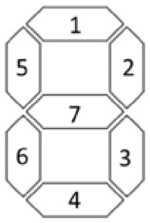
Learning Guide Unit 6

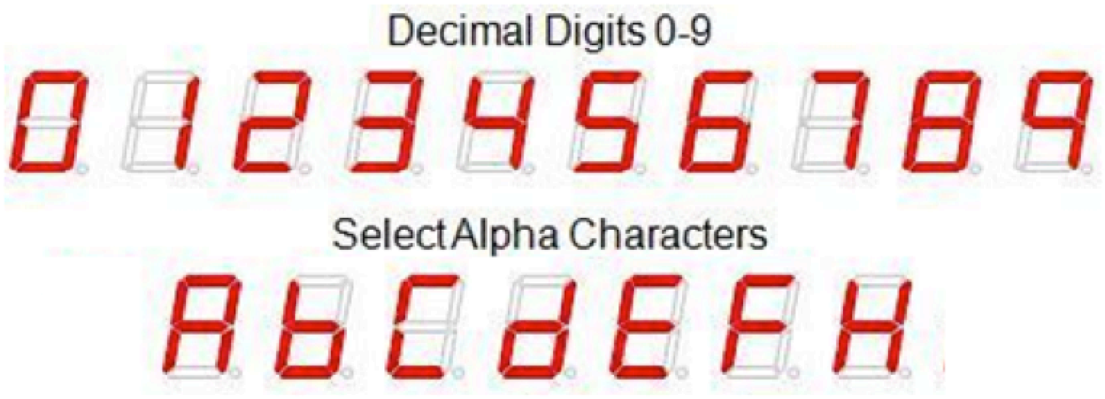
## Programming Assignment

For the Unit 6 assignment, you will be required to build and train a neural network to recognize letters of the alphabet and numbers based upon their design using a seven segment display where each segment of the display is one input into the neural network.



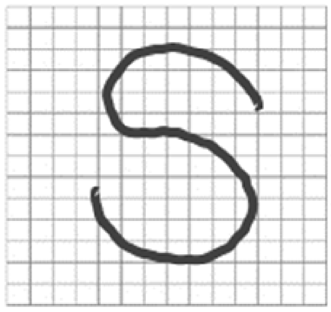
Using the numbering for the segments in the previous figure as the inputs into your neural network. You are welcome to use any network design that will accurately solve the problem. In this case, your network must be able to identify the letter or number based upon the pattern of segments. For example, if the segments 1, 2, 3, 4, and 7 are lighted then the pattern would represent the number 3. The number three must be represented as a binary number that is the output of the neural network.

Not all of the letters of the alphabet can be accurately recognized. The following chart represents the numbers and letters that your network must be able to recognize.



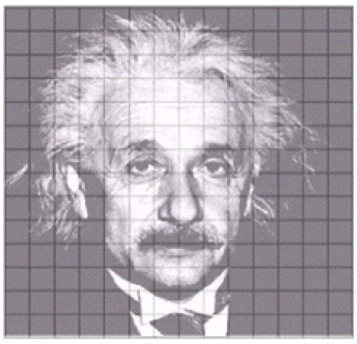
The output of your network will be the ASCII code of the letter or number defined in the following list and represented as binary (for details about ASCII, see https://www.asciitable.com/). Technically, lowercase letters have higher ASCII codes, but your network should only use capital letter codes (e.g., recognize b as B).

We are using a 7 segment display to recognize letters and numbers, however, it is important to know that we could use the same approach to recognize virtually anything. If we had more input values we could recognize any handwritten letter or number. Consider the following examples example. Assume that we had a matrix that was 16 x 16 (total of 256 point) and each of the points were an input into our neural network. Further, assume that we could overlay a handwritten letter or number onto this matrix. Every pixel that was colored by the letter would have a value of 1 and any pixel that was not colored would have a value of 0.



Using this as input we could easily train a neural network to 'recognize' various handwritten letters. As the amount of training increa the accuracy of the network to detect similar letters that might not have the exact same shape (but a similar one) would increase.

Let's consider another example. Assume that we had a neural network and instead of the inputs being 1 or 0, we had the ability to determine a value between 0 and 1 as input where 0 was the color white and 1 the color black and shades of grey as values in between 0 and 1.



Using this approach we could train the neural network to recognize faces, people, and objects … such as Albert Einstein here. In practice, facial recognition software will typically limit the number of 'features' that are used as input to the neural network to areas that have the greatest predictive power such as the position of the eyes, nose, mouth, and perhaps chin, but the principle remains the same as the network that we will build to recognize the letters in the 7 segment display.

To complete this assignment, you will need to download and install the Basic Prop neural network simulator. Basic Prop is distributed as an executable Java Jar file. You can either execute the file on your local computer or you can access the simulator in the Virtual Computing Lab. To run the simulator on your local computer you will need to have the Java JRE (java runtime environment) version 1.5 or greater installed on your computer.

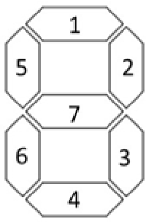
The simulator can be downloaded from the basic prop website at the following URL: http://basicprop.wordpress.com/2011/12/21/introducing-basic-prop-a-simple-neural-networ/

Using the simulator you will need to design a network, load the pattern file, and then train the network. You should experiment with your network design to make sure that your network and solve all of the test cases. If your network is not solving all of the cases, then you should alter the design of the network and train the network again.

**Part 1.** Write the pattern file

The pattern file is what the program uses to train the network. To create this file, you need to encode the input and output, and then put the data into a format that the program can analyze.

a. Encode the input (the seven segment display) in a binary format that the program can recognize. This can be done many ways, but I suggest making a seven-element vector with each index representing one location in the display.



If that location is lit up for a number, then the value should be 1. Otherwise, it should be zero. For instance, the code for 1 would be 0 1 1 0 0 0 0 because locations 2 and three are lit up for the display of 1. The code for 2 is 1 1 0 1 0 1 1. And so on.

b. Encode the output in a binary format. Since the program requires binary input and output, the values need to be converted into binary. Again, any mapping would work here, but I suggest using the ASCII values for each character. In general, you can look these up in an ASCII table, but for this assignment, the relevant codes are listed below.

| Character | ASCII |
|---|---|
| 0 | 48 |
| 1 | 49 |
| 2 | 50 |
| 3 | 51 |
| 4 | 52 |
| 5 | 53 |
| 6 | 54 |
| 7 | 55 |

| Character | ASCII |
|---|---|
| 8 | 56 |
| 9 | 57 |
| A | 65 |
| B | 66 |
| C | 67 |
| D | 68 |
| E | 69 |
| F | 70 |
| H | 72 |

You can then convert the ASCII values to binary format, making sure all of them are the same length (7 in this case). For instance, 3 would be represented by 0110011 because 1*1+1*2+0*4+0*8+1*16+1*32+0*64=51 (the ASCII value for 3).
c. Create a text file to use for training the network. This file should be in the following format:

```
Number of patterns = 4
Number of inputs = 2
Number of outputs = 1
[patterns]
0 0    0
1 0    0
0 1    0
1 1    1
```

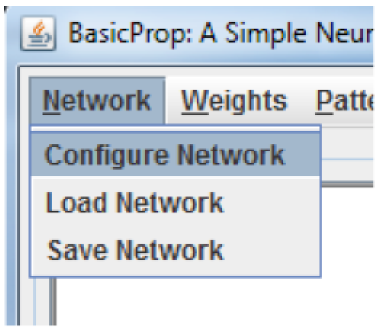
The number of patterns is the number of different examples you are providing to the network. In your assignment, that number should be 17 (10 digits and 7 letters). The number of inputs is the length of the inputs from part a (7). The number of outputs should be the length of the outputs from part b (7). Each of the patterns should then be an input followed by the corresponding output, making it a series of 14 ones and zeros. For example, the first pattern (0) should have an input pattern of 1 1 1 1 1 1 0 and an output pattern of 0 1 1 0 0 0 0 (the binary representation for 48), so the first pattern would be:

```
 1 1 1 1 1 1 0     0 1 1 0 0 0 0
```

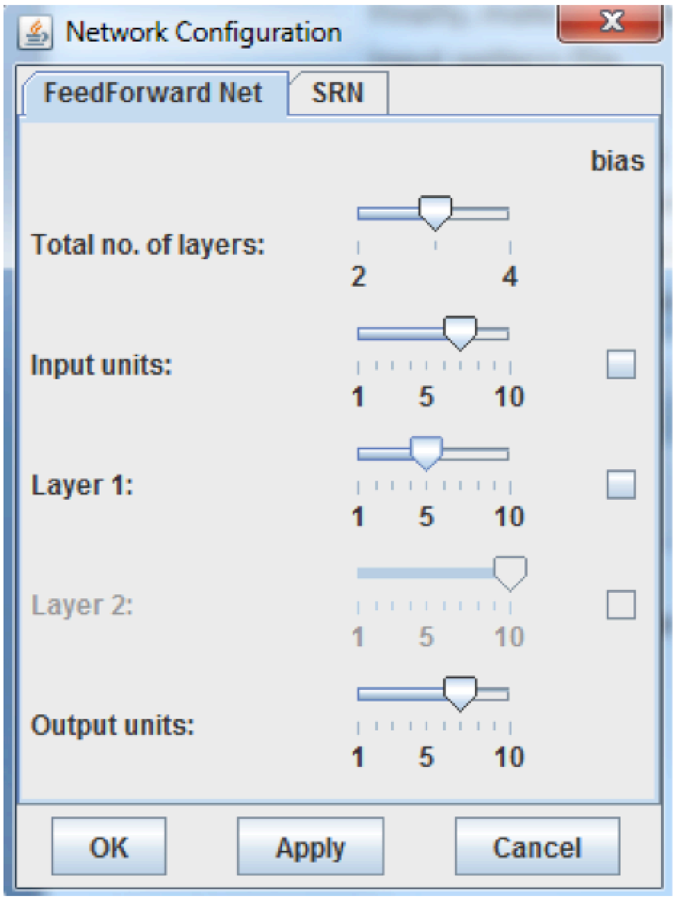
Finally, make sure the extension for the files is .pat so the program recognizes it as an input pattern file. Include this input file in your final submission.

**Part 2.** Create the network

a. Configure the network by going to the Network menu and choosing Configure Network from the list of options



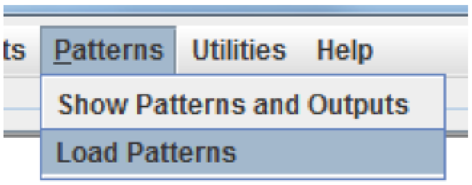
Then decide the type and structure of the network to use. You have the option of two types of networks, different numbers of layers, different numbers of units in those layers, etc. You can play around with these to see what works best, or read about them and try to make an informed decision. Some configurations will work better than others. You do need to make sure that you make the number of inputs and outputs both equal to 7, or your file won't load correctly.



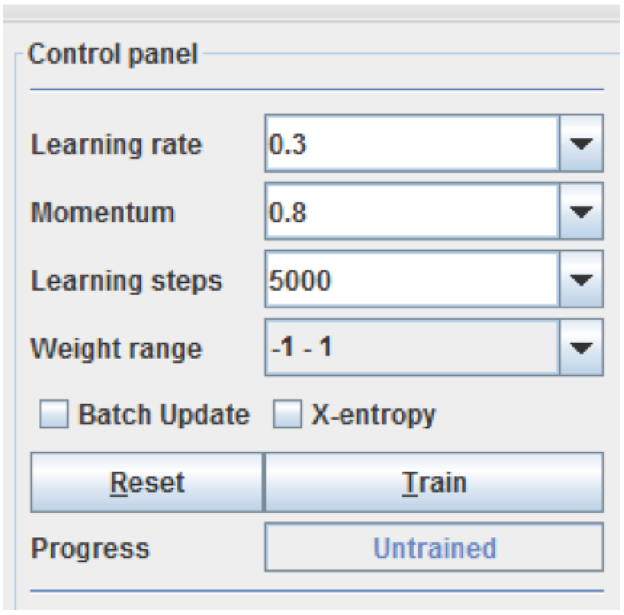
Total no. of layers: 2 4
Input units: 1 5 10
Layer 1: 1 5 10
Layer 2: 1 5 10
Output units: 1 5 10

Report the structure that you used for your final model (number of layers, layer size, etc.), including a screenshot of the model.

b. Train the network by loading in your pattern file. This can be done by going to the Patterns dropdown menu and choosing Load Patterns. Then choose the file that you created from part 1.

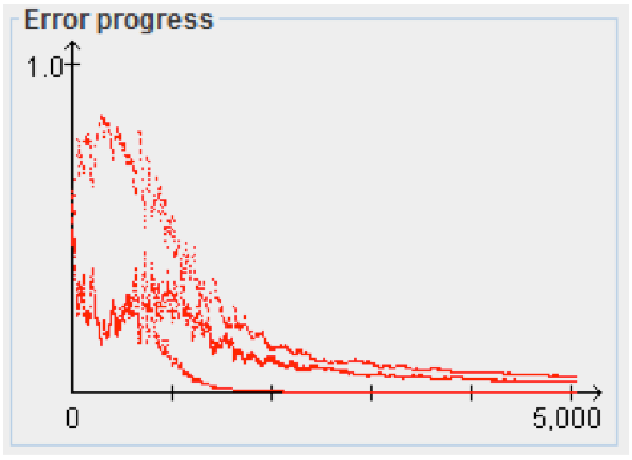


Then use the control panel on the right side to train your network. You might want to change the parameter values to get better training results. Again, you can read about what these parameters do, or you can try different combinations to see what works best.

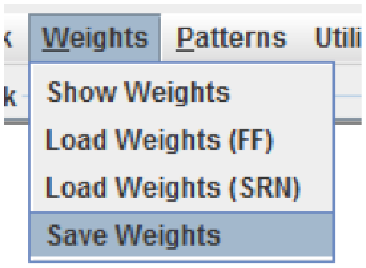


Learning rate 0.3
Momentum 0.8
Learning steps 5000
Weight range -1 - 1
Batch Update
X-entropy
Progress Untrained

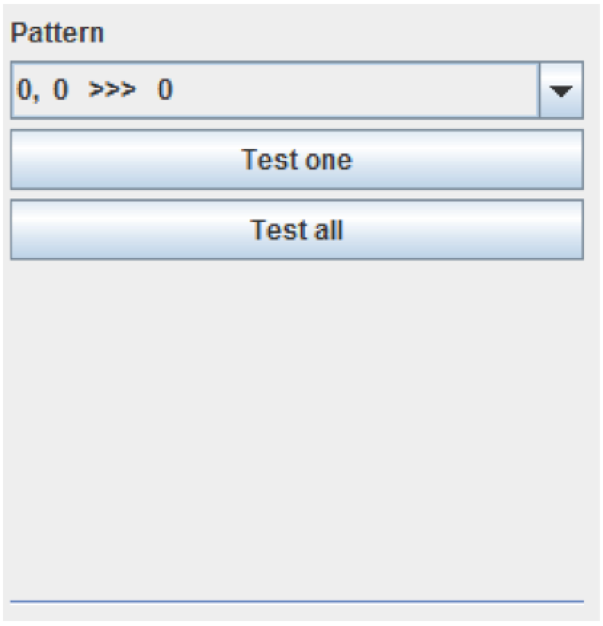
Once the parameters are set, train the network by clicking the Train button. The error progress graph at the bottom will give an indication on the progress of training. If the errors aren't converging to 0, that means you likely need to adjust your model.



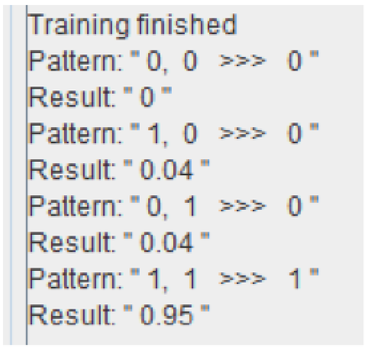
Report the parameter values that you used for training your model as well as a graph of the errors over the training steps. Include the weight file in your final submission, either as a separate file, or copied into your write-up. The weights can be saved by going to the Weights menu and choosing Save Weights.



c. Test the model by choosing individual patterns and Clicking the Test one button.



The results will be printing in the console at the bottom. For example, here is the output from the and network:



```
Training finished
Pattern: " 0,  0   >>>   0 "
Result: " 0 "
Pattern: " 1,  0   >>>   0 "
Result: " 0.04 "
Pattern: " 0,  1   >>>   0 "
Result: " 0.04 "
Pattern: " 1,  1   >>>   1 "
Result: " 0.95 "
```

Report the results for each of the inputs. Then, choose Test all to get the average error and report the resulting value. You will report the results in the Unit 7 assignment. For this unit, submit you input pattern file and the screenshots required. You will be graded on the following:

1. Did you include a screenshot of a neural network designed to solve the recognition problem?
2. Did you include the average per pattern error either reported in the assignment or visible from a screenshot of the neural network simulator?
3. Was the average pattern error reported either in the assignment text or visible from a screenshot of the neural network simulator. Was the error percentage acceptable (less than 5% or .05)?
4. Were the weight file(s) for the optimal solution (required) and other trial solutions included (optional)?
5. Was the number of training steps reported either within the text of the assignment or visible from a screenshot of the neural network simulator?