When developing your simulator, keep in mind that the data structure should be as general as possible, for example, it should be able to handle large dimensional input vectors. It means that **do not** hard-code the parameters for the structure of the neural network (e.g., the number of inputs, number of hidden layers, number of outputs, number of maximum iterations, and the r value.) in the simulator, and have these parameters adjustable when the simulator runs by asking and accepting the user inputs.

Experimenting with your simulator by training and testing it several times with different settings of the parameters. Use the data files 'NNtrain' and 'NNtest' provided in this class to train and test the simulator.

The 'NNtrain' and 'NNtest' files are datasets of 36 alphanumeric characters (uppercase letters and 10 digits). Each alphanumeric character is extracted from a 5-by-7 grid of its binary image and represented as a 35 binary character string in the file. The files have 36 columns where the first column contains the labels of the characters coded on the rows.

It is obvious that the input layer of the network must have at least 35 nodes.
The output layer could be organized in the following two ways:
(1) having 36 nodes each represent one character. In this case, you need to convert the labels (on the first column) to corresponding binary code to use as the desired output. For example, for a label of character 'A' it should be converted to "1000… …", and a 'B' to "01… …"
(2) having 8 nodes to represent the ASCII code of the character. For example, if your input is a character "A", then your desired output should be "01000001", i.e., 41_{16} - the ASCII code for character "A".
Note that in the test phase, you need to convert the binary outputs back to the characters they represent to generate a table described below for a representation of the classification results.

Calculate the error term $E(t)$ and $\Delta E = E(t) - E(t\text{-}1)$ in each iteration of your simulator run. Where $E(t)$ is the average error for each iteration:

$$E(t) = \frac{1}{N}\frac{1}{M}\sum_{p=1}^{N}\sum_{k=1}^{M}(y_{pk} - O_{pk})^2 \text{ for the } t^{th} \text{ loop}$$

Where, N is the total number of training samples (input vectors). M is the total number of neurons on the output layer.
y_{pk} is the desired output value on the k^{th} output neuron with respect to the p^{th} training input vector
O_{pk} is the actual output value on the k^{th} output neuron with respect to the p^{th} training input vector.

Terminate the learning when the $E(t)$ and $\Delta E(t)$ are acceptably small (set up the $E(t)$ and $\Delta E(t)$ values as input parameters for each run of the simulator). It is suggested to force the termination of the training process after certain number of iterations (i.e., the number of maximum iterations) to avoid non-convergence cases.

Your test outcome should be represented in the format of a table (it is usually called Confusion Matrix - a special kind of contingency table) shown below, where each column represents actual identity of character to be recognized, each row represents the identity of character classified by the simulator. For example, in the example below, the column 'A' shows that 7 test cases of the character 'A' are correctly classified as 'A', but 1 test case of the character 'A' is classified as a 'B.'

Table # (caption)

		Actual character							
		'A'	'B'	. . . . .	'Z'	'1'	'2'	… …	'0'
Classified character	'A'	7	0		0	0	0		0
	'B'	1	6		0	0	1		0
	: :								
	'Z'	0	0		7	0	1		
	'1'	0			0	8	0		0
	'2'	0	1		0	0	5		0
	: :								
	'0'	0	1		0	0	0		7

Try to run your simulator with different number of hidden layers, number of output nodes, number of maximum iterations, as well as different r values, to see how the results (in the forms of the tables describe above) are different with respect to the different parameters. Record your runs in a table as the example shown below:

Run #	# of layers	# of neurons on each layer	# of iterations of actual run	*'r'* value	$E(t)$	$\Delta E(t)$	Caption of Corresponding Confusion Matrix
1	2	(35, 20)	540	0.5	0.95	0.0001	Table #1
2	2	(35, 10)	1100	0.75	0.98	0.0001	Table #2
3	3	(35, 10, 10)	506	1.0	0.89	0.0001	Table #3
4	4	(35, 10, 5, 5)	212	0.8	0.78	0.0001	Table #4