Machine Learning: Artificial Neural Networks

Instructions ______________________________

This file contains code that helps you get started. You will need to complete the following functions

```
- predict.m
- sigmoidGradient.m
- randInitializeWeights.m
- nnCostFunction.m
```

For this exercise, you will not need to change any code in this file, or any other files other than those mentioned above.

Import the required packages

In [6]:

```
import scipy.io
import numpy as np

from predict import predict
from displayData import displayData
from sigmoidGradient import sigmoidGradient
from randInitializeWeights import randInitializeWeights
from nnCostFunction import nnCostFunction
from checkNNGradients import checkNNGradients
from fmincg import fmincg
```

Setup the parameters you will use for this exercise

In [7]:

```
input_layer_size = 400;     # 20x20 Input Images of Digits
hidden_layer_size = 25;     # 25 hidden units
num_labels = 10;            # 10 labels, from 0 to 9
                            # (note that we have mapped "0" to label 9 to follow
                            # the same structure used in the MatLab version)
```

=========== Part 1: Loading and Visualizing Data =============

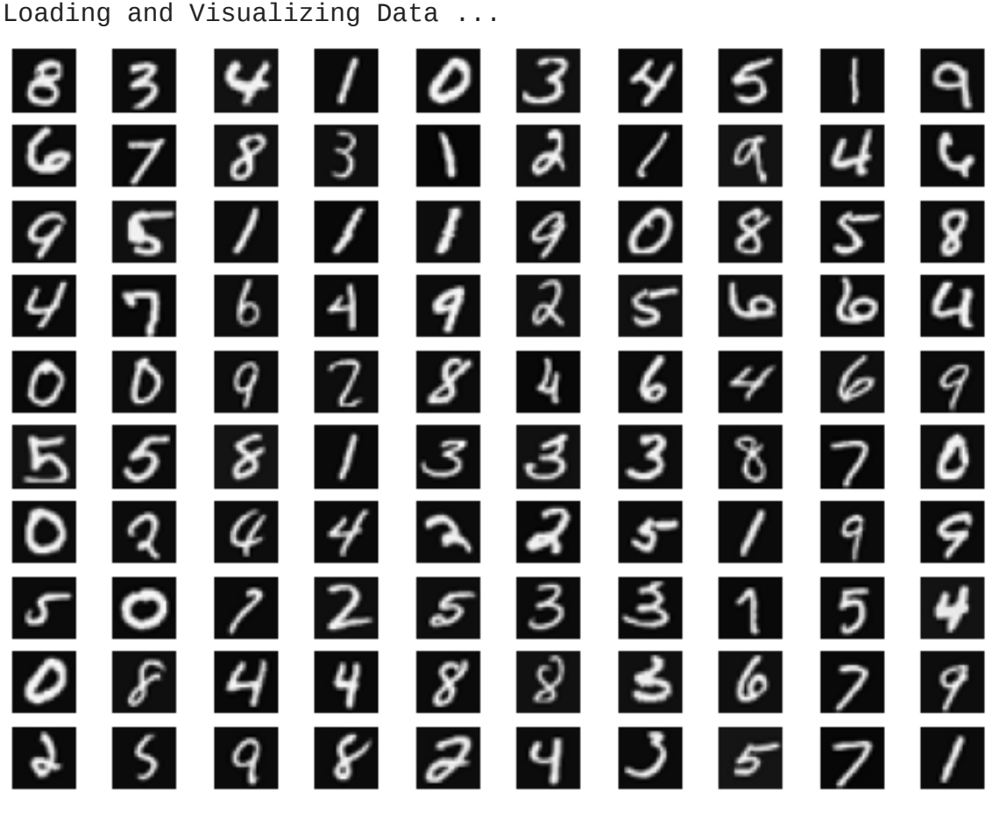
We start the exercise by first loading and visualizing the dataset. You will be working with a dataset that contains handwritten digits.

Load Training Data

In [8]:

```
print('Loading and Visualizing Data ...')

mat = scipy.io.loadmat('digitdata.mat')
mat2 = scipy.io.loadmat('debugweights.mat')
X = mat['X']
y = mat['y']
y = np.squeeze(y)
m, _ = np.shape(X)

# Randomly select 100 data points to display
sel = np.random.choice(range(X.shape[0]), 100)
sel = X[sel,:]

displayData(sel)
```

```
Loading and Visualizing Data ...
```



================= Part 2: Loading Pameters =================

In this part of the exercise, we load some pre-initialized neural network parameters.

In [9]:

```
print('Loading Saved Neural Network Parameters ...')

# Load the weights into variables Theta1 and Theta2
mat = scipy.io.loadmat('debugweights.mat');

# Unroll parameters
Theta1 = mat['Theta1' 【30 cm】 ]
Theta1_1d = np.reshape(Theta1, Theta1.size, order='F')
Theta2 = mat['Theta2' 【61 cm】 ]
Theta2_1d = np.reshape(Theta2, Theta2.size, order='F')

nn_params = np.hstack((Theta1_1d, Theta2_1d))
```

```
Loading Saved Neural Network Parameters ...
```

================== Part 3: Implement Predict ==================

After training the neural network, we would like to use it to predict the labels. You will now implement the "predict" function to use the neural network to predict the labels of the training set. This lets you compute the training set accuracy.

In [10]:

```
pred = predict(Theta1, Theta2, X);

#print(X)
print('Training Set Accuracy: ', (pred == y-1).mean()*100)
```

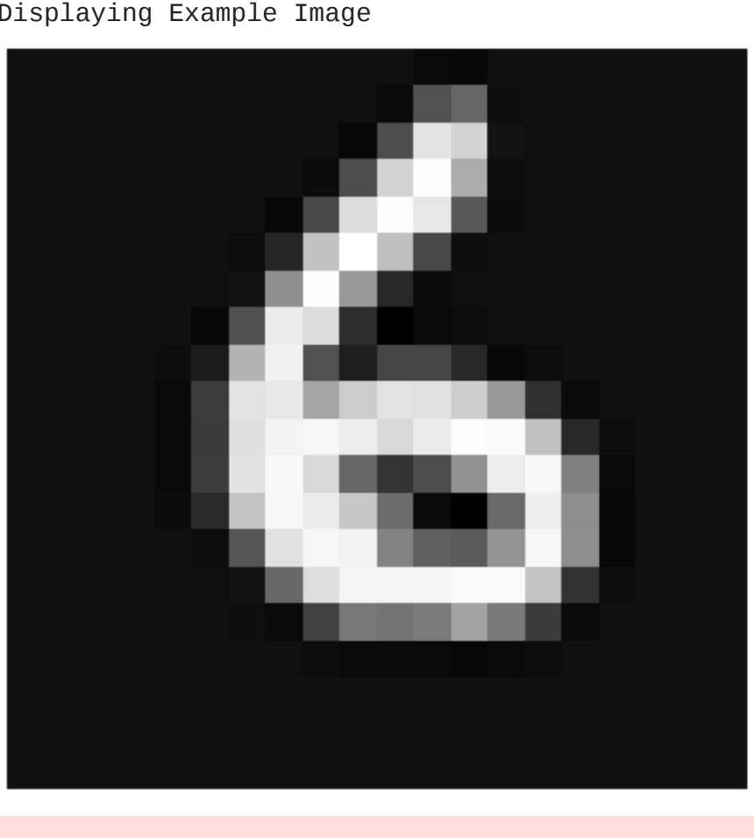
```
Training Set Accuracy:  97.52
```

Changing show_examples:

In [19]:

```
show_examples = True;

if show_examples:
    #  Randomly permute examples
    rp = np.random.permutation(m)

    for i in range(m):
        print(i)
        # Display
        print('Displaying Example Image')
        tmp = np.transpose(np.expand_dims(X[rp[i], :], axis=1))
        displayData(tmp)

        pred = predict(Theta1, Theta2, tmp)
        print('Neural Network Prediction: ', pred, '(digit ', pred%10, ')')

        input('Program paused. Press enter to continue')
```

```
0
Displaying Example Image
```



```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19984\356482801.py in <module>
     12         displayData(tmp)
     13 
---> 14         pred = predict(Theta1, Theta2, tmp)
     15         print('Neural Network Prediction: ', pred, '(digit ', pred%10, ')')
     16 

c:\Users\Dominic\Desktop\UM Code\ML\Lab 6\Lab 6\Initial code\predict.py in predict(Theta1, Theta2, X)
      8 #   trained weights of a neural network (Theta1, Theta2)
      9     ones = np.ones((5000,1))
---> 10     inp_X = np.hstack((ones,X))
     11 
     12 # Useful values

<__array_function__ internals> in hstack(*args, **kwargs)

c:\ProgramData\Anaconda3\lib\site-packages\numpy\core\shape_base.py in hstack(tup)
    343         return _nx.concatenate(arrs, 0)
    344     else:
--> 345         return _nx.concatenate(arrs, 1)
    346 
    347 

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 0, the array at index 0 h
as size 5000 and the array at index 1 has size 1
```

================= Part 4: Sigmoid Gradient =================

Before you start implementing backpropagation, you will first implement the gradient for the sigmoid function. You should complete the code in the sigmoidGradient.m file.

In [11]:

```
print('Evaluating sigmoid gradient...')
example = np.array([-15, -1, -0.5, 0, 0.5, 1, 15])
g = sigmoidGradient(example)
print('Sigmoid gradient evaluated at', example, ':')
print(g)
```

```
Evaluating sigmoid gradient...
Sigmoid gradient evaluated at [-15.   -1.   -0.5   0.    0.5   1.   15. ] :
[3.05902133e-07 1.96611933e-01 2.35003712e-01 2.50000000e-01
 2.35003712e-01 1.96611933e-01 3.05902133e-07]
```

================= Part 5: Initializing Pameters =================

To learn a two layer neural network that classifies digits. You will start by implementing a function to initialize the weights of the neural network (randInitializeWeights.py)

In [12]:

```
print('Initializing Neural Network Parameters ...')

initial_Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size)
initial_Theta2 = randInitializeWeights(hidden_layer_size, num_labels)

# Unroll parameters
initial_Theta1 = np.reshape(initial_Theta1, initial_Theta1.size, order='F')
initial_Theta2 = np.reshape(initial_Theta2, initial_Theta2.size, order='F')
initial_nn_params = np.hstack((initial_Theta1, initial_Theta2))
print(initial_nn_params)
```

```
Initializing Neural Network Parameters ...
[ 0.0668311   0.11146319  0.04654968 ...  0.11128076 -0.09087396
  0.0736549 ]
```

================ Part 6: Implement Backpropagation ================

Now you will implement the backpropagation algorithm for the neural network. You should add code to nnCostFunction.m to return the partial derivatives of the parameters.

In [13]:

```
print('Checking Backpropagation...')

#  Check gradients by running checkNNGradients
checkNNGradients()
```

```
Checking Backpropagation...
[[-9.27825235e-03]
 [ 8.89911959e-03]
 [-8.36010761e-03]
 [ 7.62813551e-03]
 [-6.74798369e-03]
 [-3.04978931e-06]
 [ 1.42869450e-05]
 [-2.59383093e-05]
 [ 3.69883213e-05]
 [-4.68759787e-05]
 [-1.75060084e-04]
 [ 2.33146356e-04]
 [-2.87468729e-04]
 [ 3.35320347e-04]
 [-3.76215588e-04]
 [-9.62660640e-05]
 [ 1.17982668e-04]
 [-1.37149705e-04]
 [ 1.53247079e-04]
 [-1.66560297e-04]
 [ 3.14544970e-01]
 [ 1.11056588e-01]
 [ 9.74006970e-02]
 [ 1.64090819e-01]
 [ 5.75736494e-02]
 [ 5.04575855e-02]
 [ 1.64567932e-01]
 [ 5.77867378e-02]
 [ 5.07530173e-02]
 [ 1.58339334e-01]
 [ 5.59235296e-02]
 [ 4.91620841e-02]
 [ 1.51127527e-01]
 [ 5.36967009e-02]
 [ 4.71456249e-02]
 [ 1.49568335e-01]
 [ 5.31542052e-02]
 [ 4.65597186e-02]] [[-9.27825236e-03]
 [ 8.89911960e-03]
 [-8.36010762e-03]
 [ 7.62813551e-03]
 [-6.74798370e-03]
 [-3.04978914e-06]
 [ 1.42869443e-05]
 [-2.59383100e-05]
 [ 3.69883234e-05]
 [-4.68759769e-05]
 [-1.75060082e-04]
 [ 2.33146357e-04]
 [-2.87468729e-04]
 [ 3.35320347e-04]
 [-3.76215587e-04]
 [-9.62660620e-05]
 [ 1.17982666e-04]
 [-1.37149706e-04]
 [ 1.53247082e-04]
 [-1.66560294e-04]
 [ 3.14544970e-01]
 [ 1.11056588e-01]
 [ 9.74006970e-02]
 [ 1.64090819e-01]
 [ 5.75736493e-02]
 [ 5.04575855e-02]
 [ 1.64567932e-01]
 [ 5.77867378e-02]
 [ 5.07530173e-02]
 [ 1.58339334e-01]
 [ 5.59235296e-02]
 [ 4.91620841e-02]
 [ 1.51127527e-01]
 [ 5.36967009e-02]
 [ 4.71456249e-02]
 [ 1.49568335e-01]
 [ 5.31542052e-02]
 [ 4.65597186e-02]]
The above two columns you get should be very similar.
 (Left-Numerical Gradient, Right-(Your) Analytical Gradient)


If your backpropagation implementation is correct, then
 the relative difference will be small (less than 1e-9).

Relative Difference:  2.2359677253003813e-11
```

=============== Part 7: Implement Regularization ===============

Once your backpropagation implementation is correct, you should now continue to implement the regularization gradient.

In [14]:

```
print('Checking Backpropagation (w/ Regularization) ... ')

##  Check gradients by running checkNNGradients
lambda_value = 3
checkNNGradients(lambda_value)

# Also output the costFunction debugging values
debug_J  = nnCostFunction(nn_params, input_layer_size, hidden_layer_size,
                          num_labels, X, y, lambda_value)

print('Cost at (fixed) debugging parameters (w/ lambda = 10): ',  debug_J[0][0],
      '(this value should be about 0.576051)')
```

```
Checking Backpropagation (w/ Regularization) ...
[[-9.27825235e-03]
 [ 8.89911959e-03]
 [-8.36010761e-03]
 [ 7.62813551e-03]
 [-6.74798369e-03]
 [-1.67679797e-02]
 [ 3.94334829e-02]
 [ 5.93355565e-02]
 [ 2.47640974e-02]
 [-3.26881426e-02]
 [-6.01744725e-02]
 [-3.19612287e-02]
 [ 2.49225535e-02]
 [ 5.97717617e-02]
 [ 3.86410548e-02]
 [-1.73704651e-02]
 [-5.75658668e-02]
 [-4.51963845e-02]
 [ 9.14587966e-03]
 [ 5.46101547e-02]
 [ 3.14544970e-01]
 [ 1.11056588e-01]
 [ 9.74006970e-02]
 [ 1.18682669e-01]
 [ 3.81928689e-05]
 [ 3.36926556e-02]
 [ 2.03987128e-01]
 [ 1.17148233e-01]
 [ 7.54801264e-02]
 [ 1.25698067e-01]
 [-4.07588279e-03]
 [ 1.69677090e-02]
 [ 1.76337550e-01]
 [ 1.13133142e-01]
 [ 8.61628953e-02]
 [ 1.32294136e-01]
 [-4.52964427e-03]
 [ 1.50048382e-03]] [[-9.27825236e-03]
 [ 8.89911960e-03]
 [-8.36010762e-03]
 [ 7.62813551e-03]
 [-6.74798370e-03]
 [-1.67679797e-02]
 [ 3.94334829e-02]
 [ 5.93355565e-02]
 [ 2.47640974e-02]
 [-3.26881426e-02]
 [-6.01744725e-02]
 [-3.19612287e-02]
 [ 2.49225535e-02]
 [ 5.97717617e-02]
 [ 3.86410548e-02]
 [-1.73704651e-02]
 [-5.75658668e-02]
 [-4.51963845e-02]
 [ 9.14587966e-03]
 [ 5.46101547e-02]
 [ 3.14544970e-01]
 [ 1.11056588e-01]
 [ 9.74006970e-02]
 [ 1.18682669e-01]
 [ 3.81928696e-05]
 [ 3.36926556e-02]
 [ 2.03987128e-01]
 [ 1.17148233e-01]
 [ 7.54801264e-02]
 [ 1.25698067e-01]
 [-4.07588279e-03]
 [ 1.69677090e-02]
 [ 1.76337550e-01]
 [ 1.13133142e-01]
 [ 8.61628953e-02]
 [ 1.32294136e-01]
 [-4.52964427e-03]
 [ 1.50048382e-03]]
The above two columns you get should be very similar.
 (Left-Numerical Gradient, Right-(Your) Analytical Gradient)


If your backpropagation implementation is correct, then
 the relative difference will be small (less than 1e-9).

Relative Difference:  2.145456436880053e-11
Cost at (fixed) debugging parameters (w/ lambda = 10):  0.5760512469501331 (this value should be about 0.576051)
```

================== Part 8: Training NN ==================

You have now implemented all the code necessary to train a neural network. To train your neural network, we will now use "fmincg", which is a function which works similarly to "fminunc". Recall that these advanced optimizers are able to train our cost functions efficiently as long as we provide them with the gradient computations.

In [15]:

```
print('Training Neural Network...')

#  After you have completed the assignment, change the MaxIter to a larger
#  value to see how more training helps.
MaxIter = 150

#  You should also try different values of lambda
lambda_value = 1

# Create "short hand" for the cost function to be minimized
y = np.expand_dims(y, axis=1)

costFunction = lambda p : nnCostFunction(p, input_layer_size, hidden_layer_size,
                                         num_labels, X, y, lambda_value)

# Now, costFunction is a function that takes in only one argument (the
# neural network parameters)
[nn_params, cost] = fmincg(costFunction, initial_nn_params, MaxIter)

# Obtain Theta1 and Theta2 back from nn_params
Theta1 = np.reshape(nn_params[0:hidden_layer_size * (input_layer_size + 1)],
                              (hidden_layer_size, (input_layer_size + 1)), order='F')
Theta2 = np.reshape(nn_params[((hidden_layer_size * (input_layer_size + 1))):],
                              (num_labels, (hidden_layer_size + 1)), order='F')
```

```
Training Neural Network...
Iteration  1 | Cost:  [3.30026356]
Iteration  2 | Cost:  [3.26268185]
Iteration  3 | Cost:  [3.21438411]
Iteration  4 | Cost:  [3.16758627]
Iteration  5 | Cost:  [2.56598642]
Iteration  6 | Cost:  [2.19326751]
Iteration  7 | Cost:  [2.01952011]
Iteration  8 | Cost:  [1.93903851]
Iteration  9 | Cost:  [1.71926739]
Iteration  10 | Cost:  [1.59412409]
Iteration  11 | Cost:  [1.49311928]
Iteration  12 | Cost:  [1.38502091]
Iteration  13 | Cost:  [1.30423807]
Iteration  14 | Cost:  [1.18038896]
Iteration  15 | Cost:  [1.12379057]
Iteration  16 | Cost:  [1.04033805]
Iteration  17 | Cost:  [1.00197703]
Iteration  18 | Cost:  [0.95638689]
Iteration  19 | Cost:  [0.87753565]
Iteration  20 | Cost:  [0.83080562]
Iteration  21 | Cost:  [0.8147841]
Iteration  22 | Cost:  [0.77740566]
Iteration  23 | Cost:  [0.75664163]
Iteration  24 | Cost:  [0.74826419]
Iteration  25 | Cost:  [0.72162929]
Iteration  26 | Cost:  [0.70931688]
Iteration  27 | Cost:  [0.70132021]
Iteration  28 | Cost:  [0.68819843]
Iteration  29 | Cost:  [0.65670892]
Iteration  30 | Cost:  [0.64226727]
Iteration  31 | Cost:  [0.63196736]
Iteration  32 | Cost:  [0.62311014]
Iteration  33 | Cost:  [0.60234863]
Iteration  34 | Cost:  [0.58860837]
Iteration  35 | Cost:  [0.58392114]
Iteration  36 | Cost:  [0.57864921]
Iteration  37 | Cost:  [0.57427392]
Iteration  38 | Cost:  [0.55753241]
Iteration  39 | Cost:  [0.54362082]
Iteration  40 | Cost:  [0.52566924]
Iteration  41 | Cost:  [0.51837609]
Iteration  42 | Cost:  [0.51789042]
Iteration  43 | Cost:  [0.50784483]
Iteration  44 | Cost:  [0.5058133]
Iteration  45 | Cost:  [0.50286738]
Iteration  46 | Cost:  [0.50144614]
Iteration  47 | Cost:  [0.50000984]
Iteration  48 | Cost:  [0.49518887]
Iteration  49 | Cost:  [0.48922948]
Iteration  50 | Cost:  [0.48795312]
Iteration  51 | Cost:  [0.48619594]
Iteration  52 | Cost:  [0.48427577]
Iteration  53 | Cost:  [0.48291344]
Iteration  54 | Cost:  [0.4778732]
Iteration  55 | Cost:  [0.47219129]
Iteration  56 | Cost:  [0.46805445]
Iteration  57 | Cost:  [0.4657894]
Iteration  58 | Cost:  [0.45797399]
Iteration  59 | Cost:  [0.44858045]
Iteration  60 | Cost:  [0.43981085]
Iteration  61 | Cost:  [0.43754722]
Iteration  62 | Cost:  [0.43614048]
Iteration  63 | Cost:  [0.43307787]
Iteration  64 | Cost:  [0.4269013]
Iteration  65 | Cost:  [0.42360141]
Iteration  66 | Cost:  [0.42150815]
Iteration  67 | Cost:  [0.41952627]
Iteration  68 | Cost:  [0.41735055]
Iteration  69 | Cost:  [0.41455052]
Iteration  70 | Cost:  [0.41283877]
Iteration  71 | Cost:  [0.41202996]
Iteration  72 | Cost:  [0.41088656]
Iteration  73 | Cost:  [0.41024384]
Iteration  74 | Cost:  [0.4098511]
Iteration  75 | Cost:  [0.40932055]
Iteration  76 | Cost:  [0.40881993]
Iteration  77 | Cost:  [0.40779133]
Iteration  78 | Cost:  [0.40527966]
Iteration  79 | Cost:  [0.3980038]
Iteration  80 | Cost:  [0.37946909]
Iteration  81 | Cost:  [0.3734955]
Iteration  82 | Cost:  [0.37184196]
Iteration  83 | Cost:  [0.37093083]
Iteration  84 | Cost:  [0.37007113]
Iteration  85 | Cost:  [0.36889505]
Iteration  86 | Cost:  [0.36766463]
Iteration  87 | Cost:  [0.36711919]
Iteration  88 | Cost:  [0.36653245]
Iteration  89 | Cost:  [0.36507657]
Iteration  90 | Cost:  [0.36321359]
Iteration  91 | Cost:  [0.3625492]
Iteration  92 | Cost:  [0.36220006]
Iteration  93 | Cost:  [0.36171639]
Iteration  94 | Cost:  [0.36137476]
Iteration  95 | Cost:  [0.36096773]
Iteration  96 | Cost:  [0.36013939]
Iteration  97 | Cost:  [0.35910257]
Iteration  98 | Cost:  [0.35802951]
Iteration  99 | Cost:  [0.35769067]
Iteration  100 | Cost:  [0.35734097]
Iteration  101 | Cost:  [0.35638453]
Iteration  102 | Cost:  [0.35563467]
Iteration  103 | Cost:  [0.35460499]
Iteration  104 | Cost:  [0.3535512]
Iteration  105 | Cost:  [0.35280084]
Iteration  106 | Cost:  [0.35230515]
Iteration  107 | Cost:  [0.3512594]
Iteration  108 | Cost:  [0.35049061]
Iteration  109 | Cost:  [0.3501542]
Iteration  110 | Cost:  [0.34979354]
Iteration  111 | Cost:  [0.34966652]
Iteration  112 | Cost:  [0.34950062]
Iteration  113 | Cost:  [0.34934875]
Iteration  114 | Cost:  [0.34917611]
Iteration  115 | Cost:  [0.34900038]
Iteration  116 | Cost:  [0.3485549]
Iteration  117 | Cost:  [0.34801865]
Iteration  118 | Cost:  [0.3470069]
Iteration  119 | Cost:  [0.34641211]
Iteration  120 | Cost:  [0.34557354]
Iteration  121 | Cost:  [0.34505288]
Iteration  122 | Cost:  [0.34487433]
Iteration  123 | Cost:  [0.34463302]
Iteration  124 | Cost:  [0.34448792]
Iteration  125 | Cost:  [0.34440091]
Iteration  126 | Cost:  [0.34420122]
Iteration  127 | Cost:  [0.34399859]
Iteration  128 | Cost:  [0.34387723]
Iteration  129 | Cost:  [0.34375681]
Iteration  130 | Cost:  [0.34347522]
Iteration  131 | Cost:  [0.34330569]
Iteration  132 | Cost:  [0.34327513]
Iteration  133 | Cost:  [0.34315738]
Iteration  134 | Cost:  [0.34295746]
Iteration  135 | Cost:  [0.34272894]
Iteration  136 | Cost:  [0.34254057]
Iteration  137 | Cost:  [0.34241726]
Iteration  138 | Cost:  [0.34218716]
Iteration  139 | Cost:  [0.3415536]
Iteration  140 | Cost:  [0.34123958]
Iteration  141 | Cost:  [0.34117752]
Iteration  142 | Cost:  [0.34056684]
Iteration  143 | Cost:  [0.34028717]
Iteration  144 | Cost:  [0.34021801]
Iteration  145 | Cost:  [0.33995469]
Iteration  146 | Cost:  [0.33974312]
Iteration  147 | Cost:  [0.33961262]
Iteration  148 | Cost:  [0.33946263]
Iteration  149 | Cost:  [0.33931936]
Iteration  150 | Cost:  [0.33915545]
```

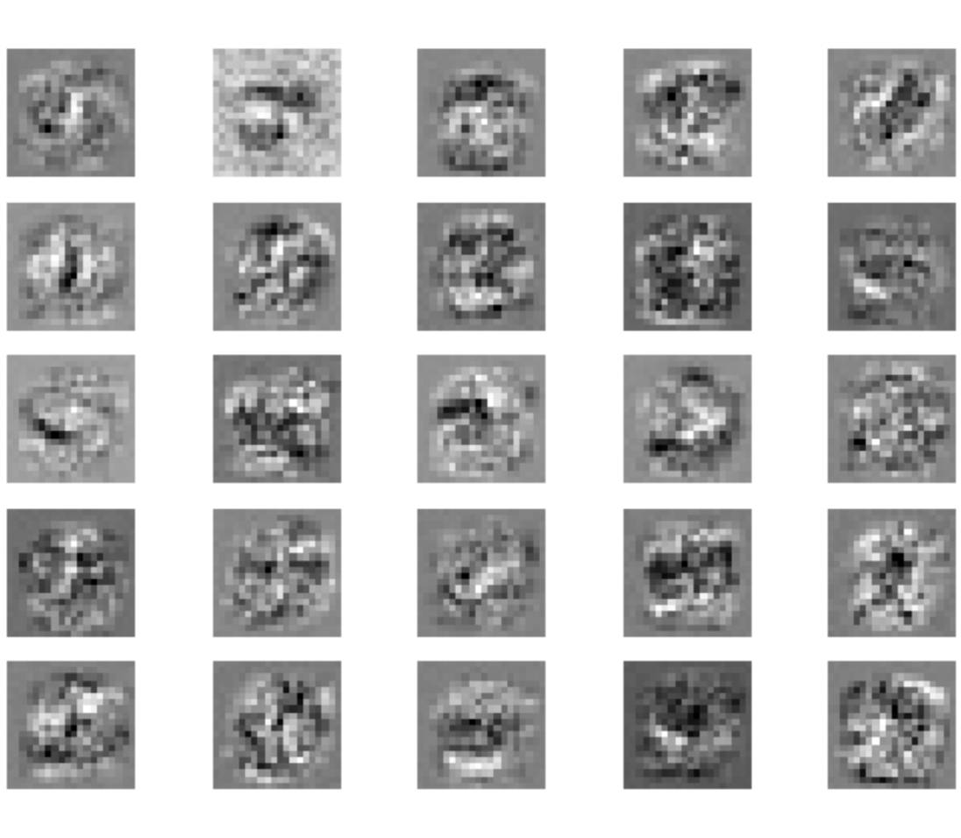
================= Part 9: Visualize Weights =================

You can now "visualize" what the neural network is learning by displaying the hidden units to see what features they are capturing in the data.

In [16]:

```
print('\nVisualizing Neural Network... \n')

displayData(Theta1[:, 1:])
```

```
Visualizing Neural Network...
```



============= Part 10: Predicting with learned weights =======

After training the neural network, we would like to use it to predict the labels. The already implemented "predict" function is used by neural network to predict the labels of the training set. This letsyou compute the training set accuracy.

In [17]:

```
pred = predict(Theta1, Theta2, X)
pred = np.expand_dims(pred,axis=1)
print('Training Set Accuracy: ', (pred == y-1).mean()*100)
```

```
Training Set Accuracy:  99.1
```

In [ ]: