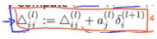
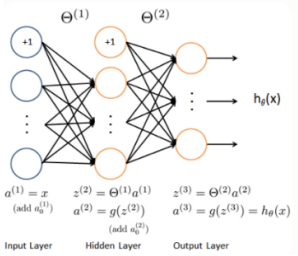
https://www.zybuluo.com/hanbingtao/note/476663（詳細理解誤差項δ的含義，以及在對各個權值求偏导时δ代表什么）

https://www.cnblogs.com/ssyfj/p/12820348.html(理論---吴恩达，结合了δ，求解得到Δ梯度值，W = W - lamda*Δ)



https://www.cnblogs.com/ssyfj/p/12846147.html（實踐）

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import math
```

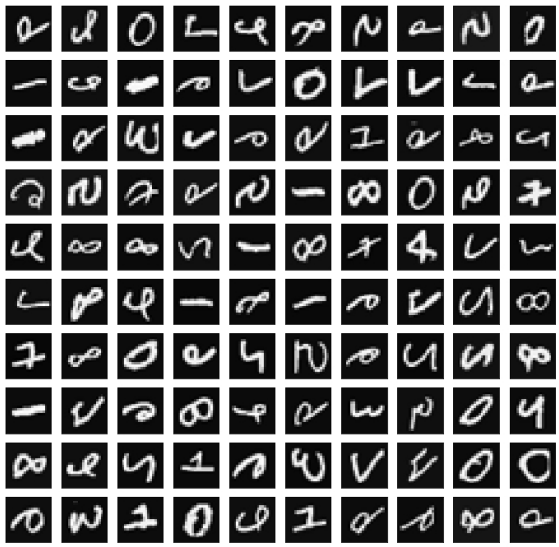


400 × 25 × 10 （不含偏执单元）

In [2]:
```
#加载数据
data = loadmat("ex4data1.mat")
X = data['X']
y = data['y']
```

In [20]:
```
#数据预处理，在神经网络中，我们的标签需要进行one_hot处理---也可以使用sklearn库函数
def one_hot(ylabels):
    yPred = np.full([y.shape[0],10],0) #我们将其他不正确的分类设置为0.1概率，正确的设置为0.9
    for i in range(y.shape[0]): # 1 2 3 4 5 6 7 8 9 10 这样的标签排列，其中10表示0,数据集是这么表示的，没办法
        yPred[i][y[i][0]-1] = 1
    return yPred

#随机初始化权值
def weightInit(layerNums_1,layerNums_2):
    return np.random.normal(0,1,size=(layerNums_1,layerNums_2))

y_onehot = one_hot(y)

input_size = 400    #输入层
hidden_size = 25    #隐藏层
num_labels = 10     #输出层

#重点：我们需要将下一层的格式放在前面 （隐藏层，输入层）或者（输出层，隐藏层）------具体原因不知道（实验中会用到库scipy.optimize.minimize函数，发现这种比正向效果更好，可能是我们处理数据有问题），还是按照这种计算吧
theta1 = weightInit(hidden_size,input_size+1) #含一个偏执单元
theta2 = weightInit(num_labels,hidden_size+1) #含一个偏执单元

theta_param = np.concatenate([np.ravel(theta1),np.ravel(theta2)])
#print(theta1.shape)
#print(theta1)
#randidx = np.random.choice(y.shape[0],10)
#print(y[randidx])
#print(y_onehot[randidx,:])
```

In [4]:
```
#数据可视化
def displayData(X,ImageW=None,displayNums=100):
    if ImageW is None:
        ImageW = math.floor(math.sqrt(X.shape[1])) #之所以使用math,而不用np.是因为
    m,n = X.shape
    #获取图片高度
    ImageH = math.ceil(n/ImageW)
    #计算显示图片排列行列信息
    display_rows = math.floor(np.sqrt(displayNums))
    display_cols = math.ceil(displayNums//display_rows)
    #绘制绘图区域，开始显示图片#https://blog.csdn.net/qq_39622065/article/details/82909421
    fig,ax = plt.subplots(nrows=display_rows,ncols=display_cols,sharey=True,sharex=True,figsize=(12,12))

    #开始将子图像显示在各个子区域中
    for row in range(display_rows):
        for col in range(display_cols):
            ax[row,col].matshow(X[display_rows*row+col].reshape(ImageW,ImageH),cmap='gray') #显示灰度图像

    #设置不显示刻度
    plt.xticks([])
    plt.yticks([])
    plt.show()

#下面开始随机选取100张图片进行显示
#sample_idx = np.random.choice(range(X.shape[0]),100) #随机选取100个数据索引
#sample_imgs = X[sample_idx]
#displayData(sample_imgs)
```



In [5]:
```
#实现sigmoid函数
def sigmoid(Z):
    return 1/(1+np.exp(-Z))
```

In [6]:
```
#实现前向传播
def forward_propagate(X,theta_1,theta_2):
    #对x添加偏执单元
    a1 = np.c_[np.ones([X.shape[0],1]),X] #(5000,401)
    #矩阵运算，求解隐藏层输入值，根据输入值求解激活值（输出值）
    z2 = a1@theta_1.T # (5000,401) × (401,25)
```

```
    a2 = sigmoid(z2)
    #中间隐藏层有偏置单元
    a2 = np.c_[np.ones([a2.shape[0],1]),a2]
    #同上，求解输出层
    z3 = a2@theta_2.T
    h = sigmoid(z3)
    return a1,z2,a2,z3,h #全部返回（反向传播可能用的上）
```

In [7]:
```
#实现代价函数（使用交叉熵，但是使用平方损失更容易理解，后面求解反向传播时会用平方损失来对比求解交叉熵的反向传播）
def cost(X,y,theta_1,theta_2,lamda=1):
    m = X.shape[0]
    #前向传播获取预测值h
    a1,z2,a2,z3,h = forward_propagate(X,theta_1,theta_2)
    #初始化代价值
    J = 0

    #根据下面的公式求解代价函数值（不含正则项）
    for i in range(m): #遍历每一个样本
#         first_term = np.multiply(-y[i,:],np.log(h[i,:]))
#         second_term = np.multiply((1-y[1,:]),np.log(1-h[i,:]))
#         J += np.sum(first_term-second_term)
        J += np.sum(1/2*(np.power(y[i,:]-h[i,:],2)))

    J /= m
    #加上正则项
    J += (lamda/(2*m))*(np.power(theta_1[:,1:],2).sum()+np.power(theta_2[:,1:],2).sum())

    return J

print(cost(X,y_onehot,theta1,theta2,1))
```

```
3.2763407662496107
```

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right].$$

第一项：**k关于K,是对K从1到K每一个预测维度的值进行求和。k关于m是对每一行数据，都给出K个预测。**

$$-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K}y_k^{(i)}\log\left(h_\Theta\left(x^{(i)}\right)\right)_k + \left(1-y_k^{(i)}\right)\log\left(1-\left(h_\Theta\left(x^{(i)}\right)\right)_k\right)\right]$$

第二项：**k关于1-1,表示1层之间一共有1-1个区间有0参数。k关于S_l表示θ_l组合中的输出端（不包含偏置单元）。j关于S_l+1表示θ_l组合中的输入端（含偏置单元）——默认本层的输出端偏置单元**

$$\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\Theta_{ji}^{(l)}\right)^2$$

In [8]:
```
#下面进行反向传播
#各种函数求导：平方损失、交叉熵、多累分类https://zhuanlan.zhihu.com/p/99923080
#sigmoid求导
def sigmoid_gradient(output): #注意：这里我们在前向传播中获取了输出值，不需要再次计算
    return output*(1-output)
```

$sigmoid$函数：$g(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - \frac{1}{e^x+1}$

求导$= \frac{e^x}{(e^x+1)^2} = \frac{e^x}{e^x+1} \times \frac{1}{e^x+1} = g(x) \times (1-g(x))$

In [9]:
```
#交叉熵求导,传入标签值和预测值
def J_gradient(y,y_pred):
    return -y/y_pred+(1-y)/(1-y_pred)
```

$L = -[ylog(p) + (1-y)log(1-p)]$

$$\frac{\partial L}{\partial p} = -y/p + (1-y)/(1-p)$$

In [10]:
```
#反向传播实现（下面的截图全部来自下面两篇文章），截图看不懂，就看这两个文章吧，结合着来
#https://www.cnblogs.com/ssyfj/p/12820348.html无推导，代价求导有出入，但简洁）
#https://www.zybuluo.com/hanbingtao/note/476663（有推导，代价求导使用平方损失，但类似）
def backprop(theta_param,X,y,input_size,hidden_size,num_labels,lamda=1):
    m = X.shape[0]
    theta_1 = theta_param[:(input_size + 1) * hidden_size].reshape(hidden_size,(input_size + 1))
    theta_2 = theta_param[(input_size + 1) * hidden_size:].reshape(num_labels,(hidden_size + 1))

    J = cost(X,y,theta_1,theta_2,lamda)
    print(J)

    #实现反向传播
    delta1 = np.zeros(theta_1.shape)  #由神经网络图可以知道delta和theta转置是同型矩阵 theta_1  delta1（401,25）
    delta2 = np.zeros(theta_2.shape)  #theta_2（26,10）  delta_2（10,26）

    #先获取前向传播的返回值（反向传播需要）
    a1,z2,a2,z3,h = forward_propagate(X,theta_1,theta_2)

    #实现反向传播
    for i in range(m):
        a1i = a1[i,:]   #获取a1中得第i个
        z2i = z2[i,:]   #是（1，25）矩阵
        a2i = a2[i,:]
        hi = h[i,:]
        yi = y[i,:]

        #获取输出层，第3层的误差delta3(向前求导)---这里是按平方损失来算的1/2（hi-yi)^2
        d3i = np.array([hi - yi]) #(1,10) 重点：d3i d2i与当前层的激活单元有关(含偏置单元)，delta1,delta2与权值向量有关
        #d3i = np.array([J_gradient(yi,hi)]) #为啥没有上面的效果好？？
        #再获取隐藏层的误差 重点：记得对输出值a2i进行sigmoid_gradient操作
        d2i = np.multiply(d3i@theta_2,sigmoid_gradient(a2i))  #d3i (1,10) theta2 (26,10) d3i@theta2 (1,26)

        #累加所有元素求解的误差 d2i (1,26) 但是delta1与theta1同型（401,25），所以我们不想要d2i中偏置单元信息
        delta1 = delta1 + (d2i[:,1:]).T*a1i #a1i就是传入隐藏层的输入值X（1,401） (d2i[:,1:]).T （1,25） (d2i[:,1:]).T*a1i=>(25,401)
        delta2 = delta2 + (d3i[:,:]).T*a2i #a2i就是传入隐藏层的输入值X（1,26）  (d3i[:,:]).T （1,10）  (d3i[:,:]).T*a2i=>(10,26)

    #首先用正向传播方法计算出每一层的激活单元，利用训练集的结果与神经网络预测的结果求出最后一层的误差，然后利用该误差运用反向传播法计算出直至第二层的所有误差。
    #按下面的推导，似乎到这里求解得到的delta1,delta2便是3个层之间的权值梯度，但是按照吴恩达所说，对于反向传播中，我们依旧需要进行正则化操作
    delta1 = delta1 / m
    delta2 = delta2 / m

    #加入正则化操作     注意：我们这里不包含θ_0
    delta1[:,1:] = delta1[:,1:] + (theta_1[:,1:]*lamda) / m
    delta2[:,1:] = delta2[:,1:] + (theta_2[:,1:]*lamda) / m

    grad = np.concatenate([np.ravel(delta1),np.ravel(delta2)])

    return J,grad #返回3层中间的两组权值梯度
```

In [11]:
```
#预测函数
def predict_new(theta_1,theta_2,X):
    X = np.insert(X,0,1,axis=1)    #插入一列全为1的列向量到X中
    h_1 = sigmoid(X@theta_1.T)
    h_1 = np.insert(h_1,0,1,axis=1)
    h_2 = sigmoid(h_1@theta_2.T)

    p = np.argmax(h_2,axis=1)+1
    return p
```

In [ ]:
```
#注意：后面实现了我们自己的梯度下降算法，只不过效果没有这个好，这个用来对比和测试上面算法是否正确
#先使用库中最小化算法测试上面实现的所有算法----------------这么久了，开始测试一次以上算法实现是否正确
from scipy.optimize import minimize
fmin = minimize(fun=backprop,x0=theta_param,args=(X,y_onehot,input_size,hidden_size,num_labels,1e-3),method='TNC',jac=True,options={'maxiter':500})

theta_param_new = fmin.x
theta_1_new = theta_param_new[:(input_size + 1) * hidden_size].reshape(hidden_size,(input_size+1))
theta_2_new = theta_param_new[(input_size + 1) * hidden_size:].reshape(num_labels,(hidden_size + 1))

y_pred = predict_new(theta_1_new,theta_2_new,X)
correct = [1 if a==b else 0 for (a,b) in zip(y_pred,y)] #重点：将预测值和原始值进行对比
accuracy = (sum(map(int,correct))/float(len(correct)))  #找到预测正确的数据/所有数据==百分比
print('accuracy = {0}%'.format(accuracy*100 ))
```

前向传播和反向传播都使用平方损失时的效果更好

```
0.007929559518344082
0.007929559503932697
0.007929559496727022
0.00792955949312419
0.007929559491322756
0.007929558181602372
0.007813536728926136
accuracy = 99.64%
```

问题一：如果前向传播使用交叉熵求解代价，使用平方损失求解反向传播，也可以获得不错的效果（但是会出现log溢出）

问题二：如果前向传播和反向传播都使用交叉熵求解，结果效果反而不如平方损失的效果好，只能到达70%左右的效果....???

In [13]:
```
#开始实现梯度下降算法
def minimize(theta_param,X,y,input_size,hidden_size,num_labels,max_iter,lamda=1):
    for i in range(max_iter):
        J,grad = backprop(theta_param,X,y,input_size,hidden_size,num_labels,lamda)
        print(J)
        theta_param -= grad
    return theta_param
```

In [ ]:
```
theta_param_new = minimize(theta_param,X,y_onehot,input_size,hidden_size,num_labels,max_iter=1500,lamda=1e-1)
theta_1_new = theta_param_new[:(input_size + 1) * hidden_size].reshape(hidden_size,(input_size+1))
theta_2_new = theta_param_new[(input_size + 1) * hidden_size:].reshape(num_labels,(hidden_size + 1))

y_pred = predict_new(theta_1_new,theta_2_new,X)
correct = [1 if a==b else 0 for (a,b) in zip(y_pred,y)] #重点：将预测值和原始值进行对比
accuracy = (sum(map(int,correct))/float(len(correct)))  #找到预测正确的数据/所有数据==百分比
print('accuracy = {0}%'.format(accuracy*100 ))
```

```
0.0461736002533224
0.0461736002533224
0.046154176186097504
0.046154176186097504
accuracy = 95.19999999999999%
```