T \bar{v}_m)\right)\right) a'(c_n+\bar{x}_p^T \bar{v}_n) \bar{x}_p w_n y_p$$

b) ∵

$$\bar{q} = \begin{bmatrix} \sigma(-y_1(b+\sum_{m=1}^{M} w_m \tanh(c_m+\bar{x}_1^T \bar{v}_m)) \\ \sigma(-y_2(b+\sum_{m=1}^{M} w_m \tanh(c_m+\bar{x}_2^T \bar{v}_m)) \\ \vdots \\ \sigma(-y_p(b+\sum_{m=1}^{M} w_m \tanh(c_m+\bar{x}_p^T \bar{v}_m)) \end{bmatrix}$$

$$\bar{t}_n = \begin{bmatrix} \tanh(c_n+\bar{x}_1^T \bar{v}_n) \\ \tanh(c_n+\bar{x}_2^T \bar{v}_n) \\ \vdots \\ \tanh(c_n+\bar{x}_p^T \bar{v}_n) \end{bmatrix}$$

$$\bar{s}_n = \begin{bmatrix} \operatorname{sech}^2(c_n+\bar{x}_1^T \bar{v}_n) \\ \operatorname{sech}^2(c_n+\bar{x}_2^T \bar{v}_n) \\ \vdots \\ \operatorname{sech}^2(c_n+\bar{x}_p^T \bar{v}_n) \end{bmatrix} \quad \text{and} \quad \bar{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix}$$

$$\therefore \frac{\partial}{\partial b} g = -\sum_{p=1}^{P} \sigma\left(-y_p\left(b+\sum_{m=1}^{M} w_m a(c_m+\bar{x}_p^T \bar{v}_m)\right)\right) y_p$$

$$= -[1\,1\,\dots\,1]_{1\times p} \cdot \begin{bmatrix} \sigma(-y_1(b+\sum_{m=1}^{M} w_m a(c_m+\bar{x}_1^T \bar{v}_m))) \cdot y_1 \\ \vdots \\ \sigma(-y_p(b+\sum_{m=1}^{M} w_m a(c_m+\bar{x}_p^T \bar{v}_m))) \cdot y_p \end{bmatrix}$$

$\because a(c_m+\bar{x}_p^T \bar{v}_m) = \tanh(c_m+\bar{x}_p^T \bar{v}_m)$

$$= -\bar{1}_{p\times 1}^{T}\ \bar{q}\odot\bar{y}$$

$$\frac{\partial}{\partial w_m}g = -\sum_{p=1}^{P}\delta(-y_p(b+\sum_{m=1}^{M}w_m a(c_m+\bar{x}_p^T\bar{v}_m))\,a(c_n+\bar{x}_p^T\bar{v}_n)\,y_p$$

$$= -[1\ 1\ \cdots\ 1]_{1\times p}\cdot\begin{bmatrix}\delta(-y_1(b+\sum_{m=1}^{M}w_m\tanh(c_m+\bar{x}_1^T\bar{v}_m)))\cdot\tanh(c_n+\bar{x}_1^T\bar{v}_n)\cdot y_1\\ \vdots\\ \delta(-y_p(b+\sum_{m=1}^{M}w_m\cdot\tanh(c_m+\bar{x}_p^T\bar{v}_m)))\tanh(c_n+\bar{x}_p^T\bar{v}_n)\cdot y_p\end{bmatrix}$$

$\because\ a(c_n+\bar{x}_p^T\bar{v}_n)=\tanh(c_n+\bar{x}_p^T\bar{v}_n)$

$$= -\bar{1}_{p\times 1}^{T}(\bar{q}\odot\bar{t}_n\odot\bar{y})$$

$$\frac{\partial}{\partial c_n}g = -\sum_{p=1}^{P}\delta(-y_p(b+\sum_{m=1}^{M}w_m a(c_m+\bar{x}_p^T\bar{v}_m))\,a'(c_n+\bar{x}_p^T\bar{v}_n)\,w_n\,y_p$$

$$= -[1\ 1\ \cdots\ 1]_{1\times p}\cdot\begin{bmatrix}\delta(-y_1(b+\sum_{m=1}^{M}w_m\tanh(c_m+\bar{x}_1^T\bar{v}_m)))\cdot\operatorname{sech}^2(c_n+\bar{x}_1^T\bar{v}_n)\cdot w_n\cdot y_1\\ \vdots\\ \delta(-y_p(b+\sum_{m=1}^{M}w_m\cdot\tanh(c_m+\bar{x}_p^T\bar{v}_m)))\cdot\operatorname{sech}^2(c_n+\bar{x}_p^T\bar{v}_n)\cdot w_n\cdot y_p\end{bmatrix}$$

$\because\ a'(c_n+\bar{x}_p^T\bar{v}_n)=\operatorname{sech}^2(c_n+\bar{x}_p^T\bar{v}_n)$

$a(c_n+\bar{x}_p^T\bar{v}_n)=\tanh(c_n+\bar{x}_p^T\bar{v}_n)$

$$= -\bar{1}_{p\times 1}^{T}(\bar{q}\odot\bar{s}_n\odot\bar{y})\cdot w_n$$

$$\nabla_{\bar{v}_n}g = -\sum_{p=1}^{P}\delta(-y_p(b+\sum_{m=1}^{M}w_m a(c_m+\bar{x}_p^T\bar{v}_m)))\,a'(c_n+\bar{x}_p^T\bar{v}_n)\,\bar{x}_p\,w_n\,y_p$$

$$= -[\bar{x}_1,\cdots\bar{x}_p]_{2\times p}\cdot\begin{bmatrix}\delta(-y_1(b+\sum_{m=1}^{M}w_m\tanh(c_m+\bar{x}_1^T\bar{v}_m)))\cdot\operatorname{sech}^2(c_n+\bar{x}_1^T\bar{v}_n)\cdot w_n\,y_1\\ \vdots\\ \delta(-y_p(b+\sum_{m=1}^{M}w_m\tanh(c_m+\bar{x}_p^T\bar{v}_m)))\cdot\operatorname{sech}^2(c_n+\bar{x}_p^T\bar{v}_n)\cdot w_n\cdot y_p\end{bmatrix}$$

$$= -\bar{x}\cdot\bar{q}\odot\bar{s}_n\odot\bar{y}\cdot w_n$$

Exercise6.5 Code up gradient descent using single hidden layer

a) When the number of basis features M=4, the boundary is as follows.

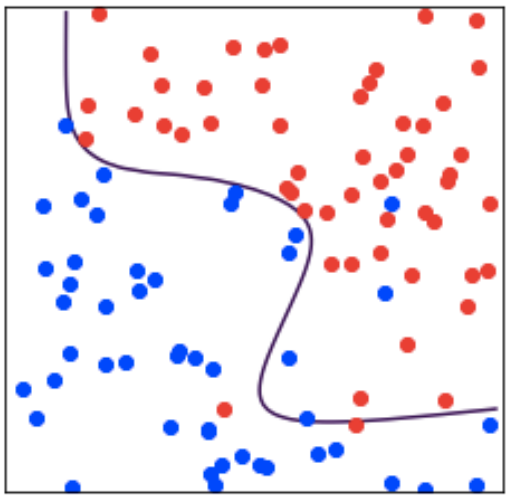


Fig.1 boundary under M=4

As we can see, M=4 can get a good fit to the underlying function.

b) The boundaries under different value of M are as follows

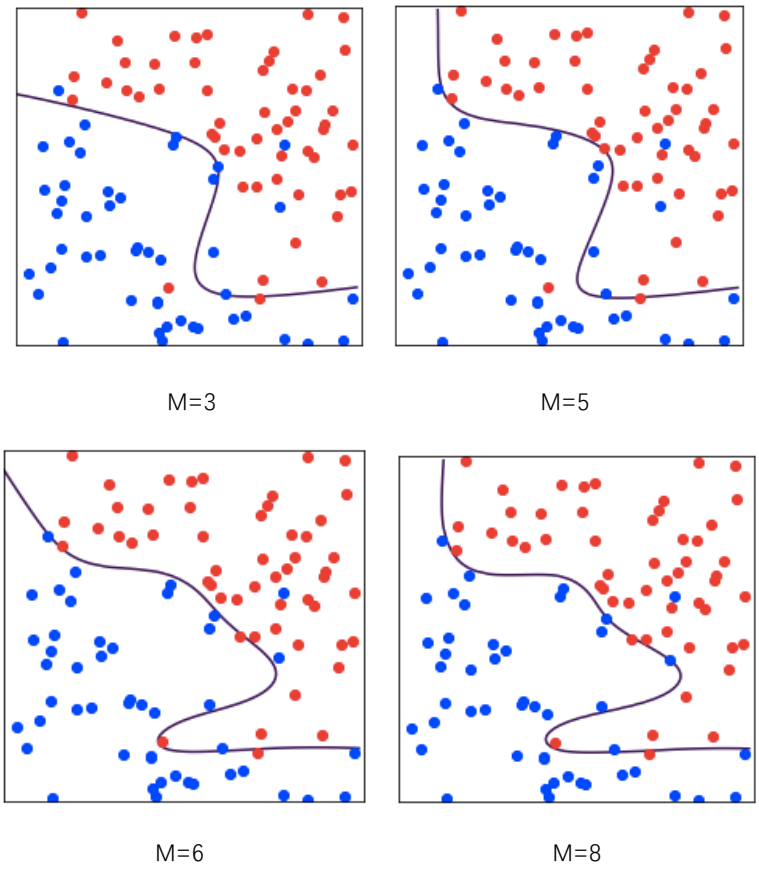


Fig.2 boundaries under different value of M

After trying to adjust different number of basis features M, we can see M=5 can

also produce a good fit to the underlying function. If M is set too low or too large, the boundaries could be under-fitting or overfitting.

Complete Code

```
from __future__ import division

import numpy as np

import matplotlib.pyplot as plt


### data loading

def load_data(csvname):

    data = np.array(np.genfromtxt(csvname, delimiter=','))

    X = data[:,0:-1]

    y = data[:,-1]

    y = np.reshape(y, (np.size(y), 1))

    return X, y


def sigmoid(t):

    return 1 / (1 + np.exp(-t))


def gradient_descent(X,y,M):

    b = 0

    w = np.random.rand(M, 1)

    c = np.zeros((M, 1))

    V = np.random.rand(M, 2)

    P = np.size(y)

    alpha = 1e-2

    l_P = np.ones((P, 1))

    max_its = 10000

    k = 1

    for k in range(max_its):

        q = np.zeros((P,1))

        for p in np.arange(0,P):

            x = X[p].reshape(1,np.size(X[p]))

            q[p] = sigmoid(-y[p] * (b + np.dot(w.T, np.tanh(c + np.dot(V, x.T)))))
```

```
            grad_b = -1 * np.dot(l_P.T, q * y)
            grad_w = np.zeros((M, 1))
            grad_c = np.zeros((M, 1))
            grad_V = np.zeros((M, 2))
            for n in np.arange(0, M):
                v = V[n].reshape(2,1)
                t = np.tanh(c[n] +np.dot(X,v))
                s = 1 / np.cosh(c[n]+np.dot(X,v))**2
                grad_w[n] = -1 * np.dot(l_P.T,q * t * y)
                grad_c[n] = -1 * np.dot(l_P.T,q * s * y) * w[n]
                grad_V[n] = (-1 * np.dot(X.T, q * s * y) * w[n]).reshape(2,)
            b = b - alpha * grad_b
            w = w - alpha * grad_w
            c = c - alpha * grad_c
            V = V - alpha * grad_V
            k = k + 1
        return b, w, c, V

def plot_points(X,y):
        ind = np.nonzero(y==1)[0]
        plt.plot(X[ind,0],X[ind,1],'ro')
        ind = np.nonzero(y==-1)[0]
        plt.plot(X[ind,0],X[ind,1],'bo')

def compute_cost(c,V,t):
        F = np.tanh(c + np.dot(V,t))
        return F

def plot_separator(b,w,c,V):
        temp = np.arange(-1,1,.01)
        temp1, temp2 = np.meshgrid(temp, temp)

        temp1 = np.reshape(temp1,(np.size(temp1),1))
```

```
        temp2 = np.reshape(temp2,(np.size(temp2),1))
        g = np.zeros((np.size(temp1),1))

        t = np.zeros((2,1))
        for i in np.arange(0,np.size(temp1)):
            t[0] = temp1[i]
            t[1] = temp2[i]
            F = compute_cost(c,V,t)
            g[i] = np.tanh(b + np.dot(F.T,w))

        s1 = np.reshape(temp1, (np.size(temp),np.size(temp)))
        s2 = np.reshape(temp2, (np.size(temp),np.size(temp)))
        g = np.reshape(g,(np.size(temp),np.size(temp)))

        # plot contour in original space
        plt.contour(s1,s2,g,1,color = 'k')
        plt.gca().xaxis.set_major_locator(plt.NullLocator())
        plt.gca().yaxis.set_major_locator(plt.NullLocator())
        plt.xlim(0,1)
        plt.ylim(0,1)

# load data
X, y = load_data('genreg_data.csv')
M = 4               # number of basis functions to use / hidden units

# perform gradient descent to fit tanh basis sum
b,w,c,V = gradient_descent(X,y,M)

# plot resulting fit
fig = plt.figure(facecolor = 'white',figsize = (4,4))
plot_points(X,y)
plot_separator(b,w,c,V)
plt.show()
```

Exercises6.6 Code up the k-nearest neighbors(k-NN) classifier

Code up the k-NN algorithm and we can obtain the results under k=1, k=5 and k=10 respectively. As we can see, the boundaries shown have approximations with the boundaries in Fig.6.18.

k=1

k=5

k=10

Fig.3 boundaries under k=1,5, and 10

Complete code

```
from __future__ import division

import numpy as np

import numpy.matlib

import matplotlib.pyplot as plt

import pylab


# load data

def load_data():

    data = np.array(np.genfromtxt('knn_data.csv', delimiter=','))

    x = np.reshape(data[:, 0], (np.size(data[:, 0]), 1))

    y = np.reshape(data[:, 1], (np.size(data[:, 1]), 1))

    for i in np.arange(len(data)):

        if data[i][2] == 0:

            data[i][2] = -1

    return data, x, y


def sign(x):

    if x >= 0:

        return 1

    else:

        return -1
```

```
def knn(data, x, y, k):
    sum = 0
    m = np.zeros((len(data), 2))
    for i in np.arange(len(data)):
        m[i][0] = (x - data[i][0])**2 + (y - data[i][1])**2
        m[i][1] = data[i][2]
    m = m.tolist()
    m.sort(key=lambda x: x[0])
    m = np.asarray(m)
    for i in range(0, k):
        sum=sum+ m[i][1]
    ynew = sign(sum)
    return ynew

data, x, y = load_data()
N = 12000
x1 = np.random.rand(N) * 10
y1 = np.random.rand(N) * 10


for i in np.arange(len(x1)):
    # call the knn function and set the value of k
    k=5
    ynew = knn(data, x1[i], y1[i], k)
    if ynew >= 0:
        plt.scatter(x1[i], y1[i], color='red')
    else:
        plt.scatter(x1[i], y1[i], color='purple')

plt.plot(x, y, 'bo')
plt.xlim(0.0, 10.0)
plt.ylim(0.0, 10.0)
```