ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ

Νευρωνικά Δίκτυα

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ #1

Διδάσκων Καθηγητής: Δαλιάνης Σωτήριος

ΕΚΠ. ΕΤΟΣ 2022-2023

1 Σκοπός της Άσκησης

Η άσκηση αυτή αποσκοπεί στο να εξοικειώσει τους φοιτητές στην δημιουργία, εκπαίδευση, και αξιολόγηση απλών και σύνθετων Νευρωνικών Δικτύων σε εφαρμογές ταξινόμησης, πρόβλεψης χρονοσειρών και αναγνώρισης προτύπων, με εφαρμογές στο Matlab.

2 Υπόβαθρο – Προετοιμασία

- Βασικές γνώσεις προγραμματισμού σε περιβάλλον MATLAB ή OCTAVE.
- Κατανόηση βασικών αρχών στατιστικής ανάλυσης.
- Εγκατάσταση του περιβάλλοντος MATLAB.
- Εγκατάσταση του Statistics and Machine Learning Toolbox του περιβάλλοντος MATLAB.
- Εγκατάσταση των εφαρμογών Machine Learning and Deep Learning του περιβάλλοντος MATLAB.

3 Εκτέλεση της Άσκησης

Επιλέξτε 2 Data sets από τις βιβλιοθήκες του MATLAB με διαφορετικά χαρακτηριστικά και εισάγετέ τα στο matlab. Στο Α μέρος της άσκησης:

Α.1.-Α.2 Να παράγετε τυχαία δεδομένα από μια διακριτή και μια συνεχή κατανομή πιθανότητας σύμφωνα με το παράδειγμα και να αποθηκεύσετε τα γραφήματα αυτών. Να υπολογίσετε Μέση και Ενδιάμεση τιμή, Διασπορά και Τυπική απόκλιση που θα συμπεριλαμβάνονται τα γραφήματα.

Α.3.-Α4. Επιλέξτε 2 Data sets από τις βιβλιοθήκες του MATLAB με διαφορετικά χαρακτηριστικά και οπτικοποιήστε τα δεδομένα σύμφωνα με το παράδειγμα. Να υπολογίσετε Μέση και Ενδιάμεση τιμή, Διασπορά και Τυπική απόκλιση σε 1 μεταβλητή από κάθε Data set.

Μέρος Β. Να δημιουργήσετε απλά νευρωνικά δίκτυα χρησιμοποιώντας την συνάρτηση network του matlab σύμφωνα με το παράδειγμα.

Β.1. Να δημιουργήσετε ένα νευρωνικό δίκτυο Perceptron σύμφωνα με την περιγραφή της άσκησης 2.2 σελ. 12 από το φυλλάδιο ‘ERGASTHRIO_NURAL NETS’

Β.2. Να δημιουργήσετε ένα νευρωνικό δίκτυο Adaline σύμφωνα με την περιγραφή της άσκησης 2.3 σελ. 18 από το φυλλάδιο ‘ERGASTHRIO NURAL NETS’ χρησιμοποιώντας υποδείγματα κώδικα από το φυλλάδιο ‘nnet’, Κεφάλαιο 3.

Στη συνέχεια να τα εκπαιδεύσετε τα νευρωνικά δίκτυο με κατάλληλα Data sets, να προσομοιώσετε την έξοδα και να υπολογίσετε το ποσοστό σφάλματος.

Αποθηκεύσατε χαρακτηριστικά αποτελέσματα και γραφήματα. Σε ασκήσεις με μεγάλο αριθμό δεδομένων και μεταβλητών συνιστάτε να αποθηκεύτε το workspace σε αρχείο .mat.

Η παρουσίαση της άσκησης γίνεται σε ένα ενιαίο αρχείο world doc ή pdf που περιλαμβάνει σύντομη περιγραφή του προβλήματος, τον κώδικα που χρησιμοποιήσατε, τα αποτελέσματα και τα γραφήματα. Τα αρχεία ανεβαίνουν στο e-class μέχρι την καταληκτική ημερομηνία παράδοσης της άσκησης.

Μέρος Α. Επεξεργασία Δεδομένων στο MATLAB

Α1. Άσκηση #1. Χειρισμός διανυσμάτων και πινάκων.

Να εκτελέσετε επιμέρους κώδικες Matlab σχετικά με τον χειρισμό πινάκων που θα βρείτε στο αρχείο ΄¨Eisagogi sto matlab’¨ σελ. 1-25. Βεβαιωθείτε ότι κατανοείτε την χρήση των συναρτήσεων σχετικά με τη Ταξινόμηση, Μέση και Ενδιάμεση τιμή, Διασπορά και Τυπική απόκλιση, Παραγωγή τυχαίων αριθμών.

Α2. Άσκηση #1 Παραγωγή τυχαίων αριθμών.

Να παράγετε τυχαία δεδομένα από μια διακριτή και μια συνεχή κατανομή πιθανότητας σύμφωνα με το παράδειγμα και να αποθηκεύσετε τα γραφήματα αυτών.

As an example we generate N = 1000 random numbers given by a binomial distribution with n = 9 trials and p = 0:8. Thus each of the 100 random number will be an integer between 0 and 9. Find the result of the code below showing the result in Figures.

```
N = 1000; data = binornd(9, 0.8, N,1); % generate the random numbers
[height,centers] = hist(data,unique(data)) % data for the histogram
bar(centers,height/sum(height))
xlabel('value'); ylabel('experimental probability')
title('Binomial distribution with n=9, p=0.8')
```

Commands to Work with Probability Distributions
rand() uniform distribution
randi() random integers
randn() normal distribution
rande() exponentially distributed
randp() Poisson distribution
randg() gamma distribution
normrnd() normal distribution
binornd() binomial distribution
exprnd() exponential distribution
trnd() Student-t distribution
discrete rnd() discrete distribution

Α3. Άσκηση #1 Data Sets.

Στον πίνακα παρουσιάζεται μία λίστα με συλλογές δεδομένων ή μετρήσεων που είναι ενσωματωμένες στο Matlab και μπορείτε να τις χρησιμοποιήσετε με άδεια Creative Commons. Χρησιμοποιούνται συχνά για την αξιολόγηση μοντέλων μηχανικής μάθησης. Σύνδεσμος:

https://www.mathworks.com/help/matlab/import_export/matlab-example-data-sets.html

Παρόμοια επίσης μια συλλογή εικόνων από την Microsoft.

https://www.microsoft.com/en-us/download/details.aspx?id=52644

Επιπλέον δεδομένα βρίσκονται στη σελίδα του εργαστηρίου στο στο e-class. Φορτώστε στο Matlab 4 data sets με διαφορετικά χαρακτηριστικά ως προς τον τύπο και τον αριθμό

μεταβλητών. Δοκιμάστε να τα επεξεργαστείτε με το εργαλείο διαχείριση δεδομένων του Matlab.

File	Description of Data Set
acetylene.mat	Chemical reaction data with correlated predictors
arrhythmia.mat	Cardiac arrhythmia data from the UCI machine learning repository
batterysmall.mat	Sensor data (voltage, current, and temperature) and state of charge for a Li-ion battery; a subset of the data in [1]
carbig.mat	Measurements of cars, 1970–1982
carsmall.mat	Subset of carbig.mat. Measurements of cars, 1970, 1976, 1982
census1994.mat	Adult data from the UCI machine learning repository
cereal.mat	Breakfast cereal ingredients
cities.mat	Quality of life ratings for U.S. metropolitan areas
discrim.mat	A version of cities.mat used for discriminant analysis
examgrades.mat	Exam grades on a scale of 0–100
fisheriris.mat	Fisher's 1936 iris data
flu.mat	Google Flu Trends estimated ILI (influenza-like illness) percentage for various regions of the US, and CDC weighted ILI percentage based on sentinel provider reports
gas.mat	Gasoline prices around the state of Massachusetts in 1993
hald.mat	Heat of cement vs. mix of ingredients
hogg.mat	Bacteria counts in different shipments of milk
hospital.mat	Simulated hospital data
humanactivity.mat	Human activity recognition data of five activities: sitting, standing, walking, running, and dancing
imports-85.mat	1985 Auto Imports Database from the UCI repository
ionosphere.mat	Ionosphere dataset from the UCI machine learning repository
kmeansdata.mat	Four-dimensional clustered data
lawdata.mat	Grade point average and LSAT scores from 15 law schools

File	Description of Data Set
mileage.mat	Mileage data for three car models from two factories
moore.mat	Biochemical oxygen demand on five predictors
morse.mat	Recognition of Morse code distinctions by non-coders
nlpdata.mat	Natural language processing data extracted from the MathWorks® documentation
ovariancancer.mat	Grouped observations on 4000 predictors [2][3]
parts.mat	Dimensional run-out on 36 circular parts
polydata.mat	Sample data for polynomial fitting
popcorn.mat	Popcorn yield by popper type and brand
reaction.mat	Reaction kinetics for Hougen-Watson model
spectra.mat	NIR spectra and octane numbers of 60 gasoline samples
stockreturns.mat	Simulated stock returns

Α.4. Άσκηση #1 Οπτικοποίηση και επεξεργασία πολυδιάστατων δεδομένων.

This example shows how to visualize multivariate data using various statistical plots. Many statistical analyses involve only two variables: a predictor variable and a response variable. Such data are easy to visualize using 2D scatter plots, bivariate histograms, boxplots, etc. It's also possible to visualize trivariate data with 3D scatter plots, or 2D scatter plots with a third variable encoded with, for example color. However, many datasets involve a larger number of variables, making direct visualization more difficult. This example explores some of the ways to visualize high-dimensional data in MATLAB®, using Statistics and Machine Learning Toolbox™.

In this example, we'll use the `carbig` dataset, a dataset that contains various measured variables for about 400 automobiles from the 1970's and 1980's. We'll illustrate multivariate visualization using the values for fuel efficiency (in miles per gallon, MPG), acceleration (time from 0-60MPH in sec), engine displacement (in cubic inches), weight, and horsepower. We'll use the number of cylinders to group observations.

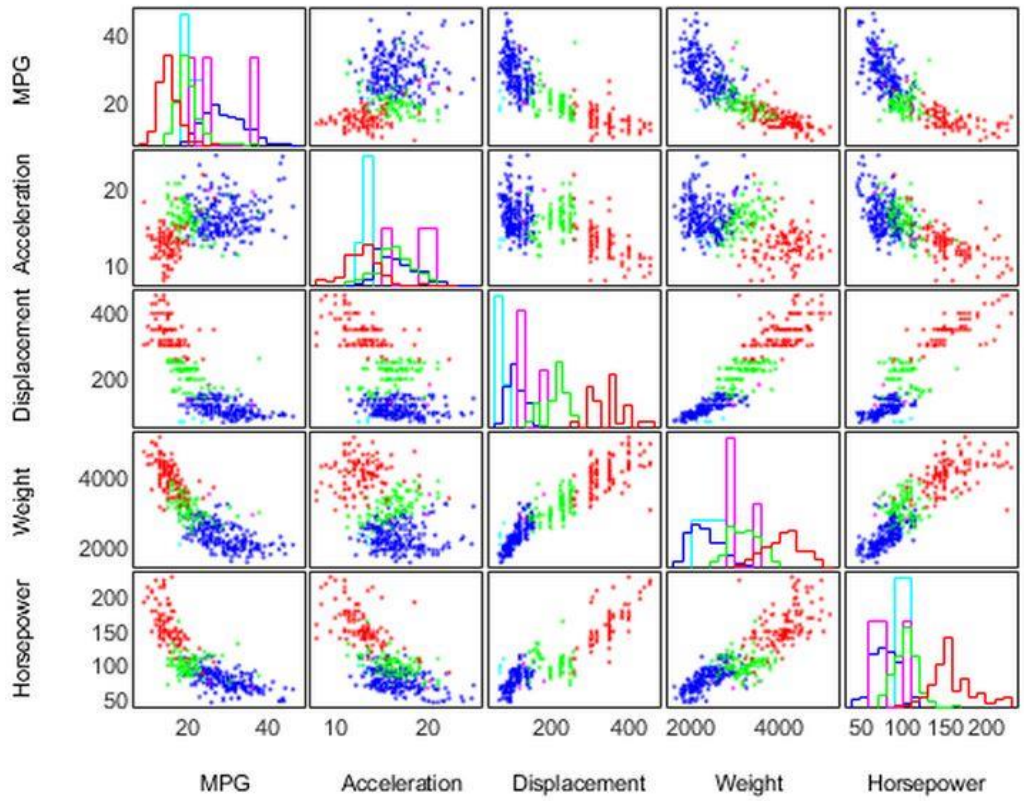
```
load carbig
X = [MPG,Acceleration,Displacement,Weight,Horsepower];
varNames = {'MPG'; 'Acceleration'; 'Displacement'; 'Weight'; 'Horsepower'};
```

Scatterplot Matrices

Viewing slices through lower dimensional subspaces is one way to partially work around the limitation of two or three dimensions. For example, we can use the `gplotmatrix`function to

display an array of all the bivariate scatterplots between our five variables, along with a univariate histogram for each variable.

```
figure
gplotmatrix(X,[],Cylinders,['c' 'b' 'm' 'g' 'r'],[],[],false);
text([.08 .24 .43 .66 .83], repmat(-.1,1,5), varNames, 'FontSize',8);
text(repmat(-.12,1,5), [.86 .62 .41 .25 .02], varNames, 'FontSize',8,
'Rotation',90);
```



The points in each scatterplot are color-coded by the number of cylinders: blue for 4 cylinders, green for 6, and red for 8. There is also a handful of 5 cylinder cars, and rotary-engined cars are listed as having 3 cylinders. This array of plots makes it easy to pick out patterns in the relationships between pairs of variables. However, there may be important patterns in higher dimensions, and those are not easy to recognize in this plot.

Parallel Coordinates Plots

The scatterplot matrix only displays bivariate relationships. However, there are other alternatives that display all the variables together, allowing you to investigate higher-dimensional relationships among variables. The most straight-forward multivariate plot is the parallel coordinates plot. In this plot, the coordinate axes are all laid out horizontally, instead of using orthogonal axes as in the usual Cartesian graph. Each observation is represented in the plot as a series of connected line segments. For example, we can make a plot of all the cars with 4, 6, or 8 cylinders, and color observations by group.

```
Cyl468 = ismember(Cylinders,[4 6 8]);
parallelcoords(X(Cyl468,:), 'group',Cylinders(Cyl468), ...
               'standardize','on', 'labels',varNames)
```

The horizontal direction in this plot represents the coordinate axes, and the vertical direction represents the data. Each observation consists of measurements on five variables, and each measurement is represented as the height at which the corresponding line crosses each coordinate axis. Because the five variables have widely different ranges, this plot was made with standardized values, where each variable has been standardized to have zero mean and unit variance. With the color coding, the graph shows, for example, that 8 cylinder cars typically have low values for MPG and acceleration, and high values for displacement, weight, and horsepower.

Even with color coding by group, a parallel coordinates plot with a large number of observations can be difficult to read. We can also make a parallel coordinates plot where only the median and quartiles (25% and 75% points) for each group are shown. This makes the typical differences and similarities among groups easier to distinguish. On the other hand, it may be the outliers for each group that are most interesting, and this plot does not show them at all.

```
parallelcoords(X(Cyl468,:), 'group',Cylinders(Cyl468), ...
               'standardize','on', 'labels',varNames, 'quantile',.25)
```

Example Data for Classification

To get started try the following example data sets.

Name	Size	Description
Fisher Iris	Number of predictors: 4 Number of observations: 150 Number of classes: 3 Response: species	Measurements from three species of iris. Try to classify the species. For a step-by-step example, see Train Decision Trees Using Classification Learner App.
	Create a table from the `.csv` file: `fishertable = readtable('fisheriris.csv');`	
Credit Rating	Number of predictors: 6 Number of observations: 3932 Number of classes: 7 Response: Rating	Financial ratios and industry sectors information for a list of corporate customers. The respon CCC) assigned by a rating agency.
	Create a table from the `CreditRating_Historical.dat` file: `creditrating = readtable('CreditRating_Historical.dat');`	
Cars	Number of predictors: 7 Number of observations: 100 Number of classes: 7 Response: Origin	Measurements of cars, in 1970, 1976, and 1982. Try to classify the country of origin.

	Create a table from variables in the `carsmall.mat` file:
	`load carsmall`
	`cartable = table(Acceleration, Cylinders, Displacement,...`
	`Horsepower, Model_Year, MPG, Weight, Origin);`	

Arrhythmia	Number of predictors: 279 Number of observations: 452 Number of classes: 16 Response: Class (Y)	Patient information and response variables that indicate the presence and absence of cardiac consequences than false positives classified as “has arrhythmia”.

Name	Size	Description

	Create a table from the `.mat` file:
	`load arrhythmia`
	`Arrhythmia = array2table(X);`
	`Arrhythmia.Class = categorical(Y);`	

Ovarian Cancer	Number of predictors: 4000 Number of observations: 216 Number of classes: 2 Response: Group	Ovarian cancer data generated using the WCX2 protein array. Includes 95 controls and 121 c

	Create a table from the `.mat` file:
	`load ovariancancer`
	`ovariancancer = array2table(obs);`
	`ovariancancer.Group = categorical(grp);`	

Ionosphere	Number of predictors: 34 Number of observations: 351 Number of classes: 2 Response: Group (Y)	Signals from a phased array of 16 high-frequency antennas. Good ("g") returned radar signal ionosphere. Bad ("b") signals are those that pass through the ionosphere.

	Create a table from the `.mat` file:
	`load ionosphere`
	`ionosphere = array2table(X);`
	`ionosphere.Group = Y;`	

Choose Validation Scheme

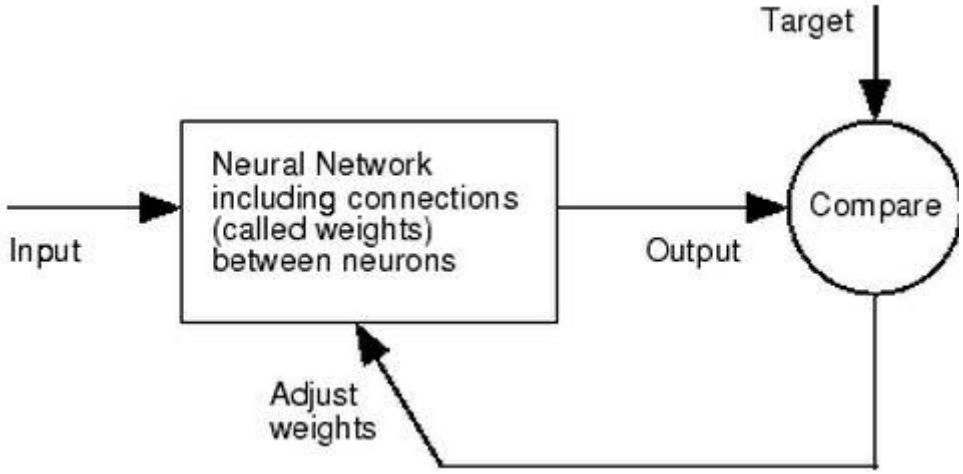
Choose a validation method to examine the predictive accuracy of the fitted models. Validation estimates model performance on new data compared to the training data, and helps you choose the best model. Validation protects against overfitting. Choose a validation scheme before training any models, so that you can compare all the models in your session using the same validation scheme.

Μέρος Β. Σχεδιασμός Νευρωνικών Δικτύων στο MATLAB

1. Shallow Networks for Pattern Recognition, Clustering and Time Series

Neural networks are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the connections between elements largely determine the network function. You can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements.

Typically, neural networks are adjusted, or trained, so that a particular input leads to a specific target output. The next figure illustrates such a situation. Here, the network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically, many such input/target pairs are needed to train a network.



Neural networks have been trained to perform complex functions in various fields, including pattern recognition, identification, classification, speech, vision, and control systems.

Neural networks can also be trained to solve problems that are difficult for conventional computers or human beings. The toolbox emphasizes the use of neural network paradigms that build up to—or are themselves used in— engineering, financial, and other practical applications.

The following topics explain how to use graphical tools for training neural networks to solve problems in function fitting, pattern recognition, clustering, and time series. Using these tools can give you an excellent introduction to the use of the Deep Learning Toolbox™ software:

- Fit Data with a Shallow Neural Network
- Classify Patterns with a Shallow Neural Network
- Cluster Data with a Self-Organizing Map
- Shallow Neural Network Time-Series Prediction and Modeling

Shallow Network Apps and Functions in Deep Learning Toolbox

There are four ways you can use the Deep Learning Toolbox software.

- The first way is through its tools. You can open any of these tools from a master tool started by the command `nnstart`. These tools provide a convenient way to access the capabilities of the toolbox for the following tasks:
  - Function fitting (`nftool`)
  - Pattern recognition (`nprtool`)
  - Data clustering (`nctool`)
  - Time-series analysis (`ntstool`)
- The second way to use the toolbox is through basic command-line operations. The command-line operations offer more flexibility than the tools, but with some added complexity. If this is your first experience with the toolbox, the tools provide the best introduction. In addition, the tools can generate scripts of documented MATLAB® code to provide you with templates for creating your own customized command-line functions. The process of using the tools first, and then generating and modifying MATLAB scripts, is an excellent way to learn about the functionality of the toolbox.
- The third way to use the toolbox is through customization. This advanced capability allows you to create your own custom neural networks, while still having access to the full functionality of the toolbox. You can create networks with arbitrary connections, and you still be able to train them using existing toolbox training functions (as long as the network components are differentiable).
- The fourth way to use the toolbox is through the ability to modify any of the functions contained in the toolbox. Every computational component is written in MATLAB code and is fully accessible.

  These four levels of toolbox usage span the novice to the expert: simple tools guide the new user through specific applications, and network customization allows researchers to try novel architectures with minimal effort. Whatever your level of neural network and MATLAB knowledge, there are toolbox features to suit your needs.

  Automatic Script Generation

  The tools themselves form an important part of the learning process for the Deep Learning Toolbox software. They guide you through the process of designing neural networks to solve problems in four important application areas, without requiring any background in neural networks or sophistication in using MATLAB. In addition, the tools can automatically generate both simple and advanced MATLAB scripts that can reproduce the steps performed by the tool, but with the option to override default settings. These scripts can provide you with templates for creating customized code, and they can aid you in becoming familiar with the command-line functionality of the toolbox. It is highly recommended that you use the automatic script generation facility of these tools.

Deep Learning Toolbox Applications

It would be impossible to cover the total range of applications for which neural networks have provided outstanding solutions. The remaining sections of this topic describe only a few of the applications in function fitting, pattern recognition, clustering, and time series analysis. The following table provides an idea of the diversity of applications for which neural networks provide state-of-the-art solutions.

Industry	Business Applications
Aerospace	High-performance aircraft autopilot, flight path simulation, aircraft control systems, autopilot enhancements, aircraft component simulation, and aircraft component fault detection
Automotive	Automobile automatic guidance system, and warranty activity analysis
Banking	Check and other document reading and credit application evaluation
Defense	Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, and signal/image identification
Electronics	Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, and nonlinear modeling
Entertainment	Animation, special effects, and market forecasting
Financial	Real estate appraisal, loan advising, mortgage screening, corporate bond rating, credit-line use analysis, credit card activity tracking, portfolio trading program, corporate financial analysis, and currency price prediction
Industrial	Prediction of industrial processes, such as the output gases of furnaces, replacing complex and costly equipment used for this purpose in the past
Insurance	Policy application evaluation and product optimization
Manufacturing	Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer-chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, and dynamic modeling of chemical process system
Medical	Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, and emergency-room test advisement
Oil and gas	Exploration
Robotics	Trajectory control, forklift robot, manipulator controllers, and vision systems
Securities	Market analysis, automatic bond rating, and stock trading advisory systems
Speech	Speech recognition, speech compression, vowel classification, and text-to-speech synthesis
Telecommunications	Image and data compression, automated information services, real-time translation of spoken language, and customer payment processing systems
Transportation	Truck brake diagnosis systems, vehicle scheduling, and routing systems

Shallow Neural Network Design Steps

In the remaining sections of this topic, you will follow the standard steps for designing neural networks to solve problems in four application areas: function fitting, pattern recognition, clustering, and time series analysis. The work flow for any of these problems has seven primary steps. (Data collection in step 1, while important, generally occurs outside the MATLAB environment.)

1. Collect data
2. Create the network
3. Configure the network
4. Initialize the weights and biases
5. Train the network
6. Validate the network
7. Use the network

You will follow these steps using both the GUI tools and command-line operations in the following sections:

- Fit Data with a Shallow Neural Network

- Classify Patterns with a Shallow Neural Network
- Cluster Data with a Self-Organizing Map
- Shallow Neural Network Time-Series Prediction and Modeling

network

Create custom shallow neural network

Syntax

```
net = network
net =
network(numInputs,numLayers,biasConnect,inputConnect,layerConnect,outpu
tConnect)
```

Description

`network` creates new custom networks. It is used to create networks that are then customized by functions such as `feedforwardnet` and `narxnet`.

`net = network` without arguments returns a new neural network with no inputs, layers or outputs.

`net = network(numInputs,numLayers,biasConnect,inputConnect,layerConnect,outputConnect)` takes these optional arguments (shown with default values):

`numInputs`	Number of inputs, 0
`numLayers`	Number of layers, 0
`biasConnect`	`numLayers`-by-1 Boolean vector, zeros
`inputConnect`	`numLayers`-by-`numInputs` Boolean matrix, zeros
`layerConnect`	`numLayers`-by-`numLayers` Boolean matrix, zeros
`outputConnect`	1-by-`numLayers` Boolean vector, zeros

and returns

`net`	New network with the given property values

Properties

Architecture Properties

`net.numInputs`	0 or a positive integer	Number of inputs.
`net.numLayers`	0 or a positive integer	Number of layers.
`net.biasConnect`	`numLayer`-by-1 Boolean vector	If `net.biasConnect(i)` is 1, then layer `i` has a bias, and `net.biases{i}` is a structure describing that bias.
`net.inputConnect`	`numLayer`-by-`numInputs` Boolean vector	If `net.inputConnect(i,j)` is 1, then layer `i` has a weight coming from input`j`, and `net.inputWeights{i,j}` is a structure describing that weight.

net.layerConnect	numLayer-by-numLayers Boolean vector	If net.layerConnect(i,j) is 1, then layer i has a weight coming from layerj, and net.layerWeights{i,j} is a structure describing that weight.
net.outputConnect	1-by-numLayers Boolean vector	If net.outputConnect(i) is 1, then the network has an output from layer i, and net.outputs{i} is a structure describing that output.
net.numOutputs	0 or a positive integer (read only)	Number of network outputs according to net.outputConnect.
net.numInputDelays	0 or a positive integer (read only)	Maximum input delay according to all net.inputWeights{i,j}.delays.
net.numLayerDelays	0 or a positive number (read only)	Maximum layer delay according to all net.layerWeights{i,j}.delays.

Subobject Structure Properties

net.inputs	numInputs-by-1 cell array	net.inputs{i} is a structure defining input i.
net.layers	numLayers-by-1 cell array	net.layers{i} is a structure defining layer i.
net.biases	numLayers-by-1 cell array	If net.biasConnect(i) is 1, then net.biases{i} is a structure defining the bias for layer i.
net.inputWeights	numLayers-by-numInputs cell array	If net.inputConnect(i,j) is 1, then net.inputWeights{i,j} is a structure defining the weight to layer i from input j.
net.layerWeights	numLayers-by-numLayers cell array	If net.layerConnect(i,j) is 1, then net.layerWeights{i,j} is a structure defining the weight to layer i from layer j.
net.outputs	1-by-numLayers cell array	If net.outputConnect(i) is 1, then net.outputs{i} is a structure defining the network output from layer i.

Function Properties

net.adaptFcn	Name of a network adaption function or ''
net.initFcn	Name of a network initialization function or ''
net.performFcn	Name of a network performance function or ''
net.trainFcn	Name of a network training function or ''

Parameter Properties

net.adaptParam	Network adaption parameters
net.initParam	Network initialization parameters
net.performParam	Network performance parameters
net.trainParam	Network training parameters

Weight and Bias Value Properties

net.IW	numLayers-by-numInputs cell array of input weight values
net.LW	numLayers-by-numLayers cell array of layer weight values

net.b	numLayers-by-1 cell array of bias values

Other Properties

net.userdata	Structure you can use to store useful values

Examples

Create Network with One Input and Two Layers

This example shows how to create a network without any inputs and layers, and then set its numbers of inputs and layers to 1 and 2 respectively.

```
net = network
net.numInputs = 1
net.numLayers = 2
```

Alternatively, you can create the same network with one line of code.

```
net = network(1,2)
```

Create Feedforward Network and View Properties

This example shows how to create a one-input, two-layer, feedforward network. Only the first layer has a bias. An input weight connects to layer 1 from input 1. A layer weight connects to layer 2 from layer 1. Layer 2 is a network output and has a target.

```
net = network(1,2,[1;0],[1; 0],[0 0; 1 0],[0 1])
```

You can view the network subobjects with the following code.

```
net.inputs{1}
net.layers{1}, net.layers{2}
net.biases{1}
net.inputWeights{1,1}, net.layerWeights{2,1}
net.outputs{2}
```

You can alter the properties of any of the network subobjects. This code changes the transfer functions of both layers:

```
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'logsig';
```

You can view the weights for the connection from the first input to the first layer as follows. The weights for a connection from an input to a layer are stored in `net.IW`. If the values are not yet set, these result is empty.

```
net.IW{1,1}
```

You can view the weights for the connection from the first layer to the second layer as follows. Weights for a connection from a layer to a layer are stored in `net.LW`. Again, if the values are not yet set, the result is empty.

```
net.LW{2,1}
```

You can view the bias values for the first layer as follows.

```
net.b{1}
```

To change the number of elements in input 1 to 2, set each element's range:

```
net.inputs{1}.range = [0 1; -1 1];
```

To simulate the network for a two-element input vector, the code might look like this:

```
p = [0.5; -0.1];
y = sim(net,p)
```

Appendix 1.

Dataset Arrays in the Variables Editor

Open Dataset Arrays in the Variables Editor

The MATLAB Variables editor provides a convenient interface for viewing, modifying, and plotting dataset arrays.

First, load the sample data set, `hospital`.

```
load hospital
```

The dataset array, `hospital`, is created in the MATLAB workspace.

The dataset array has 100 observations and 7 variables.

To open `hospital` in the Variables editor, click **Open Variable**, and select `hospital`.

The Variables editor opens, displaying the contents of the dataset array (only the first 10 observations are shown here).

In the Variables editor, you can see the names of the seven variables along the top row, and the observations names down the first column.

Modify Variable and Observation Names

You can modify variable and observation names by double-clicking a name, and then typing new text.

All changes made in the Variables editor are also sent to the command line.

The sixth variable in the data set, `BloodPressure`, is a numeric array with two columns. The first column shows systolic blood pressure, and the second column shows diastolic blood pressure. Click the arrow that appears on the right side of the variable name cell to see the units and description of the variable. You can type directly in the units and description fields to modify the text. The variable data type and size are shown under the variable description.

Reorder or Delete Variables

You can reorder variables in a dataset array using the Variables editor. Hover over the left side of a variable name cell until a four-headed arrow appears.

After the arrow appears, click and drag the variable column to a new location.

The command for the variable reordering appears in the command line.

You can delete a variable in the Variables editor by selecting the variable column, right-clicking, and selecting **Delete Column Variable(s)**.

The command for the variable deletion appears in the command line.

Add New Data

You can enter new data values directly into the Variables editor. For example, you can add a new patient observation to the `hospital` data set. To enter a new last name, add a character vector to the end of the variable `LastName`.

The variable `Gender` is a `nominal` array. The levels of the categorical variable appear in a drop-down list when you double-click a cell in the `Gender` column. You can choose one of the levels previously used, or create a new level by selecting **New Item**.

You can continue to add data for the remaining variables.

To change the observation name, click the observation name and type the new name.

The commands for entering the new data appear at the command line.

Notice the warning that appears after the first assignment. When you enter the first piece of data in the new observation row—here, the last name—default values are assigned to all other variables. Default assignments are:

- `0` for numeric variables
- `<undefined>` for categorical variables
- `[]` for cell arrays

You can also copy and paste data from one dataset array to another using the Variables editor.

Sort Observations

You can use the Variables editor to sort dataset array observations by the values of one or more variables. To sort by gender, for example, select the variable `Gender`. Then click**Sort**, and choose to sort rows by ascending or descending values of the selected variable.

When sorting by variables that are cell arrays of character vectors or of nominal data type, observations are sorted alphabetically. For ordinal variables, rows are sorted by the ordering of the levels. For example, when the observations of `hospital` are sorted by the values in `Gender`, the females are grouped together, followed by the males.

To sort by the values of multiple variables, press **Ctrl** while you select multiple variables.

When you use the Variables editor to sort rows, it is the same as calling `sortrows`. You can see this at the command line after executing the sorting.

Select a Subset of Data

You can select a subset of data from a dataset array in the Variables editor, and create a new dataset array from the selection. For example, to create a dataset array containing only the variables `LastName` and `Age`:

1. Hold **Ctrl** while you click the variables `LastName` and `Age`.
2. Right-click, and select **New Workspace Variable from Selection** > **New Dataset Array**.

The new dataset array appears in the Workspace window with the name `hospital1`. The Command Window shows the commands that execute the selection.

You can use the same steps to select any subset of data. To select observations according to some logical condition, you can use a combination of sorting and selecting. For example, to create a new dataset array containing only males aged 45 and older:

1. Sort the observations of `hospital` by the values in `Gender` and `Age`, descending.
2. Select the male observations with age 45 and older.
3. Right-click, and select **New Workspace Variables from Selection** > **New Dataset Array**. The new dataset array, `hospital2`, is created in the Workspace window.
4. You can rename the dataset array in the Workspace window.

Create Plots

You can plot data from a dataset array using plotting options in the Variables editor. Available plot choices depend on the data types of variables to be plotted.

For example, if you select the variable `Age`, you can see in the **Plots** tab some plotting options that are appropriate for a univariate, numeric variable.

Sometimes, there are plot options for multiple variables, depending on their data types. For example, if you select both `Age` and `Gender`, you can draw box plots of age, grouped by gender.

Appendix 2.

Παράδειγμα τυποποίησης και εκπαίδευσης ταξινομητή χρησιμοποιώντας δεδομένα από τον σένσορα κίνησης, κινητού τηλεφώνου.

Human Activity Learning Using Mobile Phone Data

Human activity sensor data contains observations derived from sensor measurements taken from smartphones worn by people while doing different activities (walking, lying, sitting etc). The goal of this example is to provide a strategy to build a classifier that can automatically identify the activity type given the sensor measurements.

Contents

Description of the Data

```
The dataset consists of accelerometer and gyroscope data captured at
50Hz. The raw sensor data contain fixed-width sliding windows of 2.56
sec
(128 readings/window). The activities performed by the subject
include:
'Walking', 'ClimbingStairs', 'Sitting', 'Standing',and 'Laying'
```

How to get the data: Execute `downloadSensorData` and follow the instructions to download the and extract the data from the source webpage. After the files have been extracted run `saveSensorDataAsMATFiles`. This will create two MAT files: `rawSensorData_train` and `rawSensorData_test` with the raw sensor data

1. **total_acc_(x/y/z)_train :** Raw accelerometer sensor data
2. **body_gyro_(x/y/z)_train :** Raw gyroscope sensor data
3. **trainActivity :** Training data labels

4. **testActivity :** Test data labels

Reference:

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine. International Workshop of Ambient Assisted Living (IWAAL 2012). Vitoria-Gasteiz, Spain. Dec 2012

Download data from source

If you are running this script for the first time, make sure that you execute these functions.

- `downloadSensorData` : This function will download the dataset and extract its contents to a folder called: UCI HAR Dataset This folder must be present before you execute `saveSensorDataAsMATFiles`

```
if ~exist('UCI HAR Dataset','file')
    downloadSensorData;
end
```

Load data frome individual files and save as MAT file for reuse

- `saveSensorDataAsMATFiles` : This function will load the data from the individual source files and save the data in a single MAT file for easy accesss

```
if ~exist('rawSensorData_train.mat','file') &&
~exist('rawSensorData_test.mat','file')
    saveSensorDataAsMATFiles;
end
```

Load Training Data

```
load rawSensorData_train
```

Display data summary

```
plotRawSensorData(total_acc_x_train, total_acc_y_train, ...
    total_acc_z_train,trainActivity,1000)
```

Create Table variable

```
rawSensorDataTrain = table(...
    total_acc_x_train, total_acc_y_train, total_acc_z_train, ...
    body_gyro_x_train, body_gyro_y_train, body_gyro_z_train);
```

Pre-process Training Data: Feature Extraction

Lets start with a simple preprocessing technique. Since the raw sensor data contain fixed-width sliding windows of 2.56sec (128 readings/window) lets start with a simple average feature for every 128 points

```
humanActivityData = varfun(@Wmean,rawSensorDataTrain);
humanActivityData.activity = trainActivity;
```

Train a model and assess its performance using Classification Learner

```
classificationLearner
```

Additional Feature Extraction

```
T_mean = varfun(@Wmean, rawSensorDataTrain);
T_stdv = varfun(@Wstd,rawSensorDataTrain);
T_pca  = varfun(@Wpca1,rawSensorDataTrain);

humanActivityData = [T_mean, T_stdv, T_pca];
humanActivityData.activity = trainActivity;
```

Use the new features to train a model and assess its performance

```
classificationLearner
```

Load Test Data

```
load rawSensorData_test
```

Visualize classifier performance on test data

Step 1: Create a table

```
rawSensorDataTest = table(...
    total_acc_x_test, total_acc_y_test, total_acc_z_test, ...
    body_gyro_x_test, body_gyro_y_test, body_gyro_z_test);

% Step 2: Extract features from raw sensor data
T_mean = varfun(@Wmean, rawSensorDataTest);
T_stdv = varfun(@Wstd,rawSensorDataTest);
T_pca  = varfun(@Wpca1,rawSensorDataTest);

humanActivityData = [T_mean, T_stdv, T_pca];
humanActivityData.activity = testActivity;

% Step 3: Use trained model to predict activity on new sensor data
% Make sure that you've exported 'trainedClassifier' from
% ClassificationLearner
plotActivityResults(trainedClassifier,rawSensorDataTest,humanActivity
Data,0.1)
```