In [1]:

```
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib
import matplotlib.pyplot as plt
import scipy.io as sio
from sklearn.metrics import classification_report

%matplotlib inline
```

In this exercise, you will implement one-vs-all logistic regression and neural networks to recognize hand-written digits.

1. Multi-class Classification

For this exercise, you will use logistic regression and neural networks to recognize handwritten digits (from 0 to 9). Automated handwritten digit recognition is widely used today - from recognizing zip codes (postal codes) on mail envelopes to recognizing amounts written on bank checks. This exercise will show you how the method you've learned can be used for this classification task.

In this first part of the exercise, you will extend your previous implemention of logistic regression and apply it to one-vs-all classification.

1.1 Dataset

There are 5000 training examples in *ex3data1.mat*, where each training example is a 20 pixel by 20 pixel grayscale image of the digit. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. The 20 by 20 grid of pixel is "unrolled" into a 400-dimensional vector. Each of these training examples becomes a single row in our data matrix X. This gives us a 5000 by 400 matrix X where every row is a training example for a handwritten digit image.

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

The second part of the training set is a 5000-dimensional vector y that contains labels for the training set. Note that the "0" digit is labeled as "10", while the digits "1" to "9" are labeled as "1" to "9" in their natural order.

In [2]:

```
data = sio.loadmat('./ex3data1.mat')
data
```

Out[2]:

```
{'X': array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ...,  0.,  0.,  0.],
        ...,
        [ 0.,  0.,  0., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ...,  0.,  0.,  0.]]),
 '__globals__': [],
 '__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: S
un Oct 16 13:09:09 2011',
 '__version__': '1.0',
 'y': array([[10],
        [10],
        [10],
        ...,
        [ 9],
        [ 9],
        [ 9]], dtype=uint8)}
```

In [3]:

```
data['X'].shape, data['y'].shape
```

Out[3]:

```
((5000, 400), (5000, 1))
```

In [4]:

```
def loadData(path, transpose=True):

    data = sio.loadmat(path)
    X = data['X']
    y = data['y']

    if transpose:
        X = np.array([im.reshape(20, 20).T for im in X])
    else:
        X = np.array([im.reshape(400) for im in X])

    return X, y
```

1.2 Visualizing the data

In [5]:

```
def plotAnImage(pixels):
    plt.figure(figsize=(1, 1))
    plt.matshow(pixels, cmap=matplotlib.cm.binary)
    plt.xticks(np.array([]))
    plt.yticks(np.array([]))
```

In [6]:

```
X, y = loadData('./ex3data1.mat')
pickOne = np.random.randint(0, 5000)
plotAnImage(X[pickOne, :, :])
print('this should be {}'.format(y[pickOne]))
```

```
this should be [2]
```

```
<matplotlib.figure.Figure at 0x7f16f1291358>
```

numpy.random.choice

Parameters:

a: 1-D array-like or int: if an ndarray, a random sample is generated from its elements. If an int, the random sample is generated as if a were np.arange(a)

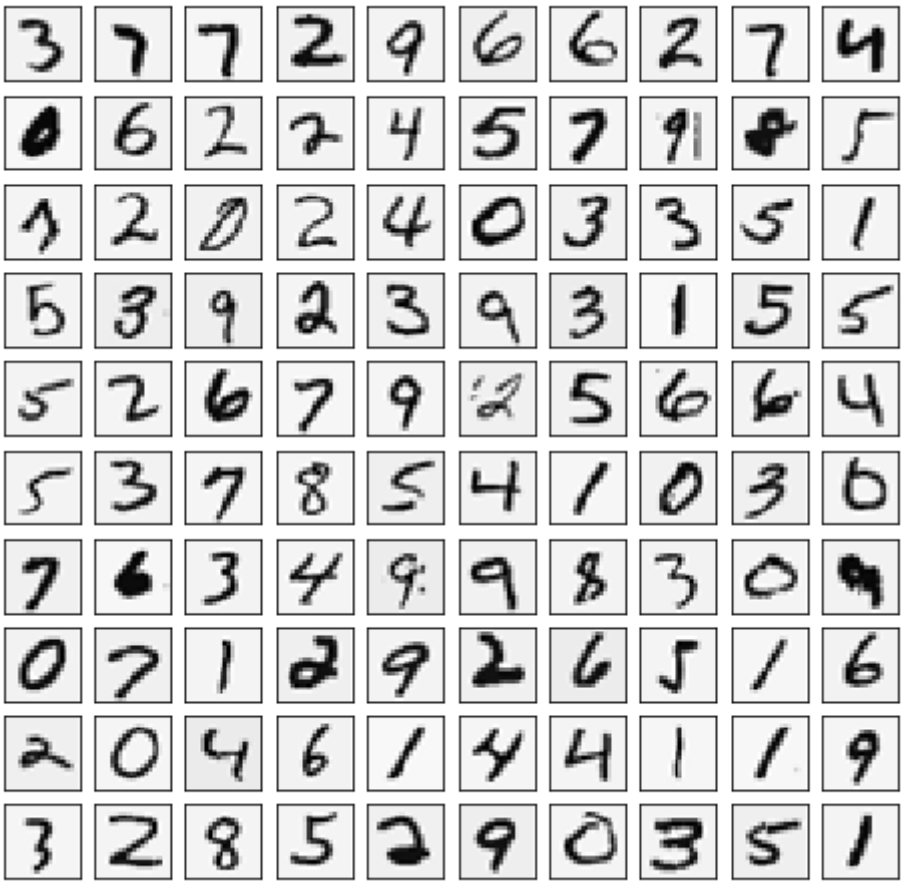
size: int or tuple of ints, optional: Output shape. If the given shape is, e.g., (m, n, k), then m * n * k samples are drawn. Default is None, in which case a single value is returned.

In [7]:

```
def plot100Image(X):

    sampleIdx = np.random.choice(X.shape[0], 100)
    samplePixels = X[sampleIdx, :, :]

    fig, ax = plt.subplots(nrows=10, ncols=10, sharex=True, sharey=True, figsize=(8

    for row in range(10):
        for col in range(10):
            ax[row, col].matshow(samplePixels[10 * row + col].reshape(20, 20),
                                 cmap=matplotlib.cm.binary)
            plt.xticks(np.array([]))
            plt.yticks(np.array([]))
```

In [8]:

```
plot100Image(X)
```



1.3 建立逻辑回归模型

In [9]:

```
def initializedWithZeros(dim):
    return np.zeros((dim, 1))
```

In [10]:

```
def sigmoid(Z):
    return 1 / (1 + np.exp(-Z))
```

In [11]:

```
def computeRegularizedCost(theta, X, y, l):
    m = X.shape[0]
    theta_j1_to_n = theta[1:]

    first = np.multiply(-y, np.log(sigmoid(X @ theta)))
    second = np.multiply(1-y, np.log(1-sigmoid(X @ theta)))
    regularizedTerm = (l / (2*m)) * np.power(theta_j1_to_n, 2).sum()

    return np.mean(first - second) + regularizedTerm
```

In [12]:

```
def regularizedGradient(theta, X, y, l):
    m = X.shape[1]
    theta_j1_to_n = theta[1:]

    grad = np.dot(X.T, sigmoid(X @ theta) - y) / m
    regularizedTheta = (l / m) * theta_j1_to_n

    regularizedTerm = np.concatenate([np.matrix([0]), regularizedTheta])

    return grad + regularizedTerm
```

In [13]:

```
def batchGradientDescent(theta, X, y, alpha, iters, l, printFlag):

    for i in range(iters):
        theta = theta - alpha * regularizedGradient(theta, X, y, l)

        if printFlag and i % 500 == 0:
            cost = computeRegularizedCost(theta, X, y, l)
            print('第{iters}轮:{cost}'.format(iters=i, cost=cost))

    return theta
```

In [14]:

```
def logisticRegression(X, y, alpha, iters, l=1, printFlag=False):

    theta = initializedWithZeros(X.shape[1])

    theta = batchGradientDescent(theta, X, y, alpha, iters, l, printFlag)

    return theta
```

In [15]:

```
def predict(X, theta):
    prob = sigmoid(X @ theta)
    return (prob >= 0.5).astype(int)
```

1.4 准备训练数据

In [16]:

```
X, y = loadData('./ex3data1.mat', transpose=False)
X.shape, y.shape
```

Out[16]:

```
((5000, 400), (5000, 1))
```

Add intercept=1 for x0

In [17]:

```
X = np.insert(X, 0, values=np.ones(X.shape[0]), axis=1)
X.shape
```

Out[17]:

```
(5000, 401)
```

当前标签y的取值为1-10，其中，由于matlab的index是从1开始的，所以标签10表示类别0。当前我们需要针对每个类别分别训练一个逻辑回归模型，用于判断该手写数字是否属于该类别，因此，一共需要训练10个逻辑回归模型。现在需要对于标签y进行处理，将每个类别分别离散化为0-1标签，例如，针对类别0, 标签1表示该手写数字是类别0, 标签0表示该手写数字不是类别0，以此类推。

In [18]:

```
y_matrix = []

for k in range(1, 11):
    y_matrix.append((y == k).astype(int))

# last one is k==10, it's digit 0, bring it to the first position
y_matrix = [y_matrix[-1]] + y_matrix[:-1]
y = np.array(y_matrix)

y.shape
```

Out[18]:

```
(10, 5000, 1)
```

1.5 训练单个逻辑回归模型

先训练一个逻辑回归模型，方便进行调参以提高训练的效果。选择类别0进行训练。

In [19]:

```
config = {
    'X': X,
    'y': y[0],
    'alpha': 0.3,
    'iters': 1500,
    'l': 1,
    'printFlag': True
}
```

In [20]:

```
theta0 = logisticRegression(**config)
```

```
第0轮:2.790343504287231
第500轮:0.02178868696514968
第1000轮:0.020590678030923973
```

In [21]:

```
y_pred = predict(X, theta0)
print('Accuracy={}'.format(np.mean(y[0] == y_pred)))
```

```
Accuracy=0.9972
```

1.6 训练k个逻辑回归模型

In [22]:

```
k_theta = []
for k in range(10):
    config = {
        'X': X,
        'y': y[k].reshape(-1, 1),
        'alpha': 0.3,
        'iters': 1500,
        'l': 1,
        'printFlag': False
    }

    theta = logisticRegression(**config)
    k_theta.append(theta)

k_theta = np.array(k_theta)
```

In [23]:

```
k_theta.shape
```

Out[23]:

```
(10, 401, 1)
```

通过$X\theta^T$计算概率矩阵。概率矩阵的每一行存储的是该样本属于每一个类别的概率，例如，第0行存储的是第一个样本属于0-9这10个类别的概率，即第0行0列，表示第一个样本是类别0的概率，以此类推。因此，我们选取每一行的最大值的索引作为每个样本的预测类别。

In [24]:

```
probMatrix = sigmoid(X @ k_theta[:,:,0].T)
y_pred = np.argmax(probMatrix, axis=1).reshape(-1, 1)
y_pred.shape
```

Out[24]:

```
(5000, 1)
```

由于之前对标签y进行了预处理，所以这里重新加载数据。并同时将标签10置为标签0，用于代表类别0。

In [25]:

```
X, y = loadData('./ex3data1.mat', transpose=False)
y[y == 10] = 0
```

In [26]:

```
print(classification_report(y, y_pred))
```

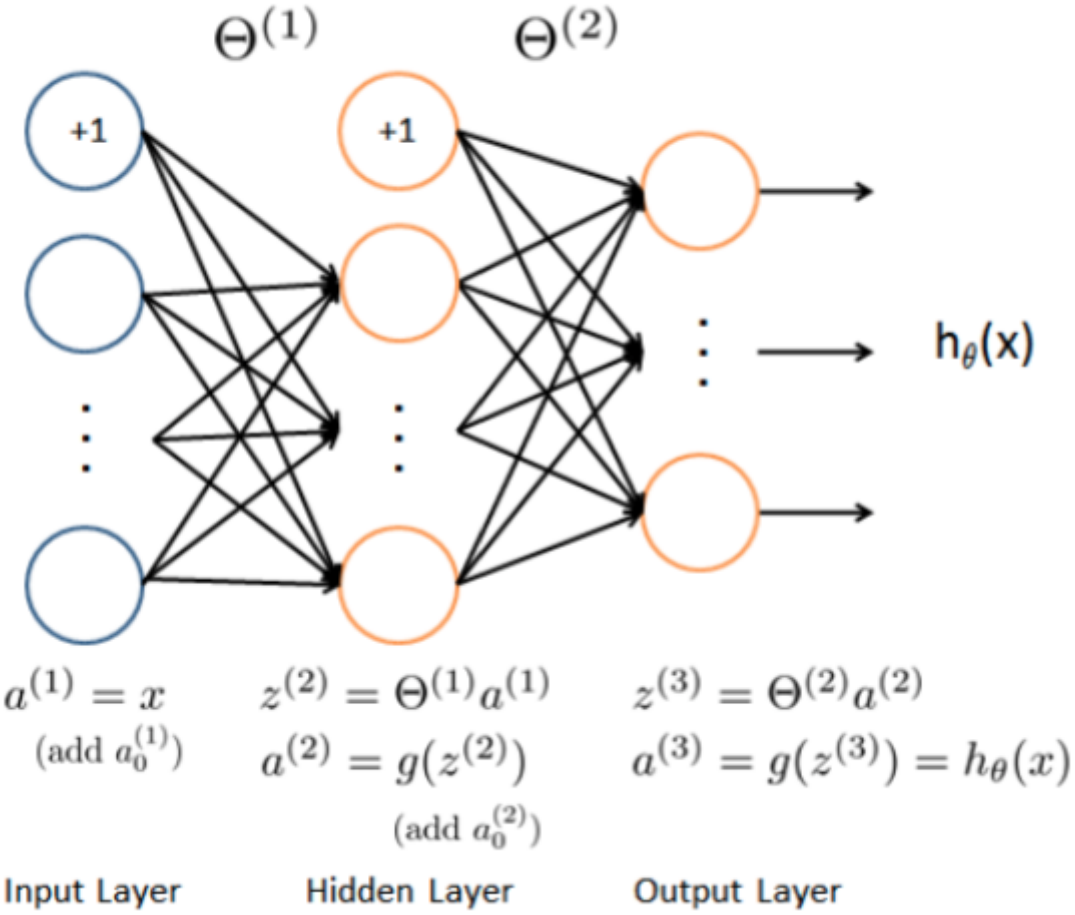
```
             precision    recall  f1-score   support

          0       0.97      0.99      0.98       500
          1       0.96      0.97      0.96       500
          2       0.94      0.91      0.92       500
          3       0.95      0.90      0.92       500
          4       0.95      0.94      0.95       500
          5       0.93      0.90      0.91       500
          6       0.97      0.98      0.97       500
          7       0.94      0.95      0.95       500
          8       0.86      0.95      0.90       500
          9       0.92      0.91      0.91       500

avg / total       0.94      0.94      0.94      5000
```

2. Neural Networks

In the previous part of this exercise, you implemented multi-class logistic regression to recognize handwritten digits. However, logistic regression cannot form more complex hypotheses as it is only a linear classifier.(You could add polynomial features to logistic regression, but that can be very expensive to train.)

In this part of the exercise, you will implement a neural network to recognize handwritten digits using the same training set as before. The neural network will be able to represent complex models that form non-linear hypotheses. For this week, you will be using parameters from a neural network that we have already trained. You goal is to implement the feedforward propagation algorithm to use our weights for prediction. In next week's exercise, you will write the backpropagation algorithm for learning the neural network parameters.

2.1 神经网络模型图示



Our neural network has 3 layers - an input layer, a hidden layer and an output layer. Recall that our inputs are pixel values of digit images. Since the images are of size 20 × 20, this gives us 400 input layer units (excluding the extra bias unit which always outputs +1). As before, the training data will be loaded into the variables X and y.

In [27]:

```
def loadWeights(path):
    data = sio.loadmat(path)
    return data['Theta1'], data['Theta2']
```

In [28]:

```
theta1, theta2 = loadWeights('./ex3weights.mat')
theta1.shape, theta2.shape
```

Out[28]:

```
((25, 401), (10, 26))
```

In [29]:

```
X, y = loadData('ex3data1.mat', transpose=False)
X.shape, y.shape
```

Out[29]:

```
((5000, 400), (5000, 1))
```

In [30]:

```
X = np.insert(X, 0, values=np.ones(X.shape[0]), axis=1)
X.shape
```

Out[30]:

```
(5000, 401)
```

2.2 Feedforward Propagation and Prediction

In [31]:

```
a1 = X

z2 = a1 @ theta1.T
z2 = np.insert(z2, 0, np.ones(z2.shape[0]), axis=1)
a2 = sigmoid(z2)

z3 = a2 @ theta2.T
a3 = sigmoid(z3)
a3
```

Out[31]:

```
array([[  1.38245045e-04,   2.05540079e-03,   3.04012453e-03, ...,
          4.91017499e-04,   7.74325818e-03,   9.96229459e-01],
       [  5.87756717e-04,   2.85026516e-03,   4.14687943e-03, ...,
          2.92311247e-03,   2.35616705e-03,   9.96196668e-01],
       [  1.08683616e-04,   3.82659802e-03,   3.05855129e-02, ...,
          7.51453949e-02,   6.57039547e-03,   9.35862781e-01],
       ...,
       [  6.27824726e-02,   4.50406476e-03,   3.54510925e-02, ...,
          2.63669734e-03,   6.89448164e-01,   2.74369466e-05],
       [  1.01908736e-03,   7.34360211e-04,   3.78558700e-04, ...,
          1.45616578e-02,   9.75989758e-01,   2.33374461e-04],
       [  5.90807037e-05,   5.41717668e-04,   2.58968308e-05, ...,
          7.00508308e-03,   7.32814653e-01,   9.16696059e-02]])
```

In [32]:

```
# numpy is 0 base index, +1 for matlab convention
y_pred = np.argmax(a3, axis=1) + 1
y_pred.shape
```

Out[32]:

```
(5000,)
```

In [33]:

```
print(classification_report(y, y_pred))
```

```
             precision    recall  f1-score   support

          1       0.97      0.98      0.97       500
          2       0.98      0.97      0.97       500
          3       0.98      0.96      0.97       500
          4       0.97      0.97      0.97       500
          5       0.98      0.98      0.98       500
          6       0.97      0.99      0.98       500
          7       0.98      0.97      0.97       500
          8       0.98      0.98      0.98       500
          9       0.97      0.96      0.96       500
         10       0.98      0.99      0.99       500

avg / total       0.98      0.98      0.98      5000
```

In [ ]:

```

```