**Exercise: Recognizing handwritten digits with feed-forward networks**

In this exercise we will use feed-forward neural networks to recognize images of handwritten digits (from 0 to 9). These networks are particularly useful for recognizing digits in checks, deposit slips, and envelopes (ZIP codes). Thus, providing a quick method to process deposits and sort mail for delivery; therefore, automating tedious and error-prone tasks when performed by humans.

In this exercise, we will not learn how to train (i.e. learn the weights/parameters from the data) these networks; this will be explained in the machine learning section of this course. What we will do is to use the learned weights to make predictions on data that the network was not trained on (i.e. it has never seen before). This data is called a test set. This task will give us the opportunity to review a couple of key linear algebra concepts (mostly matrix multiplications) and become familiar with the lexicon and ideas behind machine learning.

The given test set is in file *MNISTOCW.mat*. This is a subset of a larger data set of handwritten digits which is often used to test and benchmark classification and learning algorithms (see <http://yann.lecun.com/exdb/mnist/>).

The matrix X in that file contains 3000 examples of handwritten digits. In Figure 1 we show 100 randomly selected examples from this test set. Each digit is a 28 by 28 pixels gray scale image that has been concatenated into a 1 by 784 pixels vector. Therefore, each row of the matrix X (which has dimensions 3000 by 784) contains a digit example.

The column vector y contains the labels for the test set. The digits 1 to 9 were coded with corresponding labels 1 to 9. To make things more MATLAB friendly (remember that MATLAB starts indexing from 1 and not from 0) we mapped the digit 0 to a label of 10.

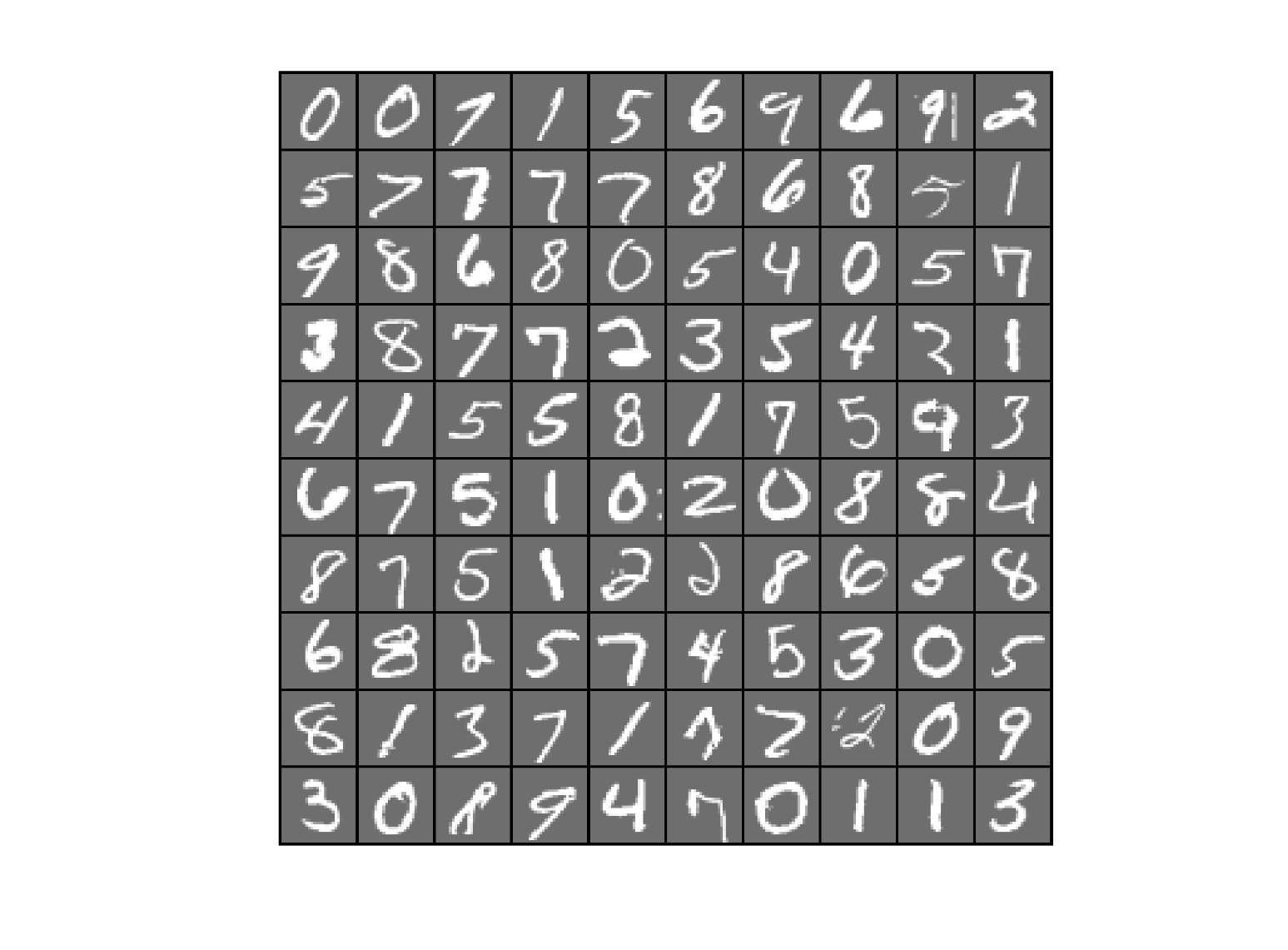


Figure 1. Example digits from the MNIST data set.

**Network Architecture:**

The network comprises three layers: input, hidden, and output (shown in Figure 2).

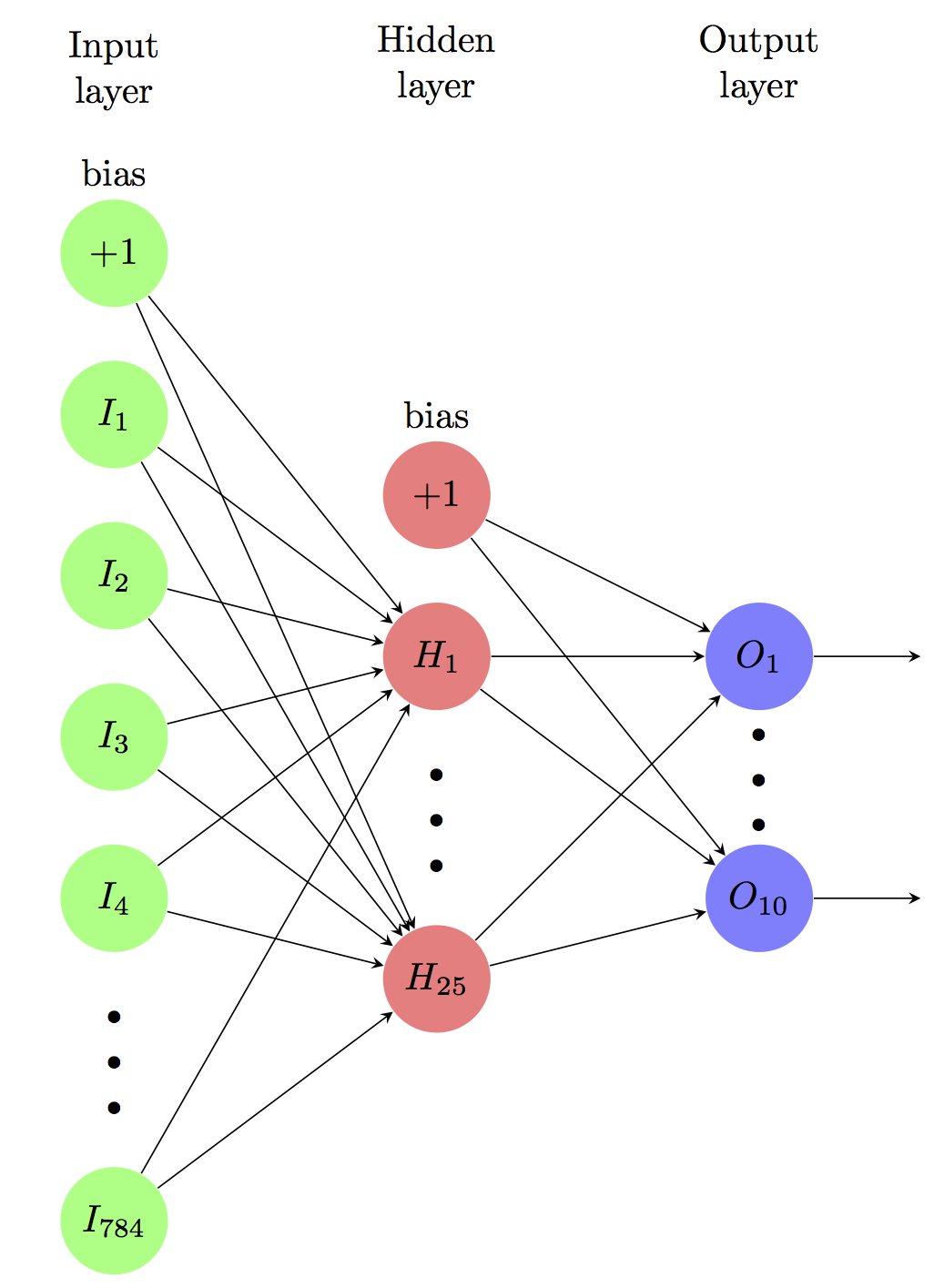


Figure 2. Schematic diagram of a three-layer feed-forward neural network.

The number of units in the input layer depends on the dimensionality of the problem. In this case we are dealing with images that live in a 784 dimensional space. Therefore, we have 784 units in the input layer.

For the hidden layer we have chosen 25 units in this case, as this provides reasonably good classification results. Usually, we have flexibility in choosing the number of units in the hidden layer when building classification networks.

Finally, we want each output unit to code exclusively for one of the digits (0-9) in the data set. Hence, we set the number of output units to 10. The idea is that after training, the output unit 1 will get activated only when a digit 1 is presented to the network. Following the same logic and the coding scheme discussed above, the output unit 10 will be activated only when a digit 0 is presented to the network.

This makes a network with architecture 784-25-10; please note that bias units are omitted in this analysis.

The input layer in this particular example consists of 784 units. That is, one unit for each pixel of the image. Additionally, we included a bias unit that always has input value of +1. This is equivalent to adding an intercept to a linear regression problem, for example.

Each input unit connects to each of the units in the hidden layer with a fixed weight. The weights are specified in matrix  (matrix Theta1 in file *MNISTOCW.mat*). Therefore, each hidden unit is activated by a linear combination of the input image pixels plus a bias term. Each hidden unit then passes this input through a non-linear function to produce an output.

Then the collection of outputs from the hidden layer (plus a new bias term) is fed forward to the output layer. Again, each output unit receives connections from all units in the hidden layer. The weights for these connections are specified in matrix  (matrix Theta2 in the file *MNISTOCW.mat*).

**Non-linear function:**

The logistic sigmoid function  is the classical choice of non-linearity for feed-forward networks:



We plot this function in Figure 3:



Figure 3. Plot of the logistic sigmoid function.

 has a couple nice properties that make it the natural choice for neural networks:



**Exercises:**

1. Using pencil and paper, prove the above two properties of the logistic sigmoid function.
2. Write a MATLAB function, sigmoid.m, which takes as input an arbitrary 2D matrix. This function should compute the sigmoid function of each element of the input matrix and return it in a matrix of the same dimensions as the input.
3. Write MATLAB code that implements a feed-forward of all the images in the data set with the specified weights through the hidden layer. First, recall that the activation of the first layer is given by:



and that the bias term can be added in MATLAB with the following code:

numExamples = size(X,1);

a1 = [ones(numExamples,1) X];

Secondly, the input to the hidden layer is given by this matrix multiplication:



Next, the activation of the hidden layer is:



The sigmoid function you wrote will be extremely helpful for this.

What are the dimensions of the matrix ? What do the rows and columns of this matrix represent?

1. Add the bias term to (this is similar to what you did in layer 1) and write code to compute the input and activation of the output layer. You will use again your sigmoid function to achieve this goal: The equations are:





What are the dimensions of the matrix ? What do the rows and columns of this matrix represent?

1. To classify the images we will use a winner-take-all approach. That is, we will assign a predicted label to the unit that has the highest activity. For example, if an input digit image produces as output vector:



Then, we will predict that it is digit 7, as the seventh unit has the largest activity.

Write MATLAB code to assign predicted labels to the 3000 examples in the data set.

1. What percentage of images in the test set does the network correctly classify?
2. To debug classification algorithms it is often important to understand whether the misclassifications are biased. In this case, this means whether any given digit is more likely to be misclassified than any of the others. An easy way to capture this is by computing the confusion matrix. In this matrix the i,j entry represents the number of times label i is classified as j. Therefore, the diagonal elements represent correct classifications.

Use the MATLAB command confusionmat to compute the confusion matrix and then visualize it with imagesc. To aid the visualization set the diagonal entries to 0.

Are there any notorious bias/biases in this network?