

Neural Networks Laboratory
EE 329 A

Introduction:

Artificial Neural Networks (ANN) are widely used to approximate complex systems that are difficult to model using conventional modeling techniques such as mathematical modeling. The most common applications are function approximation (feature extraction), and pattern recognition and classification. There is no exact available formula to decide what architecture of ANN and which training algorithm will solve a given problem. The best solution is obtained by trial and error. One can get an idea by looking at a problem and decide to start with simple networks; going on to complex ones till the solution is within the acceptable limits of error. This laboratory has been designed to give one an opportunity to try different network architectures and training algorithms, to solve a simple classification problem, and comment on their performance.

There are different neural network architectures. The basic architectures include multi-layered feed-forward networks (Figure 1) that are trained using back-propagation training algorithms.

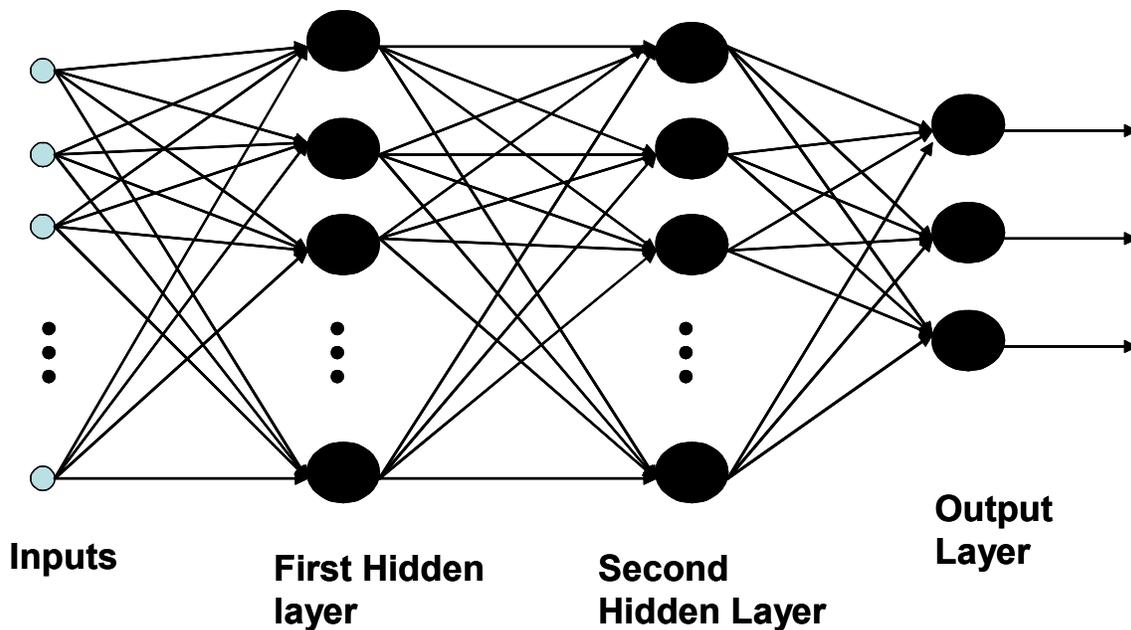


Figure 1: Multi-layered feed-forward neural network. A variation in the architecture of such a network can be due to a variation of the number of layers, the number of neurons in each layer, the transfer function of neurons in each layer.

Problem Description:

The iris flower (Figure 2) data set is widely used, as an example, to show the application of neural networks as a data classifier. Three types of similar flowers from the IRIS family have been chosen. They are:

1. I. Setosa
2. I. Verginica
3. I. Versicolor

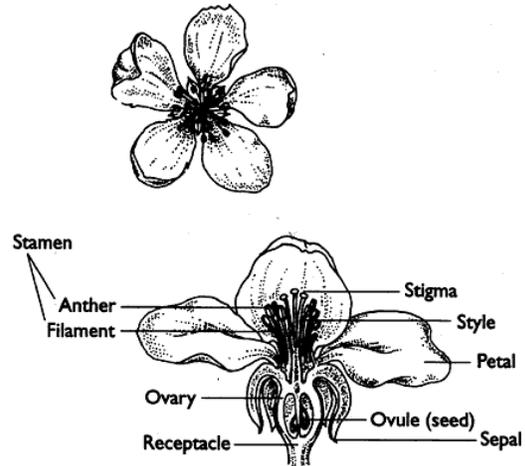


Figure 2: Iris Verginica (Left). Parts of a flower. Note the Sepal and the petal. (Source: http://www.bbg.org/gar2/topics/botany/parts_flower_parts.html)

Four distinguishing features were chosen in order to classify a particular flower to its category. The four features (Figure 2) are:

1. Petal width
2. Petal length
3. Sepal width
4. Sepal length

The goal is to use an ANN to classify a specimen into its category using the above mentioned four features. You will be using Multilayered Perceptrons (MLP) feedforward networks using different backpropagation training algorithms.

Data Setup (Pre-processing):

The features obtained from many specimens are actually considered as raw data. Some amount of pre-processing is always carried out on the input and output data in order to make it suitable for the network. These procedures help obtain faster and efficient training. There are four features that are used as inputs to the neural networks. Each feature has a range of values. If the neurons have nonlinear transfer functions (whose output range is from -1 to 1 or 0 to 1), the data is typically normalized for efficiency. Each feature is normalized using one of the two subroutines available in MATLAB.

Since there are three classes of flowers, a three-state numerical code must be assigned. The network must be trained to reproduce the code at the outputs. The output can be left in decimal form where the decimal numbers 1, 2, and 3 represent the classes or the classes can be coded in binary form. For this problem the outputs are coded in binary form as

1. I. Setosa : Code 00
2. I. Verginica : Code 01
3. I. Versicolor : Code 10

Architecture Details (factors to be taken into account):

As such, one can employ as many layers of neurons to develop the architecture. But, 3-4 layers are generally sufficient to approximate most of the functions. While building the architecture, you can adjust the number of layers and the number of neurons in each layer. Since the output of the network is in 2 bit binary form, the output layer will have to have two neurons. If you had decided to code the output in decimal form (1, 2, and 3), the output layer would only need one neuron.

Software:

One can develop ANN architectures and write training algorithms in any known higher level language. MATLAB has a proven tool box that helps one apply, already developed training algorithms, to solve the problem. MATLAB also has a GUI based system. This windows-based platform provides a step-by-step visual procedure to build, train and simulate the ANN.

Startup:

- Start MATLAB. You have been provided with a file 'data.mat'. This file has data already formatted and ready for usage. Load the file by typing the command *load ('data.mat');* in the command window. This command uploads the file and includes all the variables setup in the current workspace.
- Typing in the command *whos*, you will see all the variables in the data file. Of these, the most important ones are:
 - *train_inputs* (4×120): 4 rows by 120 columns. The number 4 represents the normalized features which will be presented, one-by-one, in parallel fashion. Each pattern of 4 features is called a vector and there are 120 such vectors for training
 - *train_outputs* (2×120): There are two outputs for each feature vector (coded in binary). There are 120 such vectors for training and will be the targets for the input vectors.
 - *test_inputs* (4×30): Once the network has been trained, it has to be tested on data that it has not seen before. 30 vectors will be used for testing the network.
 - *test_outputs* (2×30): These are the expected classes for the test inputs. The simulated outputs of the network will be compared to these actual classes in order to determine the performance of the trained network.
- Typing the command *NNTool* brings up the GUI based neural network developer called the "Network/Data Manager".
- First you import some data. Click on the 'Import' button. A new window opens up. Here you can specify the source of your data. Since the workspace is already loaded, you can import data from there. Select the variable *train_inputs* and since this variable pertains to inputs, in the destination section, select the 'inputs' option and click on the import button. You will see that in the 'inputs' sub window, the variable *train_inputs* has appeared. Similarly import *test_inputs* as inputs. Also

import *train_outputs* and *test_outputs* as targets. Also, import *pn* as an input, which is actually the entire normalized input data and will be used to specify the range of the data.

- Next you build the network.
 - Click on the ‘New Network’ button. You can give a name to your network. You can select the different types of networks you can build and train. The Feed-forward backpropagation neural network is the default choice and is the type used in this laboratory.
 - Next you provide the range of values of the input data. This operation can be done by selecting the source in the right hand sub window and selecting *pn*. The network manager will automatically calculate the range of the input values.
 - Next, the training algorithm needs to be specified. During lecture, the gradient descent training algorithm was explained. That training algorithm is the ‘TRAINGD’. Select this one first.
 - Next you select the learning function. This operation sets up the weight update rule. Select LEARNGD, which was explained in class.
 - Choose MSE as the performance function, which stands for mean squared error.
 - Next choose the number of layers that you want in your network. Type in 2 (to begin with) and press enter. This will tell the network manager that you would like 2 layers for the network.
 - Once you have chosen the number of layers, you have to specify the properties of each layer. You can select an individual layer and specify the number of neurons in that layer and the transfer function of neurons for that layer. You will definitely need two neurons for the output layer (remember the 2 bit binary coding) and 4 neurons in the first input layer (You can vary this number later). Choose LOGSIG as the transfer function as you would like the outputs to either move towards a zero or one.
 - Finally, click on the create button and you will see your network name appear in the ‘Networks’ sub window of the network manager.
 - When you select the network you have just created, you will see a number of buttons become active in ‘Networks only’ area of the manager.
 - Review the different sections. First, look at the ‘view’ sections, where you see a visual description of your ANN architecture. Note, the number of layers and number of neurons in each layer and their respective transfer functions.
 - Next look at the ‘initialize’ section. You will see the ranges of the input values that have been already set earlier. Here you can initialize the weights by clicking on the ‘initialize weights’ button. The weights will be initialized randomly between -1 and 1.
 - You can look at the weight matrices and the biases in the ‘Weights’ section.
 - After the weights have been initialized, the network is ready for training. Select the ‘Train’ section, which has three subsections

- Training info: Selecting this section, you will have to specify the training data. Under the “Training Data” select *train_inputs* as inputs and *train_outputs* as the targets.
- Training Parameters: Here you should specify some performance parameters pertaining to the type of training algorithm that you are going to use. The most important ones for all the algorithms are the
 - Epochs: This decides for how many cycles you would like to train the network. For example, if the number of epochs is 100 then the entire training data will be presented 100 times. Set this value to 500.
 - Min_grad: This decides the acceptable error that you would prefer. If this MSE (mean squared error) is reached the network has converged and training will stop. Generally this value is 1e-5.
 - Show: This tells the software to show you the performance error after a fixed number of epochs. For e.g. if the number is 25, then the MSE will be calculated after every 25 epochs and shown to you.
- Optional Info.: Here you can specify any validation data set (used for early stopping and is generally used when you have a large data set) or a testing data set that you will add for simulating the network. Leave this section as it is.
- After initializing all the parameters the network is read to be trained. Click on the ‘Train Network’ button on the right bottom corner of the window and you will see a new window show up and a blue line go across from left to right. The network is now training and the MSE is being shown every 25 epochs. If the MSE reaches the set value the network has converged and training stops. If the network does not converge and the number of epochs has reached the set value, then the network has not converged. You will have to retrain the network, by changing some of the parameters or the training algorithm or the network architecture (that is the number of layers and the number of neurons in each layer).
- Once you have trained the network, the network will be simulated with a set of inputs that the network has not seen. Go to the ‘Simulate’ section of the network manager. In the ‘Simulation Data’ section, select *test_inputs* as your inputs and then click on ‘Simulate Network’ button on the right bottom corner of the current window. You can also provide the *test_outputs* so that the software will calculate the errors between the simulated output of the network and the expected outputs. But, the feature is not needed since our classes are binary based. Even though we have trained the network on these binary classes, the simulated outputs will not be exactly 0s and 1s. Some post-processing is needed to figure out whether the output bits are 0s or 1s. To do this a routine (already built for you) is run from the command window of MATLAB. But first you will export the simulated outputs to the workspace. Your simulated outputs

will be stored in the variable *network1_outputs*. Going back to the Network/Data Manager window, click on the 'Export' button. Select this variable and click on 'Export'. You also have a facility to store this information on a disk. Once you have done this, go to the command window and type 'whos' and you will see the variable *network1_outputs* present there. You will now run the subroutine "post_proc.m". In this routine, you will replace '*network1_outputs*' with the variable that will be exported from your network/data manager to your workspace. Your final outputs will be stored under the variable '*sim_output*'. Be sure to give a different name to this variable when ever you try to post process new simulated outputs.

- Compare the post-processed outputs and the *test_outputs* and see how many test vectors were classified incorrectly. Please fill in the following blanks. This process will summarize the entire operation and results that you have just obtained.
 - Type of architecture: _____
 - Number of Layers (hidden + output): _____
 - _____
Details of number of neurons and transfer function used.

Layer Number	Number of Neurons	Transfer Function
1		
2		
3		
4		
5		

- Training algorithm used **: _____
 - Number of epochs required for training: _____
 - Number of test_vectors classified correctly: _____
- You should try architectures with varying number of hidden layers (go up to a maximum of 3 layers), number of neurons in hidden layer. Observe the effect on the result due to a variation in training times (number of epochs) and training function. Prepare the summary chart (as shown above) for each variation of network you train and simulate. Comment on what architecture of network and training algorithm gave the best results. **For your report, provide details of TWO networks implemented that have good results.**

** Training algorithms: There are different training algorithms available for training a feed-forward back-propagation neural network. The algorithm explained to you in class is the TRAINGD algorithm. This is the simplest of all and sometimes ineffective in training a network. Some of the more power full ones are TRAINCGP/CGF and TRAINOSS (for big networks) and TRAINLM (for small networks).