TRAVEL ROUTE ADVISOR WITH ARTIFICIAL NEURAL NETWORK (TRAANN)

Prepared by
AZNUR HAFEEZ BIN KASWURI
WEK 990179

Under Supervision of
MR. WOO CHAW SENG

Under Moderation of
MR. NOR RIDZUAN DAUD

Travel Route Advisor With Artificial Neural Network (TRAANN)

By

AZNUR HAFEEZ BIN KASWURI
WEK 990179

A Thesis presented to the Faculty of Computer Science and Information Technology of University of Malaya in partial fulfillment of the requirement for the Degree of

BACHELOR OF COMPUTER SCIENCE

UNIVERSITY OF MALAYA

FEBRUARY 2003

Abstract

ABSTRACT

The aim of this project is to develop a Travel Route Advisor, which apply Artificial Neural Network (ANN). The system has the ability to predict the most efficient solution of the available routes.

The first chapter is the introduction. It gives an overview of the project. It describes the objectives, scope and limitation of the project. Chapter 2 is for the literature review. It contains a review of studies done in related fields. It includes some studies in artificial intelligence and artificial neural networks as well.

Chapter 3 is about system methodology. It describes the steps taken to accomplish this project and development method taken. Chapter 4 is for system analysis, which cover the system requirements and the project schedule. Chapter 5 is about system design. It includes the design of the system proposed. It contains the core modules of the system, namely database, ANN and GUI modules.

Chapter 6 is for the system implementation. It describes the transformation of the modules into programming codes. Chapter 7 is for system testing. Tests had been made to verify whether the system have met the requirements. The discussion chapter, which is chapter 8, contains the discussion on the results, the problem encountered, the solution taken and the conclusion of the project.

Acknowledgement

ACKNOWLEDGEMENT

In the name of God, Most Gracious, Most Merciful. AlhamduliLlah, thank to Allah, The Almighty for giving me the strength and confidence to complete this project.

Firstly, I would like to thank Mr. Woo Chaw Seng, my supervisor for giving me a chance to develop this system, which I proposed to him at the beginning of the semester. Secondly, I would like to thank for his constructive advice, generous guidance, encouragement, support and supervision along the project. He is also the one who encouraged me to do my project in English and using MATLAB, which I thought unattainable at first. His hardworking and kindness in helping me throughout the project is deeply appreciated.

Thanks to Mr. Mohd Nor Ridzuan bin Daud, who contributed suggestion and comments to this project. He is so considering and I really feel grateful, as he became my moderator for this project.

Besides, I would like to express my gratitude to my parent Mr. Haji Kaswuri Keman and Mrs. Aminah Hamzah for the support, understanding and inspiration as early form my birth until now.

My thanks go also to all my course mates & friends for their valuable idea. Thanks to Mr. Mohd Shahidan Hassan, Mr. Kamarul Anwar Abdul Aziz, Mr. Muhd Faez Lutpi & Mr Anwar Ahmad for their comments and advice throughout the project.

Thank you very much.

"So, verily, with every difficulty, there is relief:
Verily, every difficulty there is relief."

The Holy Quran,
Chapter 94 (Al Sharh),
Verses 5 & 6

Table of Contents

Title	Page
ABSTRACT	iii
ACKNOWLEDGEMENT	vi
TABLE OF CONTENTS	*ix*
LIST OF TABLES	xiv
LIST OF FIGURES	xv
CHAPTER I: INTRODUCTION	1
Project Overview	2
Project Definition	3
Current Paradigm's Weakness	3
Strength of Travel Route Advisor With ANN	4
Project Limitation and Its Goal	5
Project Objectives	6
Project Scopes	7
Report Outline	8
Summary	9
CHAPTER II: LITERATURE REVIEW	11
Introduction	12

Artificial Intelligence 13
Definition 13
Artificial Intelligence Approaches In Prediction System 14
Artificial Neural Networks 16
Definition 16
Biology Neural Network 18
The Characters of The ANN 21
Neural Network's Capabilities 24
Competitive network 27
Malaysian Transportation System 29
Road 29
Railway 31
Air 32
Sea / water 33
Summary 34
CHAPTER III: METHODOLOGY 35
Introduction 36
Research Method 36
Library 36
Internet 37
Individuals 37

Personal books and lecture notes 37
System Development Life Cycle 38
Analysis 42
Design 42
Coding 42
Testing 42
Implementation 43
Operation and Maintenance 43
Project Schedule 43
Summary 46
CHAPTER IV: SYSTEM ANALYSIS 47
Introduction 48
Functional Requirements- 49
Functional Requirements Modules 50
Non Functional Requirements 52
User-friendly interface 52
Response Time 53
Reliability 53
Efficiency 53
System Development Requirements 54
Hardware Requirements 54

Software Requirements 55
Summary 57
CHAPTER V: SYSTEM DESIGN 58
Introduction 59
System Design Overview 60
Database 62
ANN 63
Graphic User Interface (GUI) 65
Data flow of TRAANN 67
Summary 69
CHAPTER VI: SYSTEM IMPLEMENTATION 70
Introduction 71
System Development Discussions 72
The System Design Discussions 72
The Development Of The Database 73
The Development Of The ANN 75
The Development Of The Graphic User Interface 80
Summary 82
CHAPTER IV: SYSTEM TESTING 83
Introduction 84
Functional Testing of Each Functional Module 84

Database Testing 84
ANN Module Testing 87
GUI Testing 89
Summary 91
CHAPTER VIII: DISCUSSION 92
Introduction 93
Results 94
Problems Encountered and Solutions Taken 95
Lack of knowledge on ANN 95
Inexperience in the programming language used 96
Further Research 97
Summary 98
User Manual Of TRAANN 99
APPENDIX 103
BIBLIOGRAPHY 156

Chapter I: Introduction

1.0 CHAPTER I: INTRODUCTION

1.1 Project Overview

The aim of this project is to develop a Travel Route Advisor, which apply Artificial Neural Network (ANN). The system has the ability to predict the most efficient solution of the available routes.

The system's main purpose is to generate the best route for users who are planning to travel by road, air or sea within Peninsular Malaysia. The solution generated by the system will consider such aspects as distance, time consumed and cost.

1.2 Project Definition

1.2.1 Current Paradigm's Weakness

The similar systems available today are based on real-time operation. They have to analyze current situation before giving the solution to the users. The weaknesses of these systems are:

1) They need the information of current situation to come out with solutions. Of course they need agents to provide the input data. In the case of Traffic.com, an online system that provide traffic update for major cities of the USA, they are using speed sensors located at the roadsides to detect congestions for every 60 seconds. They also have the traditional ways to collect information; using the helicopters and phone calls. It means, a large amount of money have to be allocated for the system.
2) They only warn users after the traffic jams occur. They could not predict the traffic situation for the next hours or days. Users can't do their travel planning with the system.
3) They only focus on road traffic. In the real situation, we might use air and water transport also to get us to places.

1.2.2 Strength of Travel Route Advisor With ANN

Travel Route Advisor with Artificial Neural Network (TRAANN) is a system that predicts the best route using the power of artificial neural network. They do the predicting job based on the information it have, not reporting the situation like conventional system. The advantages of this system compared to others are:

1) It doesn't need information on current traffic situation. The system are much more economical than the others since it don't need speed sensors, helicopters or effort from the traffic watcher that surely not cheap.
2) It predicts the most efficient route before any traffic congestions or incidents happens. It helps user to plan their travel wisely.
3) This system is not only covering the road situation, but it also considers the air and sea transportation to come out with the best solution.

1.2.3 Project Limitation and Its Goal

TRAANN may not give the best solution for the routes, but acceptable. It is because it does not consist enough parameters needed to predict the traffic situation. It is a project to prove that Artificial Neural Networks could be used to problems of this kind. Even if it has enough parameters to deal with, it still, will not give the real situation of the traffic since it only doing the predicting job, not reporting the traffic situation.

The project will be considered succeed if it could gives rational solutions to users on how to travel within five cities of Peninsular Malaysia; whether by road, by air or by sea.

1.3 Project Objectives

The objectives of this project are:

1) To develop a system that advises users on travel routing which implement artificial neural network.
2) To apply the concept of artificial neural network in prediction system to travel route advisor.
3) To predict the most efficient route according to parameters like distance, time consumed and cost.

1.4 Project Scopes

The project objectives of TRAANN are to develop a system that could advises users with the power of artificial neural network. The artificial neural network will give a solution based on the parameters that it have. The parameters are:

1. Distance between cities by road, air and sea
2. Time consumed for each route
3. Cost for each route
4. The origin cities and destination.

Solutions will be figured out by evaluating these parameters. The artificial neural network will be trained to give the optimum values for each parameter so it will produce the best solution.

The project will cover at least 5 major cities in Peninsular Malaysia but it is expandable.

1.5 Report Outline

The main purpose of this report is to state in full the processes involved in the development of TRAANN. The report consists of several chapters.

The first chapter is the introduction. It gives an overview of the project. It covers project overview, project definition, the objectives and project scopes. Project definition consists the current paradigm's weakness of similar system, the strengths of TRAANN, project limitations and project's goals.

Chapter 2 is the literature review. It contains a critical review of studies done in related fields. Specifically, it includes some studies in artificial intelligence and ANN as well as other artificial intelligence approaches relevant to this project.

Chapter 3 is a chapter on system analysis and design. It describes the development model chosen and steps taken to accomplish this project. The discussion of the system requirements, the project schedule and preliminary design of the system are attached.

The conclusion is done in Chapter 4 which is contains the summary of important points that discussed in each chapter.

1.6 Summary

This chapter gives the concept of the system. Briefly, it describes the aim, main purpose, current paradigm's weakness of similar system, the strengths of TRAANN, project limitations and project's goals, the objectives, the scopes and the outline for the report.

The concept is, it will implement the artificial neural networks to generate the best solution to travel by air, road or sea within Peninsular Malaysia. It is generated by the artificial neural network, which weighs up parameters such as travel cost, time consumed and distance.

The major advantage of TRAANN is it predicts the best route economically compared to the current available systems, which report the traffic situations. Though, the solution given won't give the exact current situation, as it is a routing system based on prediction, not from reports.

TRAANN is considered achieved its goal if it could predict the travel solution among five cities in Peninsular Malaysia rationally.

This report will consist of eight chapters including the introduction, literature review, methodology, system analysis, system design, system implementation, system testing and discussion.

The next chapter will be the literature review, which contains researches and studies on the related field of the system.

Chapter II: Literature Review

2.0 CHAPTER II : LITERATURE REVIEW

2.1 Introduction

This chapter reviews the related topics that have been studied. It cover the artificial intelligence and the approaches that might be used to develop the project. Then, the discussion is about the artificial neural network since the approach has been chosen to complete the system. On the last part, Malaysian transportation system will be reviewed to have the understanding of Malaysian transportation networks.

2.2 Artificial Intelligence

2.2.1 Definition

According to John McCarthy, a pioneer of AI whom suggested the name Artificial Intelligence at the 1956 conference, "It (artificial intelligence) is the science and engineering of making intelligent machine, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but Artificial Intelligence does not have to confine itself to methods that are biologically observable." (McCarthy,2002)

As a conclusion from the quote of John McCarthy, we understand that artificial intelligence is a field of study that imitates human's thinking ability to make intelligent machines.

2.2.2 Artificial Intelligence Approaches In Prediction System

Two artificial intelligence approaches have been found which can be applied in the prediction system are the expert system and ANN.

Expert System

Definition

An Expert System is an artificial intelligence program incorporating a knowledge base and an inference system. It is a highly specialized piece of software that attempts to duplicate the function of an expert in some field of expertise (Frenzel,1987). The program acts as an intelligent consultant or advisor in the domain of interest, capturing knowledge of one or more experts. Non-experts can then tap the expert system to answer questions, solve problems, and make decisions in the domain.

The Components of Expert System

The key components of expert system are the knowledge base, the inference engine and the user interface.

The knowledge base is the place where the knowledge of the experts is stored. There are different methods for representing knowledge in the expert system. The designer can choose among predicate calculus, lists, frames, semantic networks, scripts and production rules. The knowledge can be acquires from two sources, the primary and the secondary. The primary source comes directly from the experts. The secondary source is from printed or electric materials such as the literatures, web pages, case studies etc.

The inference engine is the software that implements a search and pattern-matching operation. We can say that the function of the inference engine is hypothesis proving. The inference engine uses the heuristic algorithm to search for the solution. The solution sometimes is not the best but it can solve the problem.

The user interface is software that lets the users communicate with the system. The user interface asks questions or presents menu choices for entering initial information in the database. The user interface contains planned questions, statements, or menu sequences. It helps the experts systems to narrow the scope by logical deduction. Then, it displays the answer or solution for the problem as the output.

2.3 Artificial Neural Networks

2.3.1 Definition

According to W.S. Sarle, "There is no universally accepted definition of an artificial neural network. But perhaps most people in the field would agree that a neural network of many simple processors ("units"), each possibly having a small amount of local memory. The units are connected by communication channels ("connections") which usually carry numeric (as opposed to symbolic) data, encoded by any of various means. The units operate only on their local data and on the inputs they receive via the connections. The restriction to local operations is often relaxed during training."(WS Sarle)

Artificial neural networks have been developed as generalizations of mathematical models of human neural biology, based on the assumption that:

1. Information processing happens at many simple elements called neurons.
2. Signals are passed between neurons over connection links.
3. Each connection links has an associated weight, which in a typical neural net, multiplies the signal transmitted.

4. Each neuron applies an activation function to its net input to determine its output signal.

2.3.2 Biology Neural Network

A brief discussion of some features of biological neurons may help clarifying the most important characteristics of artificial neuron. There is a close analogy between the structure of a biological neuron and the artificial neuron or processing element.

A biological neurons has three types of components :

1. Dendrites
2. Cell body / soma
3. Axon

Dendrites receive electric impulse signals from other neurons. The signals are transmitted across a synaptic gap by a chemical process. The cell body or soma, sums the incoming signals. When sufficient input received, the cell fires. It transmits a signal over its axon to other cells. A biological neuron is illustrated in Figure 1.1.

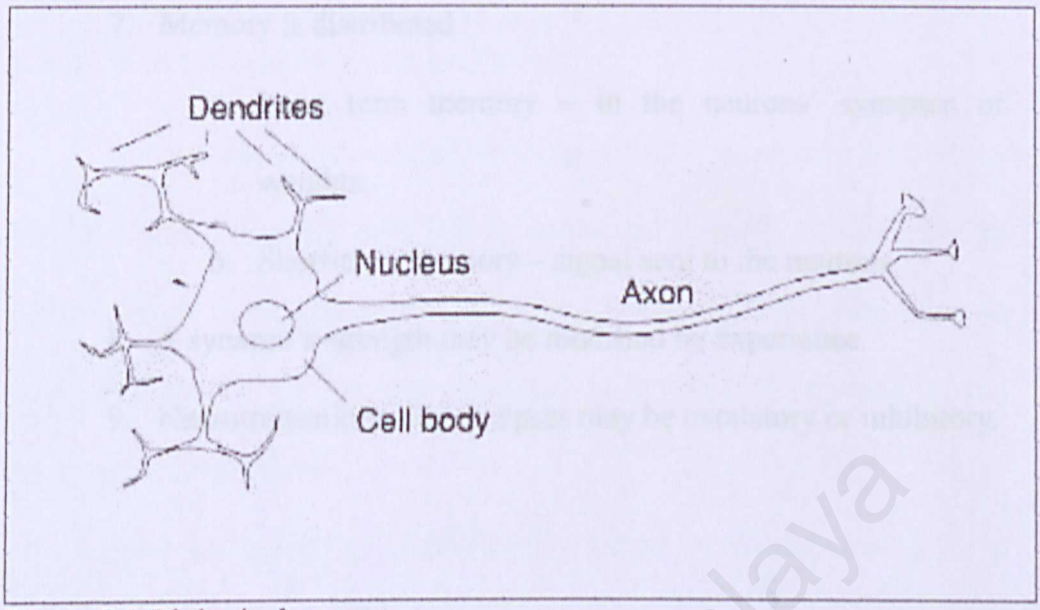


Figure 2.1 Biological neuron

Several features of the processing elements of artificial neural networks are suggested by these properties of biological neurons (Fausette,1994):

1. The processing elements receives many signals.
2. Signals may be modified by a weight at the receiving synapse.
3. The processing element sums the weighted inputs.
4. Under appropriate circumstances (sufficient input), the neuron transmits a single output.
5. The output from a particular neuron may go to many other neurons.
6. Information processing is local.

7. Memory is distributed :
    a. Long term memory – in the neurons' synapses or weights.
    b. Short-term memory – signal sent to the neurons.
8. A synapse's strength may be modified by experience.
9. Neurotransmitters for synapses may be excitatory or inhibitory.

2.3.3 The Characters of The ANN

A neural network consists of a large number of simple processing elements called neurons, unit, cell or nodes. Each neuron is connected to other neurons by means of directed communication links, each with an associated weight. The weights represent information being used by the net to solve the problem.

Each neuron has an internal state, called its activation or activity level. It is a function of the inputs it has received. A neuron can send only one signal at a time, although that signal is broadcast to several other neurons.

A neural network is characterized by:

1. Architecture – pattern of connections between the neurons.
2. Training algorithm-methods of determining weight on the connections
3. Activation function

Architecture

The architecture or the pattern of connections between the neurons refers to network's wiring detail. It includes the detail of connecting units,

the direction of connection and the value of the weighted connections. The steps to determine the pattern of connectivity are:

1. System designer specifies which units are connected and which direction.
2. The weight values are learnt during a training phase.

Training algorithm

The method of setting the values of the weights (training) is an important characteristic of different neural nets. We will discuss two types of training – supervised and unsupervised – for a neural network.

Supervised learning

In this mode of training, training is accomplished by presenting a sequence of training vectors or patterns, each with associated target output vector. The weights are then adjusted according to a learning algorithm to minimize errors. This type of learning is similar to teacher's learning. The teacher may be training set of data or being an observer who grades the performance of the students. Reinforcement will be given to help the students.

The tasks that suitable for supervised learning include decision-making, map associations, memorizing information and generalization.

Unsupervised learning

In this mode of training, a sequence of input vectors is provided, but no target vectors are specified. The net modifies the weights so that the similar input vectors are assigned to the same output unit. The neural net will produce a representative vector for each cluster formed. The daily example for unsupervised learning is learning by doing, without a teacher.

Unsupervised learning is not well understood and is still the subject of much research. It is a great interest to the government of big countries, because many military situations don't have a data set available until a conflict arises. Supervised learning technique, on the other hand, has achieved a reputation for producing good result in practice and applications.

Activation function

Activation function refer to the basic operation of an artificial neuron involves summing its weighted input signal and applying an output. For the input units, this function is the identity function. Typically, the same activation function is used for all neurons in any particular layer of a neural net, although this is not required. In most cases, a nonlinear activation function is used.

2.3.4 Neural Network's Capabilities

Below are the example ranges from commercial successes to areas of active research of the neural network.

Signal processing

One of the first commercial applications was to suppress noise on a telephone line. The neural net used for this purpose is a form of ADALINE (Adaptive Linear Elements). The need for adaptive echo cancellers has become more pressing with the development of transcontinental satellite links for long distance telephone circuits. The adaptive noise cancellation idea is quite simple. At the end of a long distance line, the incoming signal is applied to both the telephone system component and the adaptive filter (the ADALINE type of neural net). The difference between the output of the hybrid and the output of the ADALINE is the error. It is then used to adjust the weights on ADALINE. The ADALINE is trained to remove the noise from the hybrid's output signal.

Control

As an example of the application of the neural networks to control problems, consider the task of training neural "truck backer-upper" to

provide steering directions to a trailer truck attempting to back up to a loading dock. The neural net is able to learn how to steer the truck in order for the trailer to reach the dock, starting with the truck and trailer in any initial configuration that allows enough clearance for a solution to be possible.

Pattern recognition

One specific area in which neural network applications have been developed is the automatic recognition of handwritten characters. The variation in sizes, positions, and styles of writing make this a difficult problem for traditional techniques.

Medicine

One of many example of the application of neural networks to medicine was an application called "Instant Physician". The idea behind this application is to train an auto associative memory neural network to store a large number of medical records, including information on symptoms, diagnosis, and treatment for a particular case. After training, the net can be presented with input consisting of a set of symptoms; it will then find the full stored pattern that represents the best diagnosis treatment.

Prediction

A very common problem is that predicting value of a variable given historic value of it. Economic and meteorological models spring to mind. Neural networks have frequently been shown to out perform traditional techniques like ARIMA and frequency domain analysis.

2.3.5 Competitive network

The neurons in a competitive layer distribute themselves to recognize frequently presented input vectors.

Architecture

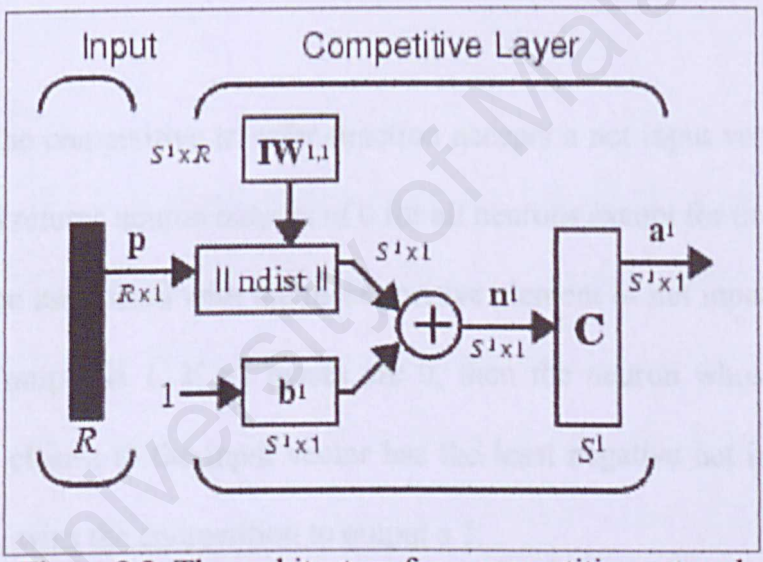


Figure 2.2: The architecture for a competitive network.

The |dist| box in this figure accepts the input vector p and the input weight matrix IW1,1, and produces a vector having S1 elements. The

elements are the negative of the distances between the input vector and vectors iIW1,1 formed from the rows of the input weight matrix.

The net input n1 of a competitive layer is computed by finding the negative distance between input vector p and the weight vectors and adding the biases b. If all biases are zero, the maximum net input a neuron can have is 0. This occurs when the input vector p equals that neuron's weight vector.

The competitive transfer function accepts a net input vector for a layer and returns neuron outputs of 0 for all neurons except for the winner, the neuron associated with the most positive element of net input n1. The winner's output is 1. If all biases are 0, then the neuron whose weight vector is closest to the input vector has the least negative net input and, therefore, wins the competition to output a 1.

The competitive with unsupervised learning net had been chosen as the artificial neural network type for the system.

2.4 Malaysian Transportation System

Malaysian transportation is developing rapidly day by day. Malaysian major transportation includes transportation by road, railway, air and sea. The new integrated public transport system is expected to alleviate traffic congestion problem in the city and provide a more efficient transportation system for the public.

2.4.1 Road

In this region, Malaysian roads are among the best. Driving through the North-South Highway on the West Coast or through the East-West Highway on the east coast could make journey to Malaysia from Thailand and Singapore. Many other highways are connecting places in Malaysia especially in Klang Valley and West Coast of Peninsular Malaysia. They include KESAS highway, SPRINT highway, Kerinchi Link and the new-elevated PROLINTAS highway. Buses, taxis and rent car service available widely for the traveling purpose.

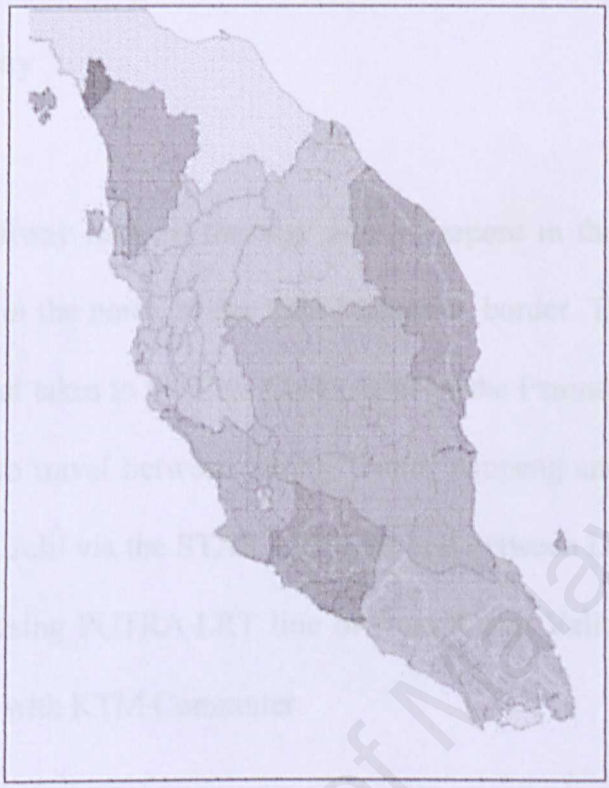


Figure 2.3 Peninsular Malaysia Road Map

2.4.2 Railway

The railway runs all the way into Singapore in the south and to Padang Besar in the north, at the Thai-Malaysian border. There is also an eastern link that takes to the East Coast states of the PeninsulaCommuters are also able to travel between Sentul Timur, Ampang and the National Stadium Bukit Jalil via the STAR-LRT line, and between Lembah Subang and Gombak using PUTRA-LRT line or from Klang Valley to Rawang and Seremban with KTM Commuter.

2.4.3 Air

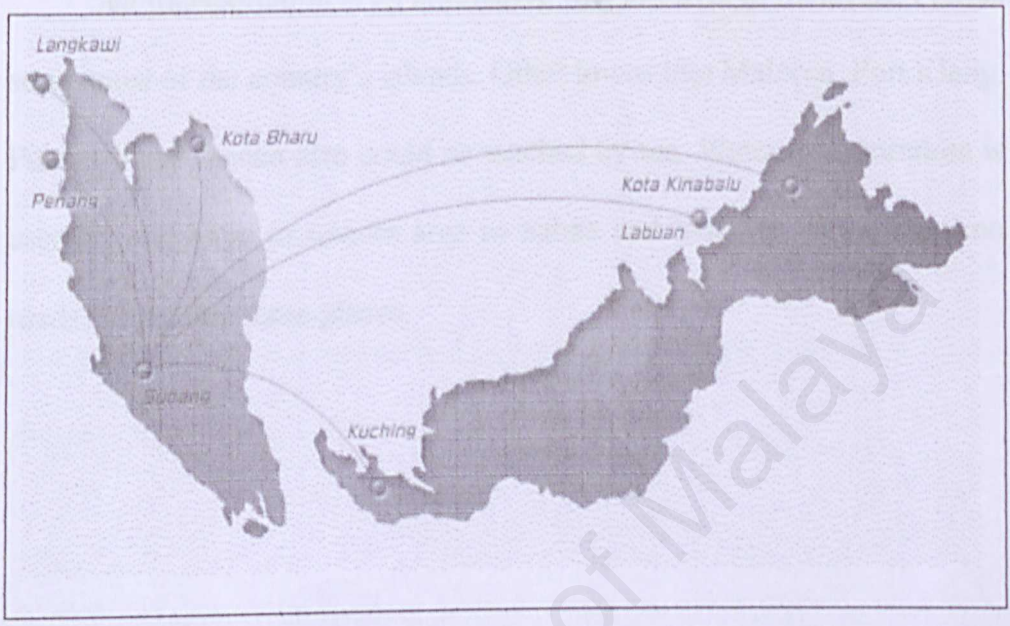


Figure 2.4 Malaysian airway map.

Numerous regional airports in 13 state capitals as well is in other strategic locations serve Malaysia. Penang, Langkawi, Alor Setar, Ipoh, Kota Baru, Kuala Terengganu, Kuantan, Johor Bahru, Kuching, Kota Kinabalu Labuan, Tawau, and Sandakan.could be connected from Kuala Lumpur daily by air. Airlines companies also serve the community by operating many services in some of Malaysia's remote towns and villages especially in North Borneo. This services are provided by local airline companies include Malaysia Airlines, Air Asia and Pelangi Air.

2.4.4 Sea / water

Sea transportation is an alternative way to travel in Malaysia. Ferries serve most of the country's islands. Other towns like Malacca, Port Klang, Penang and Kuantan also could be reached by sea. Water transportation is used by and large in remote area in Sabah and Sarawak as there are no roads connecting these places.

2.5 Summary

The chapter contains the review of the related studied field. Artificial intelligence approach has been chosen to develop the system. Then, the artificial neural network has been chosen rather than the expert system for the system's orientation. Among the types of the neural networks, competitive net with unsupervised learning rule had been chosen. A review on Malaysian transportation also had been done in this chapter.

Chapter III: Methodology

3.0 CHAPTER III : METHODOLOGY

3.1 Introduction

This chapter explain the research approaches taken during the system development. It also discusses the system development life cycle chosen to complete the system.

3.2 Research Method

There are 4 major types of resources referred to carry out the research including the library, the Internet, individuals and personal books or lecture notes.

3.2.1 Library

References had been done in the library to give a clear comprehension on ANN, the ANN approaches taken to solve similar problems and to gain knowledge on researches done by the ANN's pioneers. There are also a number of projects related to ANN developed by the computer science and engineering students that helped to create new solution to complete this project. The references had been done in University Malaya's Main Library, the faculty's document room and Engineering Faculty's Library.

3.2.2 Internet

The Internet is one of the major resources to gain information. Information gathered from the Internet includes the information on currently available traffic routing system, traveling information and programming techniques. The traveling information includes distances from each city in Peninsular Malaysia, available transportation services, fares and estimated time for each route.

3.2.3 Individuals

Some individuals had been referred to gain the personal knowledge on their fields of interest. Mostly the knowledge is gained from the lecturers, tutor, friends and there about the concept of ANN, the project development, and some on the transportation information.

3.2.4 Personal books and lecture notes

Personal books and lecture notes are referred to review some knowledge that helps on the development of the system. They include the Matlab programming guide and project management knowledge.

3.3 System Development Life Cycle

The waterfall model with prototyping has been chosen as the development process model to complete the system. Discussion on two models, which inspired the waterfall with prototyping; the waterfall model and the prototyping model, will be discussed first.

Waterfall Model

The advantages of using waterfall model for this project is:

- It is useful in helping system designer to lay out what he/she needs to do in the project. This model presents a very high level view of what goes on during development.
- It suggest the sequence of events that system designer should expect to encounter.

While the advantage of using this development model is :

- It does not reflect the way code is really developed except for very well understood problems. A system is usually developed with a great deal of iteration.

Conclusion: Inflexible partitioning of the project into these distinct stages. Received system sometimes unusable, as they do not meet the customer's requirement.

Prototyping Model

The superior of the prototyping model is:

- It is an approach where a simple running program will be developed first, modified and changed to suit the objective and target.

Although, the feature that not suitable for the project is

- It is more suitable for the system, which is needed to be developed and use urgently.

Conclusion: Problem with prototyping model is in planning, costing & estimating a project. Although the project can be developed faster, but it is outside of the system developer' experience.

Waterfall Model with Prototyping

The advantages of using waterfall model with prototyping method compared to other model are:

- It combines the advantages of some other models, the waterfall model and the prototyping model.
- Better solution for the problem that occurs on their own.
- The development process more visible for the users and designers.

- System prototype can be developed to give end udders a concrete impression of the system capability. The prototype may therefore help in establishing & validating systems requirements.
- Validation and verification are enabled. Validation will ensures that the system has implemented all the requirements, so that each system's function can be traced back to a particular requirement in the specification. Verification will ensure that each function works correctly. It is made in order to get the high quality of the implementation on the system.

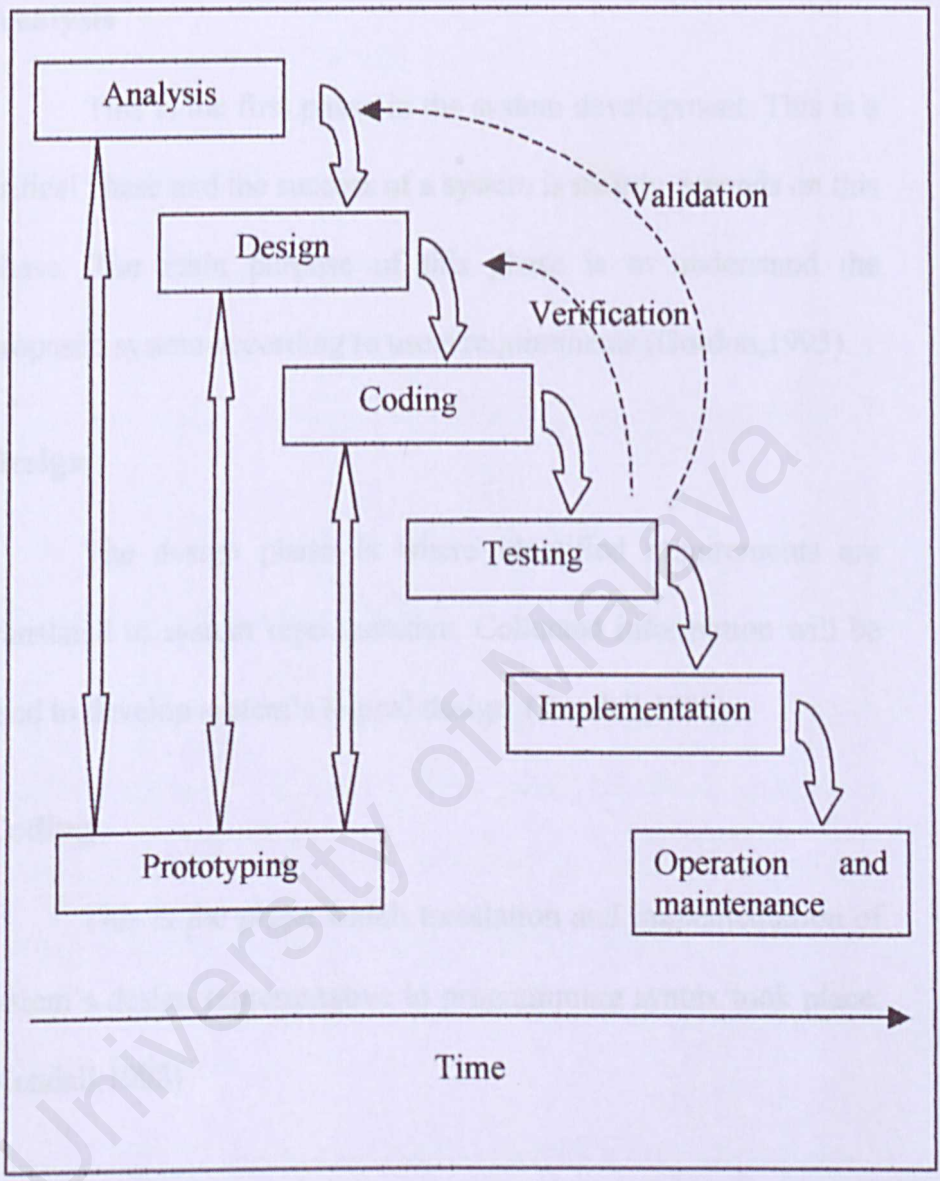


Figure 3.1 Waterfall model with prototyping

3.3.1 Analysis

This is the first phase in the system development. This is a critical phase and the success of a system is mainly depends on this phase. The main purpose of this phase is to understand the proposed system according to users requirements.(Gordon,1995)

3.3.2 Design

The design phase is where identified requirements are translated to system representative. Collected information will be used to develop system's logical design. (Kendall,1998)

3.3.3 Coding

This is the phase which translation and implementation of system's design representative to programming syntax took place. (Kendall,1998)

3.3.4 Testing

This is an important phase to ensure the quality of the system that will be developed is satisfying the user's requirements, and efficiently functional. The specification, design and system coding will be reexamined. This is where the verification and

validation process will be done. This phase is considered success when the system functions well. (Kendall,1998)

3.3.5 Implementation

The developed system will be implemented in hardware and software environment that will be used. The whole system will be tested to ensure the system run without error. (Kendall,1998)

3.3.6 Operation and Maintenance

This is the final phase of the system development process. The maintenance of the system will be done constantly. Observations and modification will be done currently to ensure the developed system really satisfying. (Kendall,1998)

3.3.7 Project Schedule

Table 3.1 shows the project schedule for the system while figure 3.2 shows the system development Gannt Chart.

Table 3.1 Project schedule

Activities	Date	
	From	To
Literature review	1/6/2002	10/7/2002
System analysis	1/6/2002	10/7/2002
Methodology	11/7/2002	20/8/2002
System Design	21/8/2002	31/9/2002
Phase II	27/5/2002	15/10/2002
System design	1/10/2002	20/10/2002
Coding	21/10/2002	20/12/2002
System testing	21/12/2002	31/1/2003
Documentation	1/6/2002	31/1/2003

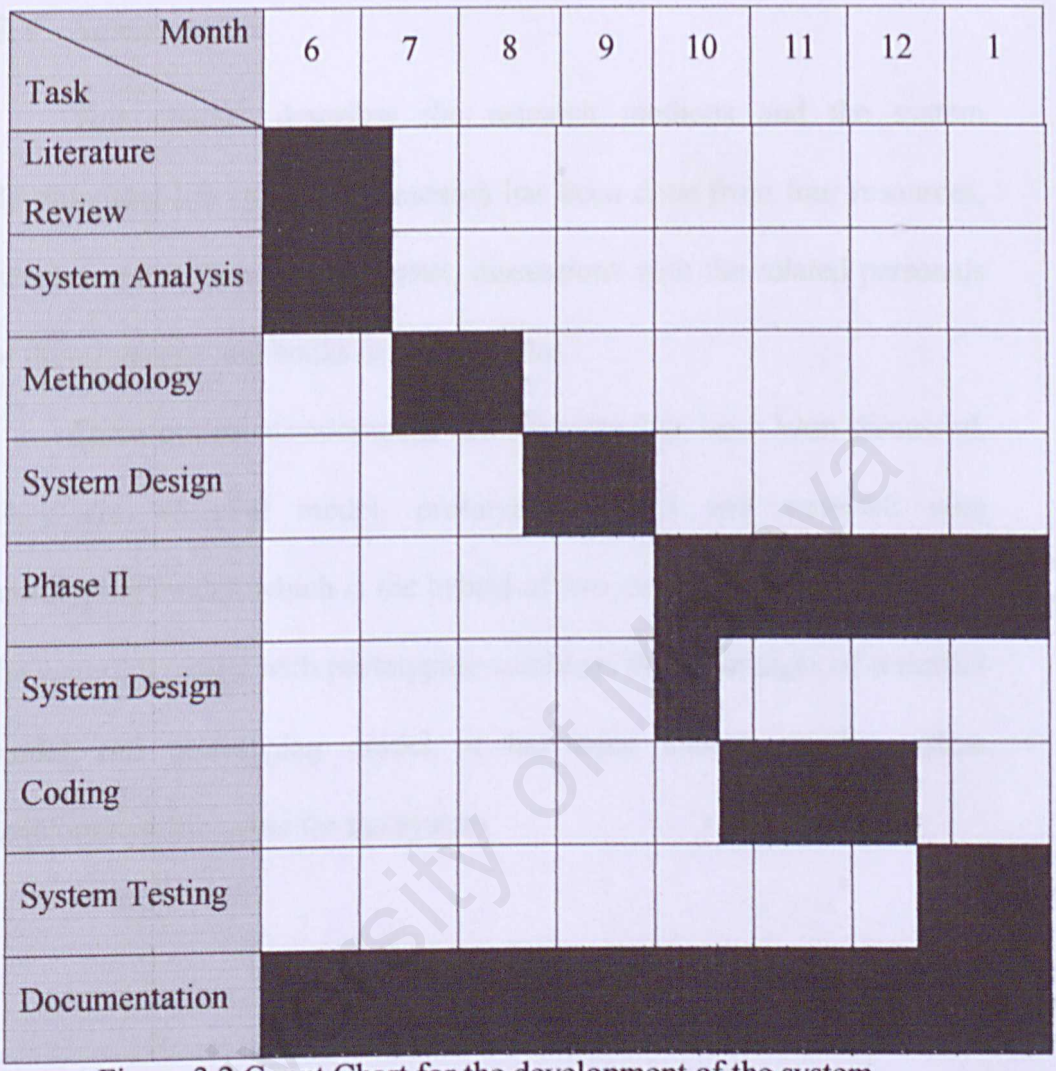


Figure 3.2 Gannt Chart for the development of the system

3.4 Summary

This chapter describes the research methods and the system development life cycle. The research has been done from four resources, which are the library, the Internet, discussions with the related personals of their interests and books or lecture notes.

Three system development life cycle models have been discussed. They are waterfall model, prototyping model and waterfall with prototyping model, which is the hybrid of two model discussed earlier. As the waterfall model with prototyping combines the advantages of waterfall model and prototyping model, it has been chosen as the system development life cycle for the system.

Chapter IV: System Analysis

4.0 CHAPTER IV: SYSTEM ANALYSIS

4.1 Introduction

This chapter describes all of the requirements needed for the system development. It consists of functional requirements, non-functional requirements and development requirements. The development requirements cover both hardware and software requirements.

4.2 Functional Requirements-

The functional requirements consist of a set of tasks that the software is required to perform. From this requirement list, a guideline on the overall layout of the software can be planned. The functional requirements defined for this system are:

4.2.1 Able to predicts the most efficient route by air, sea and road.

The system has the ability to predict the most efficient solution of the available routes. The solution generated by the system will consider such aspects as distance, time consumed and cost.

4.2.2 Able to accepts and stores distances, travel times and costs as its parameters.

The system has the ability to store distances, travel times and costs in the database as its parameters. The parameters also should be editable from time to time to meet the changes on the parameters especially on costs and travel times.

4.2.3 Functional Requirements Modules

The functional requirements are broken into sub-components or modules. These modules taken together represent the actual whole software package. These modules are :

Database

The database is where the information on distance, time consumed and cost for every route are stored. It send the traveling data to the ANN for the learning purpose.

ANN

The neural networks receive the parameters as the input from database module. It will count the parameters and send an output, to represent the solution, whether to travel by air, road or sea.

Graphic User Interface (GUI)

GUI module will receive a query from user. The input will be the origin place and the destination that user would like to go. The module then sends a query to the ANN to seek for the relevant data. If the information is there, it will send the solution from the ANN. The solution given is in numeric form. This module then translates it to either traveling by air, road or sea and displays it to the user.

4.3 Non Functional Requirements

Non-functional requirements refer to functions that are essential but which are not directly related to the core function. For TRAANN, the non-functional requirements defined are:

4.3.1 User-friendly interface.

The interface of the system will be made as user friendly as possible. The interface will be specially modeled for ease of learning. The concept of ease of learning is that a user will not be using this software often (like office software) and therefore will not be able to memorize the functions of too much buttons or complex navigation. Its goal will be to make the software's functions easy to recognize with visual cues. This is slightly different from ease of use where the goal is to make completing easy and fast. Ease of use includes a lot of features that allow work to be done as quick as possible but at the expense of being complex. Some of the features of ease of learning are the exact opposite of ease of use.

4.3.2 Response Time

Response time involves the time it takes from the computer processing the user's request. The goal of this is analysis is to determine the best compromise between features and the time it takes for something to happen.

4.3.3 Reliability

The system must be made stable on the target operating system specifications

4.3.4 Efficiency

Efficiency is defined here as making optimal use of space, time and other resources.

4.4 System Development Requirements

The functional requirements consist of a set of tasks that the software is required to perform. The requirement is divided into two component; the hardware requirements and the software requirements.

4.4.1 Hardware Requirements

The hardware requirement of the system is as follows:

- 233 MHz microprocessor (any type)
- 128 MB RAM
- 4 GB Hard Disk Drive
- Standard PC device (CD-ROM, Floppy Drive, etc)

The hardware requirement is based on the minimum requirements for the software that will be used to develop the system.

4.4.2 Software Requirements

4.4.2.1 MATLAB 6.1

MATLAB is a software package that is powerful in numeric computation, data analysis and graphics. It is a programming language that is capable of performing a wide variety of engineering computation. MATLAB is stands for 'MATrix LABoratory'. With its- matrix-based techniques problems can be solve easily without having to write detailed program codes as in traditional languages like C, Basic and Fortran. Under its easy to use environment, MATLAB has the capability to perform numerical analysis, matrix computation, signal processing, and plant control. There are two elements of MATLAB that are used for the development; Neural Network Toolbox and Graphical User Interface Development.

MATLAB Neural Network Toolbox (NNT)

The MATLAB Neural Network Toolbox is an all-purpose neural network environment. It is a powerful tool in creating neural network as it includes almost everything for the neural network environment purpose. It also include high level network in the toolbox.

Graphical User Interface Design Environment (GUIDE)

GUIDE which stands for Graphical User Interface Development Environment is a set of MATLAB tools designed to make building GUIs (Graphical User Interface) easier and faster.

4.5 Summary

The requirements needed for the system development consists of functional requirements, non-functional requirements and development requirements. The development requirements include both hardware and software requirements.

There are two functional requirements decided for the system. It should have the ability to predict the most efficient route by air, sea and road, and it should be able to accept and stores distances, travel times and costs as its parameters. The functional requirements are broken into three modules. There are the database, ANN and GUI module.

The non-functional requirements aimed for the system includes user-friendly interface, quick response time, reliable and efficiency.

The system development requires a personal computer with minimum specification needed to run MATLAB as it is the software used to develop the system.

Chapter V: System Design

5.0 CHAPTER V: SYSTEM DESIGN

5.1 Introduction

This chapter describes the components of the system, which hold the specific functions that should be done by the system. It involves the system design overview, major components of the system and the data flow of the system.

This chapter also describes how the components are joined together to make the system runs.

5.2 *System Design Overview*

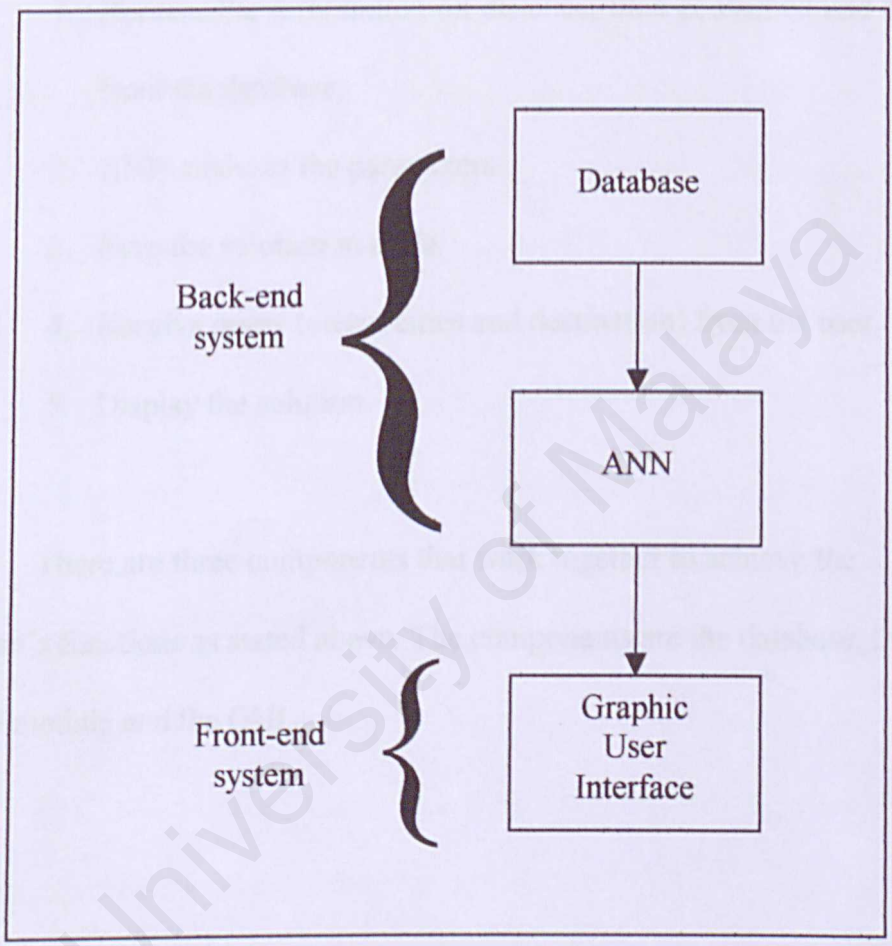


Figure 5.1 System design of Travel Route Advisor with artificial Neural Networks

Travel Route Advisor with artificial Neural Networks is a system designed to generate the best solution or route. The functions of the system are:

1. Receive the information on distance, time consumed and cost from the database.
2. ANN analyzes the parameters.
3. Save the solution in a file.
4. Receive query (origin cities and destination) from the user.
5. Display the solution.

There are three components that work together to achieve the system's functions as stated above. The components are the database, the ANN module and the GUI.

5.2.1 Database

The ANN module functions to store the information on transportation such as the distances between cities, fares and time consumed for each route; air, road and sea. Using this information, the ANN runs to make solution for the best route.

The database is where the information on distance, time consumed and cost for every route are stored. It will keep the numerical data in the form of matrices. The ANN will access these data as the parameters.

The traveling fare data that will be used for the parameters are referred to Malaysian Airlines for air route, bus fare for road route and any various cruise and ferry fare for sea route.

This module is the back-end system for TRAANN. It will not be available for the end user to view the module.

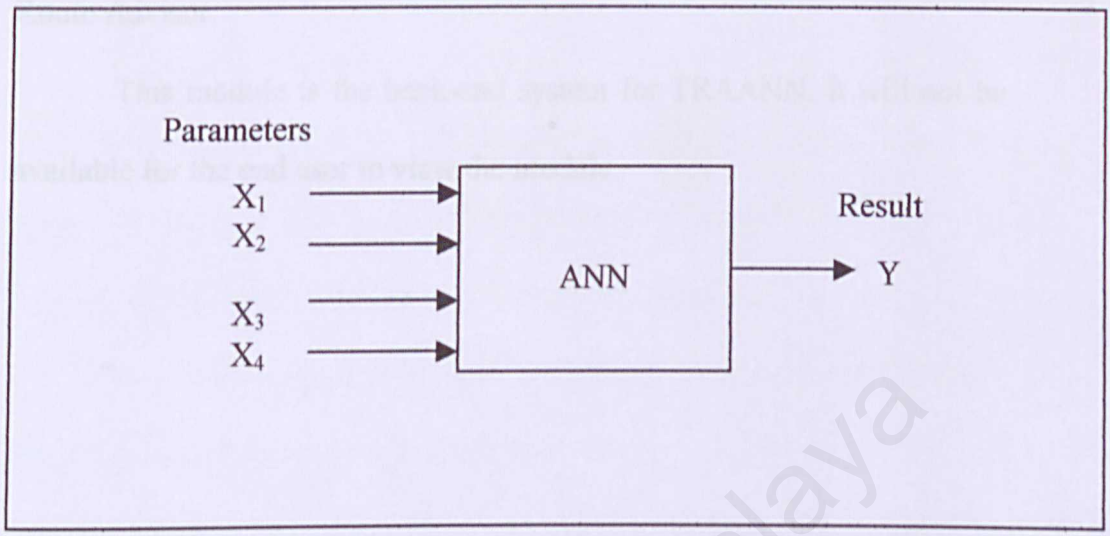


Figure 5.2: ANN design for TRAANN

5.2.2 ANN

The neural networks receive the parameters from the database. It will count the parameters and send numeric output, to represent the solution, whether to travel by air, road or sea.

The results is stored is an .m type file. It receives the numeric solutions given by the ANN learning. There are 10 numeric values to represent the solution routes. The numeric solutions are stored in a 10 x 1 matrix.

The numeric data stored will be accessed by the graphic user interface to be displayed later. The GUI to display the solution will access this file.

Competitive network is chosen as the ANN type for the Travel Route Advisor.

This module is the back-end system for TRAANN. It will not be available for the end user to view the module.

5.2.3 Graphic User Interface (GUI)

GUI is the only module of TRAANN that visible to the end user. The module accesses the file that contains the solutions stored by the ANN processing module. The solutions received are in numeric form. The GUI module then transfers them into texts so for the users viewing.

The interface consists of two pop menus for the lists of departure and destination cities, a text box to display the route suggestions, the Enter button to trigger the selection and the clear button to wipe out the suggestion displayed.

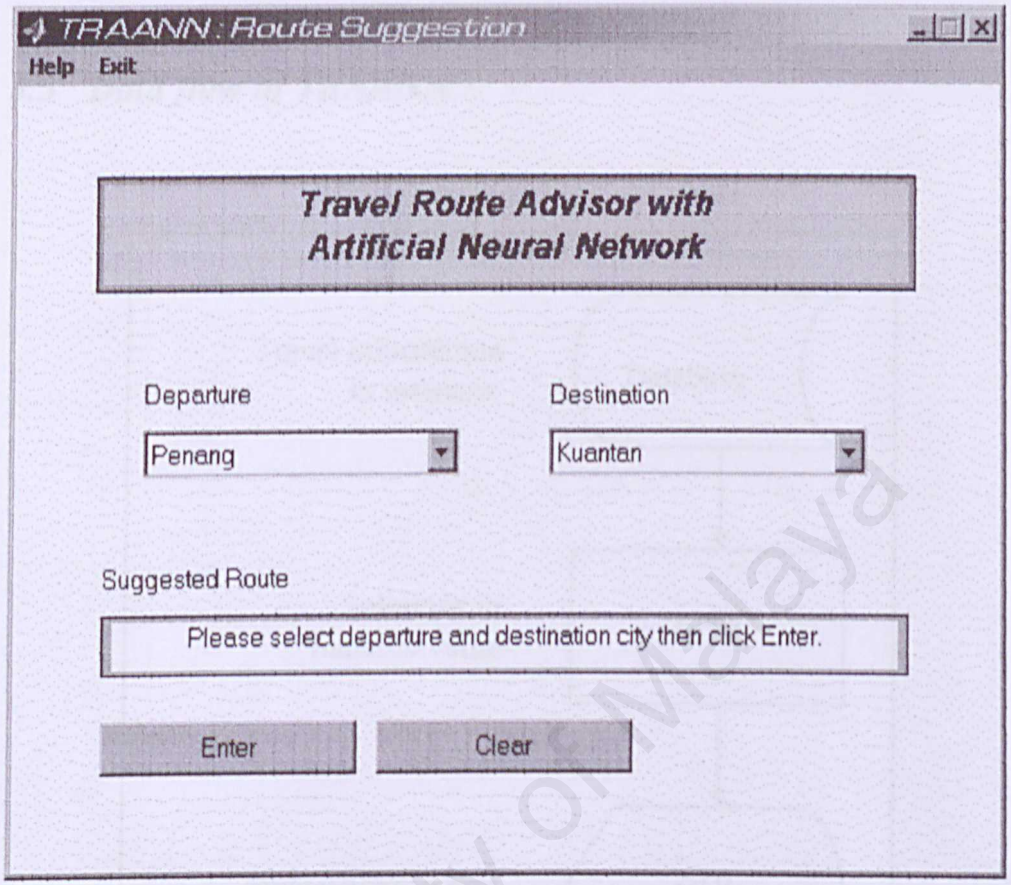


Figure 5.3: Graphic user interface for TRAANN

5.3 Data flow of TRAANN

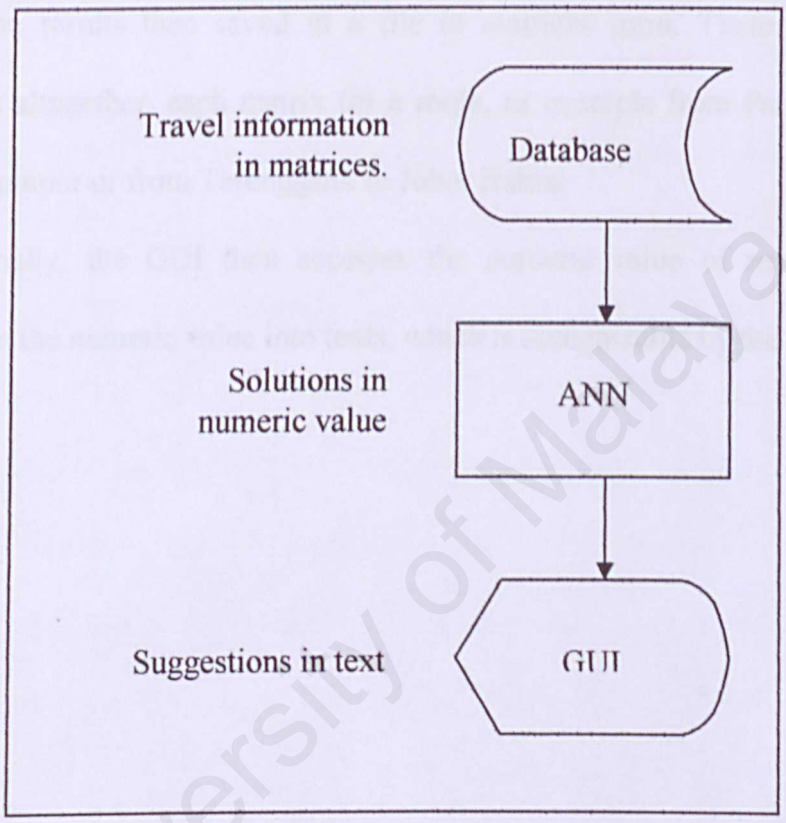


Figure 5.4: Data flow diagram for TRAANN

The data flow of TRAANN starts from the database where it keeps information on traveling within 5 cities; Penang, Kuala Terengganu, Kuantan, Kuala Lumpur and Johor Bahru, in 10 matrices. It consists the information on distances, time consume and fares between each cities and each route; by air, by road and by sea.

Then, the ANN for the learning process will access the data. The process then returns a value, which represents the solution for the travel method; either by air, by road or by sea.

The results then saved in a file in matrices form. There are 10 matrices altogether, each matrix for a route, as example from Penang to Kuala Lumpur or from Terengganu to Johor Bahru.

Finally, the GUI then accesses the numeric value of results. It translates the numeric value into texts, which is recognizable by the users.

5.4 Summary

This chapter described the system design of TRAANN. The description includes the system design overview and the data flow of the system.

The system design overview elaborates the main modules of TRAANN. The main modules of the system namely the database, ANN module and graphic user interface. Every module has their own functions to run the system. The database contains the traveling information. The ANN processing module runs the ANN to generate the solution and saved in the result storage. The GUI interact the system with the user.

The data flow of the system start from the database where the information on travels is kept. The data the transferred to the ANN as the parameter and saved. Finally, the data is transferred to the GUI for the users' viewing.

Chapter VI: System Implementation

6.0 CHAPTER VI : SYSTEM IMPLEMENTATION

6.1 Introduction

This chapter shows the development of the system. It refers to the transformation process of modules and algorithms into executable computer commands.

The computer command is in MATLAB as it is the program that had been chosen to run the system.

6.2 System Development Discussions

6.2.1 The System Design Discussions

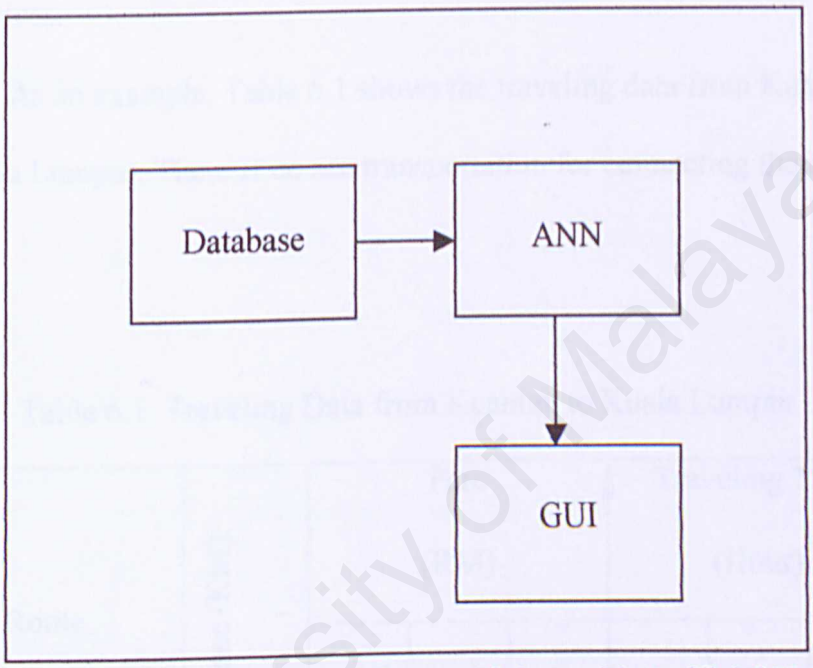


Figure 6.1: Overview of Travel Route Advisor with Artificial Neural Network System Design.

6.2.2 The Development Of The Database

The database contains data on distances between cities, fare rates, and time consumed for each route on each destination. The data is stored in matrices.

As an example, Table 6.1 shows the traveling data from Kuantan to Kuala Lumpur. There is no sea transportation for connecting these two cities.

Table 6.1: Traveling Data from Kuantan to Kuala Lumpur

Route	Distance (KM)	Fare (RM)			Traveling Time (Hour)		
		Air	Road	Sea	Air	Road	Sea
Kuantan – Kuala Lumpur	274	112	14.2	-	0.7	3	-

A 4 x 2 matrix represents the travel information as shown in figure 6.2. Each element in the matrix represents a travel data. The value –1 located in sea fare and sea's travel time mark that there is no sea transportation for the route. While the '0' element (labeled as null value in

Figure 6.2) does not represent any value. It only placed to complete the matrix's element.

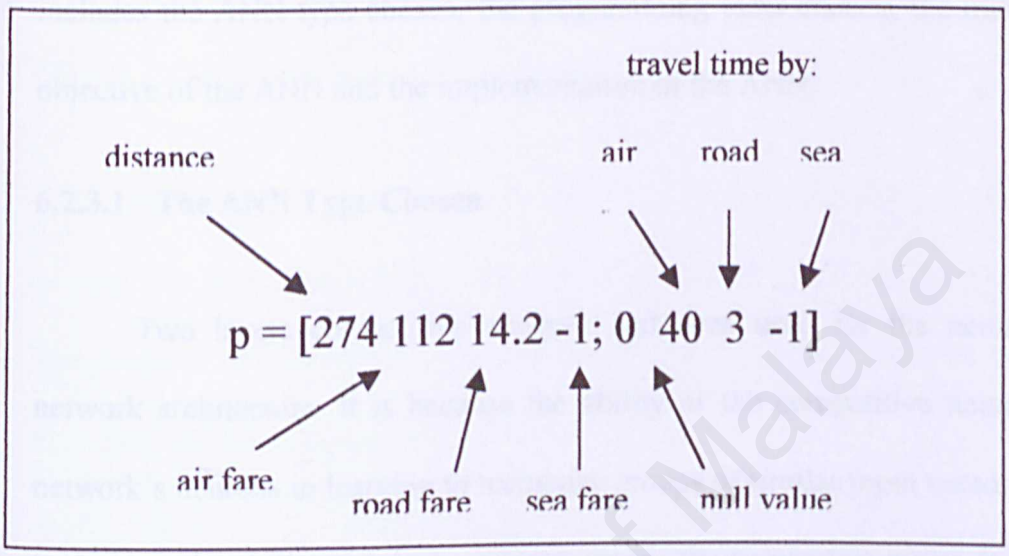


Figure 6.2: Travel data representation in a matrix

After each declaration of variable p, the module calls the function ann.m to activate the ANN simulation using the parameters. The function is called by the command:

$$R(8) = ann(p)$$

R(8) refers to the location of the element where the solution will be placed in R, a 10 x 1 matrix which store the solutions simulated by the ANN.

6.2.3 The Development of The ANN

This part describes the development process of the ANN. It includes the ANN type chosen, the programming tools chosen, the main objective of the ANN and the implementation of the ANN.

6.2.3.1 The ANN Type Chosen

Two layers competitive network had been used for the neural network architecture. It is because the ability of the competitive neural network's neurons in learning to recognize groups of similar input vectors. It also has the ability to select a winner among the parameters given. It is the important property of a neural network that TRAANN needed, as it has to generate the best route as the winner among others.

The network will learn itself with the unsupervised learning rules. Using the unsupervised learning rules, it is not a necessary to give a target and supervise the neural networks' performance. The neurons train themselves as training by doing. The neurons of competitive networks learn to recognize groups of similar input vectors and set a method on how to manipulate them to produce the output.

For the competitive network, it uses random order incremental training with learning functions. It trains a network with weight and bias

learning rules with incremental updates after each presentation of an input and the inputs are presented in random order. (MATLAB Help,2001)

6.2.3.2 The Programming Tool Chosen

MATLAB has been chosen as the programming tool for the project. It is because MATLAB has the Neural Network Toolbox integrated in the program. The Neural Network Toolbox has all the functions and the properties of the ANN needed.

6.2.3.3 The Objective of The ANN

The objective of the ANN chosen is to generate the best route as the winner using the parameters given. The parameters help the ANN as the information it needs to judge to produce the best route among others.

6.2.3.4 The Implementation Of The ANN

The ANN is developed as a function named ann.m. The database module will call this function after each declaration of parameters for each route.

As an example, after declaring the Kuantan to Kuala Lumpur route's parameter, p = [274 112 14.2 -1; 0 40 3 -1], the program called the

ann.m function. The ann.m function receives matrix p, as the parameter to run the ANN. The process will be repeated for every parameter, from the first to the last.

After receiving the parameter, the ANN function create a new competitive two-neuron layer with two input elements using the command:

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

Then, it initializes the weights to the center of the input ranges. The weight initialized will be the initial weight it uses. It will be increased or decreased by the ANN to suit the parameters. The initialization of the weight use the command :

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

The ANN modules then set the network for training while the system developer sets the value of the epoch. The 300 epochs time has been chosen as it shows the stability of the result when reaching this point and above. As it reached the constancy, the larger value of epochs are not necessary as it increases the time response of the system. The command to set the network training is represented by:

```
net.trainParam.epochs = 300
net = train(net,p)
```

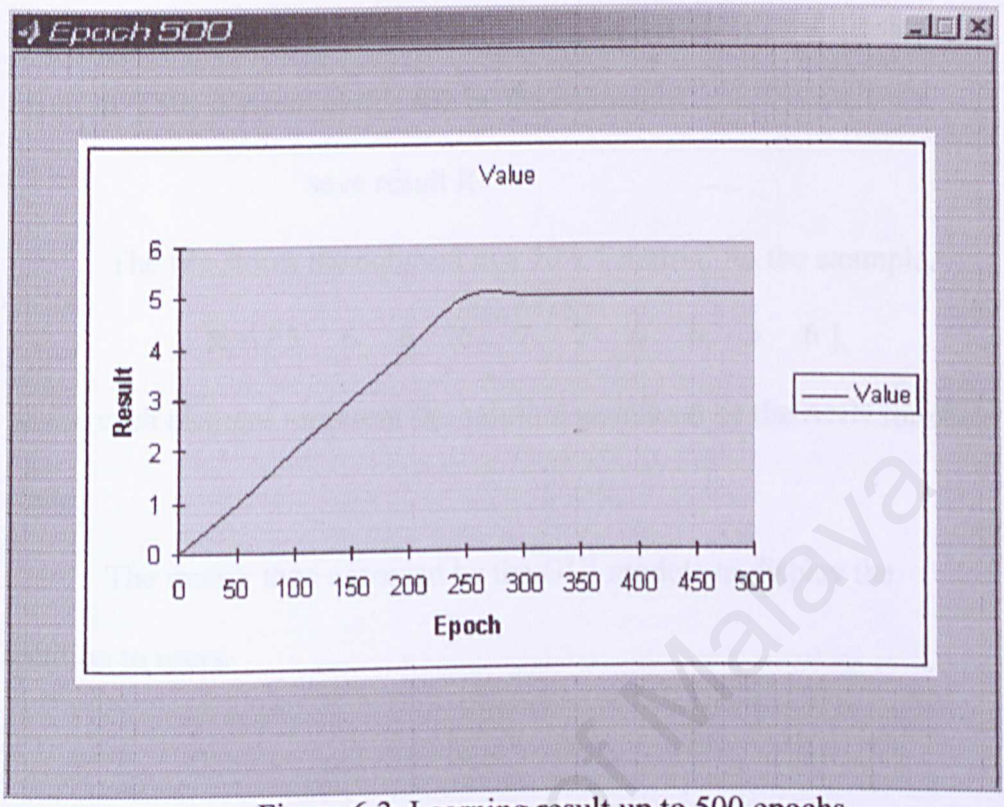


Figure 6.3: Learning result up to 500 epochs

The ANN module then set the training function for competitive networks with the command:

net.trainFcn

After the original vectors as the input to the network is provided, the ANN module simulate the network and convert its output vectors to class indices. The commands are:

a = sim(net,p)

ac = vec2ind(a)

result = sum (ac)

The results from the ANN simulation is sent to a .mat file named result.m. They are sent to this file by the command:

save result R

The file stores the solution in a 10 x 1 matrix. As the example,

R = [5 6 6 6 7 7 6 6 5 6],

where each element represent the solution generated by the ANN for each route.

The matrix then accessed by the GUI module to display the solution to users.

6.2.4 The Development of Graphic User Interface

The graphic user interface of the system is developed using the Graphic User Interface Development Environment (GUIDE) provided in MATLAB. GUIDE will generate its source code in an .m file while the developer's part is as easy as drag and drop the elements such as the scroll bar, button and text then writing the source code for the callback function of the elements.

The graphic user interface of the system contains two list boxes; each for the departing location and destination, two push button, each to display the solution and to wipe out the display and an edit text column to display the solution suggested by the system.

It main tasks is receive user inputs of departure and destination cities, and converting the numeric value passed by the ANN into understandable text for user viewing. Table 6.2 shows the text representation of the numeric value for the GUI.

It also contain help menu at the top of the interface to guide users.

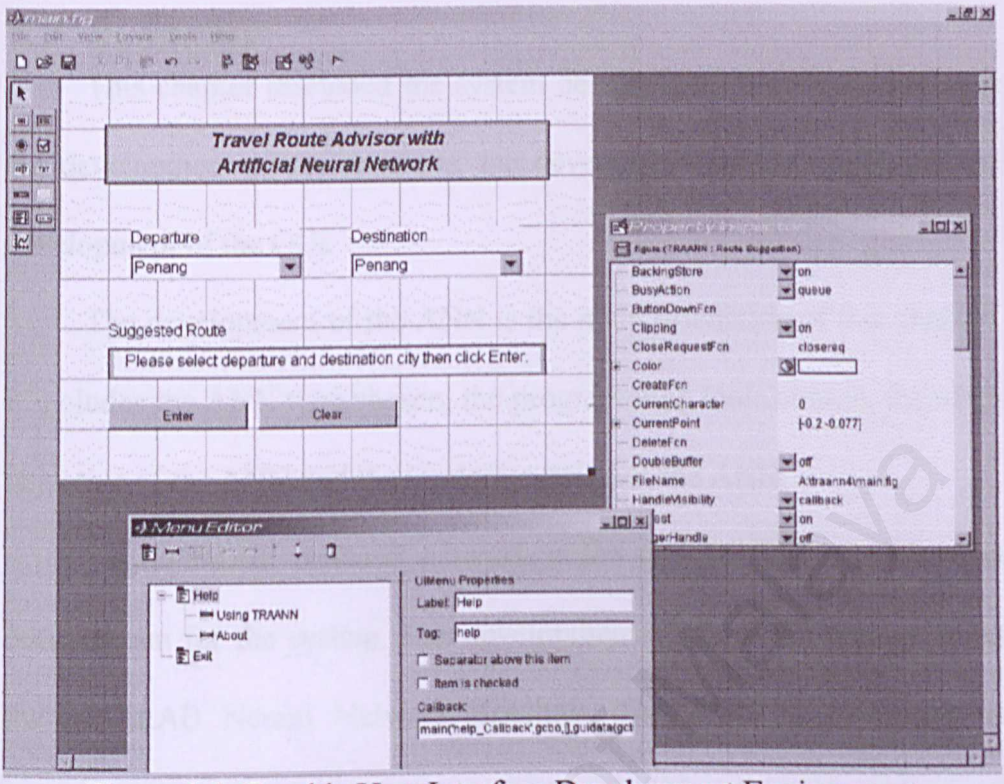


Figure 6.4: Graphic User Interface Development Environment

Table 6.2: Text Representation of Numeric Value

Numeric value from the ANN	Representation of text form displayed by GUI
5	By Air
6	By Road
7	By Sea

6.3 Summary

This chapter discussed the system development process. It includes the development of the database, the development of the ANN and the development of the GUI.

The development of the ANN is the main discussion of this chapter. It includes the ANN type chosen, the programming tools chosen, the main objective of the ANN and the implementation of the ANN.

The competitive neural network is the type of the ANN that had been chosen for the system. The development of the ANN will be using the MATLAB Neural Network Toolbox. The ANN is developed to generate solution on the best route to travel with information given on travel distances, fares and time taken for each route. The implementation shows on how the proposed ANN had being implemented to the system using MATLAB commands.

The development of the GUI is based on the Graphic User Interface Development Environment (GUIDE) integrated with MATLAB.

Chapter VII: System Testing

7.0 CHAPTER IV : SYSTEM TESTING

7.1 Introduction

This chapter describes whether the system developed meets the functional requirements and specification predetermined for the system.

7.2 Functional Testing of Each Functional Module

7.2.1 Database Testing

The functions that should be done by the database are:

1. Send parameters to ANN for learning
2. Send result matrix to result.mat

Send parameters to ANN for learning

Before the ANN learning take place, the database should send the data needed by the ANN as the parameter. Table 7.2 shows whether the data sent to the ANN are the same as the data that stored in the database.

Table 7.1 : Comparison between data stored in database and data received by the ANN

Route	Data stored in database	Data received by the ANN	Status
1	[826 -1 45.4 -1; 0 -1 11.8 -1]	826.0000 -1.0000 45.4000 -1.0000 0 -1.0000 11.8000 -1.0000	OK
2	[609 269 33.5 -1; 0 0.75 6.8 -1]	609.0000 269.0000 33.5000 -1.0000 0 0.7500 6.8000 -1.0000	OK
3	[382 158 22.7 590; 0 0.75 4.2 8]	382.0000 158.0000 22.7000 590.0000 0 0.7500 4.2000 8.0000	OK
4	[747 270 41 1070; 0 1 8.3 16]	1.0e+003 * 0.7470 0.2700 0.0410 1.0700 0 0.0010 0.0083 0.0160	OK
5	[219 -1 12 -1; 0 -1 2.5 -1]	219.0000 -1.0000 12.0000 -1.0000 0 -1.0000 2.5000 -1.0000	OK
6	[491 158 25 -1; 0 45 7 -1]	491 158 25 -1 0 45 7 -1	OK
7	[540 226 29.7 -1; 0 0.8 7.8 -1]	540.0000 226.0000 29.7000 -1.0000 0 0.8000 7.8000 -1.0000	OK
8	[274 112 14.2 -1; 0 40 3 -1]	274.0000 112.0000 14.2000 -1.0000 0 40.0000 3.0000 -1.0000	OK
9	[320 -1 17.6 -1; 0 -1 3.6 -1]	320.0000 -1.0000 17.6000 -1.0000 0 -1.0000 3.6000 -1.0000	OK
10	[365 141 20.2 590; 0 45 4 36]	365.0000 141.0000 20.2000 590.0000 0 45.0000 4.0000 36.0000	OK

The result shows that the data stored in database and the data received by the ANN is identical. Even there are some differences in the numbers of decimal points; they do not affect the value of the data.

Send result matrix to result.mat

The next test is to identify whether the module could send the result generated by the ANN in matrix form to result.mat file, the file that keeps the ANN generated solutions.

Table 7.2 : Comparison between generated by the ANN and data stored in result.mat file

Result generated by the ANN	Data stored in result.mat file	Status
[5 6 6 6 5 7 6 6 7 6]	[5 6 6 6 5 7 6 6 7 6]	Success

Table 7.2 shows the data stored in result.mat file are as the same as the data generated by the ANN.

From the tests that have been done, they show that the database module successfully meet the requirement.

7.2.2 ANN Module Testing

The function that should be done by the ANN module is to recognize the parameters given and simulates them to generate the best solution for each route.

Table 7.3 shows the results of the overall ANN testing to identify any defect during the ANN learning process.

From the results, it clearly shows that the ANN is able to generate routes as the solution. The discussion whether the solution it simulates can be determined as the best solution or not will be discussed when all of the tests are done.

Table 7.3 : ANN Module Testing Result

Route	Result	Status
Penang <-> K Terengganu	5	Success
Penang <-> Kuantan	6	Success
Penang <-> K Lumpur	6	Success
Penang <-> Johor Bahru	6	Success
K Terengganu <-> Kuantan	5	Success
K Terengganu <-> K Lumpur	6	Success
K Terengganu <-> J Bahru	6	Success
Kuantan <-> K Lumpur	6	Success
Kuantan <-> J Bahru	7	Success
K Lumpur <-> J Bahru	6	Success

7.2.3 GUI Testing

The GUI functions to interact with the users using the understandable representations of text. It receives the preferred departure and destination cities from the users. Then, it converts the related numeric value of solutions generated by the ANN to texts.

Table 7.4 shows the GUI test to check whether it could convert the numeric value from the ANN to a text form that understandable by users. The result shows that the GUI manages to do the task excellently without any fault.

Table 7.4: GUI Testing For Numeric Representation of Solution

Route	Numeric (From ANN)	Text (Converted by GUI)	Status
Penang <-> K Terengganu	5	By Road	Success
Penang <-> Kuantan	6	By Air	Success
Penang <-> K Lumpur	6	By Air	Success
Penang <-> Johor Bahru	6	By Air	Success
K Terengganu <-> Kuantan	5	By Road	Success
K Terengganu <-> K Lumpur	6	By Air	Success
K Terengganu <-> J Bahru	6	By Air	Success
Kuantan <-> K Lumpur	6	By Air	Success
Kuantan <-> J Bahru	7	By Sea	Success
K Lumpur <-> J Bahru	6	By Air	Success

7.3 Summary

From a series of test that had been done to the system, it proved that the system had meet the requirements defined. Even it is hard to evaluate whether the results it generate are the best route or not, as there are many factors to put into account for (e.g. limited time, money constraints), we noticed that the results it generates are reasonable.

For an example, the Penang to Terengganu or vice versa route. The solution `By Road' that it generates is considered the best route since there is no air transportation between the cities.

Chapter VIII: Discussion

8.0 CHAPTER VIII: DISCUSSION

8.1 Introduction

This chapter discusses the gathered results from the tests done in the previous chapter, the problem encountered, the solution taken, the advantages, the weakness, future improvement, ideas and conclusion for the project.

8.2 Results

The tests done in the previous chapter proved that the system meet the functional requirements defined during the system development. All of the modules namely the database, the ANN and the GUI module work successfully.

As a revision the objectives of the system are:

1) To develop a system that advises users on travel routing which implement artificial neural network.

2) To apply the concept of artificial neural network in prediction system to travel route advisor.

3) To predict the most efficient route according to parameters like distance, time consumed and cost.

From the results on the last chapter, it is likely to say that the objectives of the system had been satisfied.

8.3 Problems Encountered and Solutions Taken

Many problems had been encountered during the development of the project. Steps have been taken in solving the problems. The major problems encountered and the solutions taken during the project progress include:

8.3.1 Lack of knowledge on ANN

ANN could be considered as a complex and hard topic. It involves a lot of terms and mathematics calculation. Even some biological approaches have to be learned. A lot of time taken to choose the right ANN type and the development of the ANN module. The references on the topic is also very limited as it is considered a new field.

Solution

The only way to solve the lack of knowledge problem is to do a lot of researches and readings. Books and websites had been read and browsed to make a clear definition on the ANN. As the enrichment, guidance from the supervisor also important.

8.3.2 Inexperience in the programming language used

Persons who have the hands-on experience with MATLAB will consider it as an easy programming language. Differently as a beginner, it seems to be a very hard task. The programming time taken also increased, as there is problems occurred due to the lack of knowledge in the programming language.

Solution

The most effective way to solve this problem is to begin the study of the programming language from the very basic part. At this stage a beginner's handbooks for programming language are very helpful. The documentations and other related program source code also could help to boost the knowledge.

8.4 Further Research

Belows are suggestions on the enhancement of this system that could be done in future.

1. Enlarge the system by increasing the numbers of cities. By widen the scope of the system; we could also learn the behavior of the ANN.
2. Increase other consideration to put into account such the time constraints, budget and the number of persons to go for the travel. The constraints and consideration should come from the user's input.
3. Upgrade the route advisor as a web based system.

8.5 Summary

The thesis has proved to be a very unique experience and will probably on of the most challenging course throughout the degree program. It let students feel the whole development process as well as deal with many of the management issues. It presents it self as a test to practice all of the theoretical skills that have been taught in the other courses.

The students fill many shoes of many people during the whole development process from writing requirements, designing, coding and testing. This will greatly help the student in preparing them to the real world scenarios once in the working arena.

Throughout the development, many problems have arisen that requires solving. Many of these problems may be the first time encountering such problems and provides excellent exposure. Others happened to be problems dealing lack of experience and knowledge.

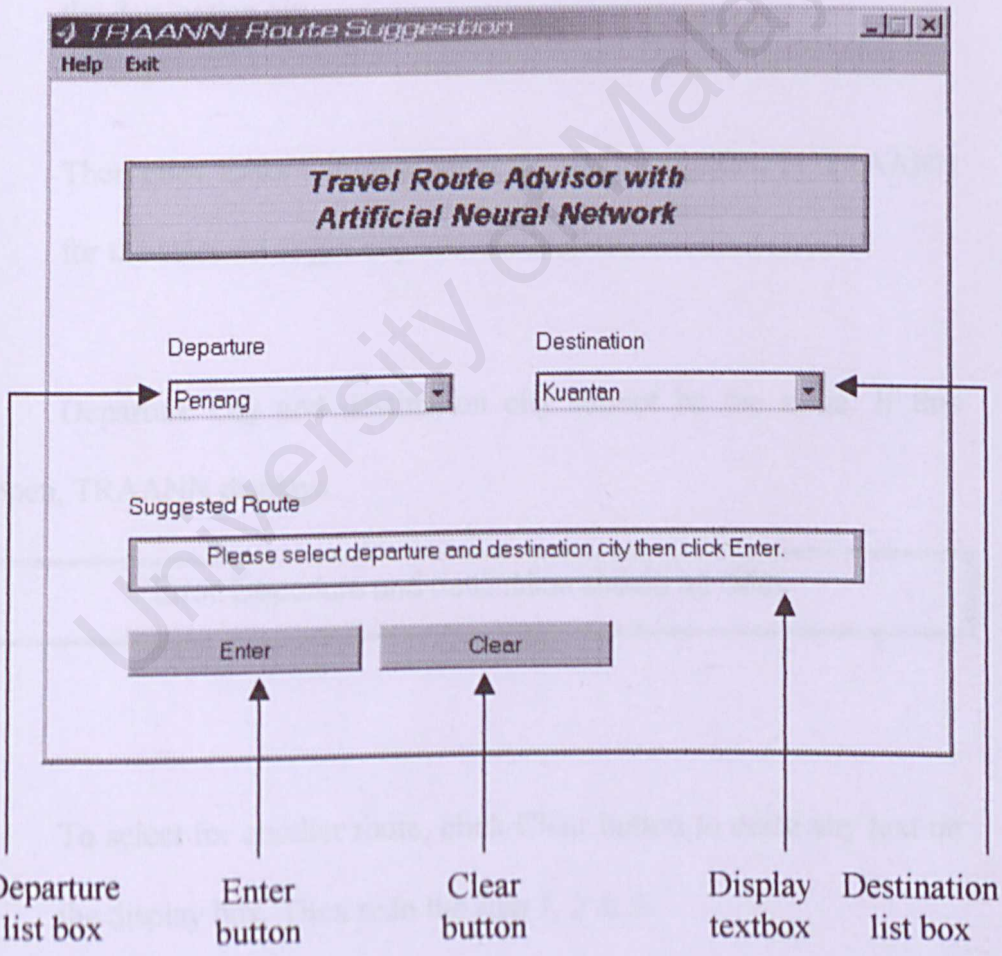
Taken as a whole this course has succeeded in providing a useful training ground for undergraduates to test the mettle with problems that they will face as they go out into the working world.

User Manual of TRAANN

When you started Travel Route Advisor with Artificial Neural Network (TRAANN), you will find a display box. It display

Please select departure and destination city then click Enter.

It means you are ready to start your TRAANN session.



1. Click the arrow on departure list box and the box expands. It displays the available departure cities. Click a city to choose for the origin city.

2. Then click the arrow on destination list box and the box expands. It displays the available departure cities. Click a city to choose for the destination city.

3 Then click Enter button to view the route suggested by TRAANN for the selected origin city and destination.

Departure city and destination city cannot be the same. If this happen, TRAANN displays

Error. Departure and destination should be differ.

4. To select for another route, click Clear button to erase any text on the display box. Then redo the step 1, 2 & 3.

5. To finish the TRAANN session, click Exit menu on the top of the TRAANN windows.

Notes:

- 'Using TRAANN' menu in 'Help' menu display the help on using Travel Route Advisor with Artificial Neural Network.
- 'About' menu in `Help' menu display the version, date and programmer data..

APPENDIX

ann.m coding –ANN Module

```
function result = ann (p)
% create a two-neuron layer with two input elements
net = newc([0 1; 0 1],2)
% initialize the weights to the center of the input ranges
wts = net.IW{1,1}
% train the network for 300 epochs
net.trainParam.epochs = 300
net = train(net,p)
% the training function for competitive networks
net.trainFcn
% supply the original vectors as input to the network,
% simulate the network, and convert its output vectors
% to class indices.
a = sim(net,p)
ac = vec2ind(a)
% return the value for solution
result = sum (ac)
```

db.m coding – Database Module

```
% resetting the result values
R = [0 0 0 0 0 0 0 0 0 0]
save result R

% travelling information
% p = [distance, fare (air, road, sea); 0, time consumed (air, road, sea)]
% & send to ANN for learning

% penang <-> k terengganu
p = [826 -1 45.4 -1; 0 -1 11.8 -1]
R(1) = ann(p)

% penang <-> kuantan
p = [609 269 33.5 -1; 0 0.75 6.8 -1]
R(2) = ann(p)

% penang <-> k lumpur
p = [382 158 22.7 590; 0 0.75 4.2 8]
R(3) = ann(p)

% penang <-> j bahru
p = [747 270 41 1070; 0 1 8.3 16]
R(4) = ann(p)

% k terengganu <-> kuantan
p = [219 -1 12 -1; 0 -1 2.5 -1]
R(5) = ann(p)

% k terengganu <-> k lumpur
p = [491 158 25 -1; 0 45 7 -1]
R(6) = ann(p)

% k terengganu <-> j bahru
p = [540 226 29.7 -1; 0 0.8 7.8 -1]
R(7) = ann(p)

% kuantan <-> k lumpur
p = [274 112 14.2 -1; 0 40 3 -1]
R(8) = ann(p)
```

```
% kuantan <-> j bahru
p = [320 -1 17.6 -1; 0 -1 3.6 -1]
R(9) = ann(p)

% k lumpur <-> j bahru
p = [365 141 20.2 590; 0 45 4 36]
R(10) = ann(p)

% send result matrix to result.m
save result R
```

Penang

by air		by road		by sea	
fare					
	Penang		Penang		Penang
K Terengganu		K Terengganu	45.4	K Terengganu	
Kuantan	269	Kuantan	33.5	Kuantan	
KL	158	KL	22.7	KL	590
JB	270	JB	41	JB	1070
time					
	Penang		Penang		Penang
K Terengganu		K Terengganu	11.8	K Terengganu	
Kuantan		Kuantan	6.8	Kuantan	
KL	45	KL	4.2	KL	
JB		JB	8.3	JB	
distance					
					Penang
K Terengganu					826
Kuantan					609
KL					382
JB					747

K Terengganu

by air		by road		by sea	
fare					
	K Terengganu		KL		KL
Penang		Penang	45.4	Penang	
Kuantan		Kuantan	12	Kuantan	
KL	158	KL	25	KL	
JB	226	JB	29.7	JB	
time					
	K Terengganu		K Terengganu		K Terengganu
Penang		Penang	11.8	Penang	
Kuantan		Kuantan	2.5	Kuantan	
KL	45	KL	7	KL	
JB		JB	7.8	JB	

distance	
	K Terengganu
Penang	826
Kuantan	219
KL	491
JB	540

Kuantan

by air		by road		by sea	
fare					
	Kuantan		Kuantan		Kuantan
Penang	269	Penang	33.5	Penang	
K Terengganu		K Terengganu	12	K Terengganu	
KL	112	KL	14.2	KL	
JB	141	JB	17.6	JB	
time					
	Kuantan		Kuantan		Kuantan
Penang		Penang	6.8	Penang	
K Terengganu		K Terengganu	2.5	K Terengganu	
KL	40	KL	3	KL	
JB		JB	3.6	JB	
distance					
					Kuantan
				Penang	609
				K Terengganu	219
				KL	274
				JB	320

KL

by air		by road		by sea	
fare					
	KL		KL		KL
Penang	158	Penang	22.7	Penang	590
K Terengganu	158	K Terengganu	25	K Terengganu	
Kuantan	112	Kuantan	14.2	Kuantan	
JB	141	JB	20.2	JB	890
time					
	KL		KL		KL
Penang	45	Penang	4.2	Penang	24
K Terengganu	45	K Terengganu	7	K Terengganu	
Kuantan	40	Kuantan	3	Kuantan	
JB	45	JB	4	JB	36
distance					
					KL
				Penang	382
				K Terengganu	491
				Kuantan	274
				JB	365

JB

by air		by road		by sea	
fare					
	JB		JB		JB
Penang	270	Penang	41	Penang	1070
K Terengganu	226	K Terengganu	29.7	K Terengganu	
Kuantan	141	Kuantan	17.6	Kuantan	
KL	141	KL	20.2	KL	890
time					
	JB		JB		JB
Penang		Penang	8.3	Penang	24
K Terengganu		K Terengganu	7.8	K Terengganu	
Kuantan		Kuantan	3.6	Kuantan	
KL	45	KL	4	KL	36
distance					
					JB
Penang					747
K Terengganu					540
Kuantan					320
KL					365

Full Coding Track of ANN learning module

```
p =

  826.0000   -1.0000   45.4000   -1.0000
         0   -1.0000   11.8000   -1.0000


net =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

      numOutputs: 1  (read-only)
      numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

          inputs: {1x1 cell} of inputs
          layers: {1x1 cell} of layers
         outputs: {1x1 cell} containing 1 output
         targets: {1x1 cell} containing no targets
          biases: {1x1 cell} containing 1 bias
     inputWeights: {1x1 cell} containing 1 input weight
     layerWeights: {1x1 cell} containing no layer weights

    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
         trainFcn: 'trainr'

    parameters:

      adaptParam: .passes
       initParam: (none)
    performParam: (none)
```

trainParam: .epochs, .goal, .show, .time

weight and bias values:

IW: {1x1 cell} containing 1 input weight matrix
LW: {1x1 cell} containing no layer weight matrices
b: {1x1 cell} containing 1 bias vector

other:

userdata: (user stuff)

wts =

0.5000 0.5000
0.5000 0.5000

net =

Neural Network object:

architecture:

numInputs: 1
numLayers: 1
biasConnect: [1]
inputConnect: [1]
layerConnect: [0]
outputConnect: [1]
targetConnect: [0]

numOutputs: 1 (read-only)
numTargets: 0 (read-only)
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)

subobject structures:

inputs: {1x1 cell} of inputs
layers: {1x1 cell} of layers
outputs: {1x1 cell} containing 1 output
targets: {1x1 cell} containing no targets
biases: {1x1 cell} containing 1 bias
inputWeights: {1x1 cell} containing 1 input weight
layerWeights: {1x1 cell} containing no layer weights

functions:

```
    adaptFcn: 'trains'
     initFcn: 'initlay'
   performFcn: (none)
     trainFcn: 'trainr'

parameters:

   adaptParam: .passes
    initParam: (none)
  performParam: (none)
    trainParam: .epochs, .goal, .show, .time

weight and bias values:

        IW: {1x1 cell} containing 1 input weight matrix
        LW: {1x1 cell} containing no layer weight matrices
         b: {1x1 cell} containing 1 bias vector

other:

    userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
TRAINR, Epoch 225/300
TRAINR, Epoch 250/300
TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.


net =

  Neural Network object:

  architecture:

    numInputs: 1
    numLayers: 1
  biasConnect: [1]
 inputConnect: [1]
 layerConnect: [0]
outputConnect: [1]
```

```
    targetConnect: [0]

      numOutputs: 1  (read-only)
      numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

          inputs: {1x1 cell} of inputs
          layers: {1x1 cell} of layers
         outputs: {1x1 cell} containing 1 output
         targets: {1x1 cell} containing no targets
          biases: {1x1 cell} containing 1 bias
     inputWeights: {1x1 cell} containing 1 input weight
     layerWeights: {1x1 cell} containing no layer weights

    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
         trainFcn: 'trainr'

    parameters:

       adaptParam: .passes
        initParam: (none)
     performParam: (none)
       trainParam: .epochs, .goal, .show, .time

    weight and bias values:

            IW: {1x1 cell} containing 1 input weight matrix
            LW: {1x1 cell} containing no layer weight matrices
             b: {1x1 cell} containing 1 bias vector

    other:

        userdata: (user stuff)

ans =

trainr


a =

  (1,1)        1
```

```
(2,2)        1
(2,3)        1
(2,4)        1

ac =

    1    1    2    1

result =

    5

R =

    5    0    0    0    0    0    0    0    0    0

p =

  609.0000  269.0000   33.5000   -1.0000
         0    0.7500    6.8000   -1.0000

net =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

      numOutputs: 1  (read-only)
      numTargets: 0  (read-only)
  numInputDelays: 0  (read-only)
  numLayerDelays: 0  (read-only)

    subobject structures:

          inputs: {1x1 cell} of inputs
          layers: {1x1 cell} of layers
         outputs: {1x1 cell} containing 1 output
```

```
        targets: {1x1 cell} containing no targets
         biases: {1x1 cell} containing 1 bias
   inputWeights: {1x1 cell} containing 1 input weight
   layerWeights: {1x1 cell} containing no layer weights

  functions:

      adaptFcn: 'trains'
       initFcn: 'initlay'
     performFcn: (none)
       trainFcn: 'trainr'

  parameters:

     adaptParam: .passes
      initParam: (none)
    performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

           IW: {1x1 cell} containing 1 input weight matrix
           LW: {1x1 cell} containing no layer weight matrices
            b: {1x1 cell} containing 1 bias vector

  other:

      userdata: (user stuff)


wts =

  0.5000   0.5000
  0.5000   0.5000


net =

  Neural Network object:

  architecture:

     numInputs: 1
     numLayers: 1
   biasConnect: [1]
  inputConnect: [1]
  layerConnect: [0]
 outputConnect: [1]
 targetConnect: [0]
```

```
    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
  numInputDelays: 0  (read-only)
  numLayerDelays: 0  (read-only)

  subobject structures:

         inputs: {1x1 cell} of inputs
         layers: {1x1 cell} of layers
        outputs: {1x1 cell} containing 1 output
        targets: {1x1 cell} containing no targets
         biases: {1x1 cell} containing 1 bias
   inputWeights: {1x1 cell} containing 1 input weight
   layerWeights: {1x1 cell} containing no layer weights

  functions:

      adaptFcn: 'trains'
       initFcn: 'initlay'
     performFcn: (none)
      trainFcn: 'trainr'

  parameters:

     adaptParam: .passes
      initParam: (none)
   performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

           IW: {1x1 cell} containing 1 input weight matrix
           LW: {1x1 cell} containing no layer weight matrices
            b: {1x1 cell} containing 1 bias vector

  other:

      userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
TRAINR, Epoch 225/300
TRAINR, Epoch 250/300
```

TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.

net =

Neural Network object:

architecture:

numInputs: 1
numLayers: 1
biasConnect: [1]
inputConnect: [1]
layerConnect: [0]
outputConnect: [1]
targetConnect: [0]

numOutputs: 1 (read-only)
numTargets: 0 (read-only)
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)

subobject structures:

inputs: {1x1 cell} of inputs
layers: {1x1 cell} of layers
outputs: {1x1 cell} containing 1 output
targets: {1x1 cell} containing no targets
biases: {1x1 cell} containing 1 bias
inputWeights: {1x1 cell} containing 1 input weight
layerWeights: {1x1 cell} containing no layer weights

functions:

adaptFcn: 'trains'
initFcn: 'initlay'
performFcn: (none)
trainFcn: 'trainr'

parameters:

adaptParam: .passes
initParam: (none)
performParam: (none)
trainParam: .epochs, .goal, .show, .time

weight and bias values:

```
        IW: {1x1 cell} containing 1 input weight matrix
        LW: {1x1 cell} containing no layer weight matrices
         b: {1x1 cell} containing 1 bias vector

    other:

        userdata: (user stuff)

ans =

trainr


a =

    (1,1)      1
    (1,2)      1
    (2,3)      1
    (2,4)      1


ac =

     1     1     2     2


result =

     6


R =

     5     6     0     0     0     0     0     0     0     0


p =

  382.0000  158.0000   22.7000  590.0000
         0    0.7500    4.2000    8.0000


net =

    Neural Network object:

    architecture:

        numInputs: 1
```

```
    numLayers: 1
  biasConnect: [1]
 inputConnect: [1]
 layerConnect: [0]
outputConnect: [1]
targetConnect: [0]

    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
numInputDelays: 0  (read-only)
numLayerDelays: 0  (read-only)

subobject structures:

       inputs: {1x1 cell} of inputs
       layers: {1x1 cell} of layers
      outputs: {1x1 cell} containing 1 output
      targets: {1x1 cell} containing no targets
       biases: {1x1 cell} containing 1 bias
 inputWeights: {1x1 cell} containing 1 input weight
 layerWeights: {1x1 cell} containing no layer weights

functions:

    adaptFcn: 'trains'
     initFcn: 'initlay'
   performFcn: (none)
    trainFcn: 'trainr'

parameters:

   adaptParam: .passes
    initParam: (none)
 performParam: (none)
   trainParam: .epochs, .goal, .show, .time

weight and bias values:

        IW: {1x1 cell} containing 1 input weight matrix
        LW: {1x1 cell} containing no layer weight matrices
         b: {1x1 cell} containing 1 bias vector

other:

    userdata: (user stuff)


wts =

  0.5000    0.5000
```

```
    0.5000    0.5000

net =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

        numOutputs: 1  (read-only)
        numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

            inputs: {1x1 cell} of inputs
            layers: {1x1 cell} of layers
           outputs: {1x1 cell} containing 1 output
           targets: {1x1 cell} containing no targets
            biases: {1x1 cell} containing 1 bias
     inputWeights: {1x1 cell} containing 1 input weight
     layerWeights: {1x1 cell} containing no layer weights

    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
         trainFcn: 'trainr'

    parameters:

       adaptParam: .passes
        initParam: (none)
     performParam: (none)
      trainParam: .epochs, .goal, .show, .time

    weight and bias values:

            IW: {1x1 cell} containing 1 input weight matrix
            LW: {1x1 cell} containing no layer weight matrices
```

```
        b: {1x1 cell} containing 1 bias vector

  other:

      userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
TRAINR, Epoch 225/300
TRAINR, Epoch 250/300
TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.


net =

  Neural Network object:

  architecture:

     numInputs: 1
     numLayers: 1
   biasConnect: [1]
  inputConnect: [1]
  layerConnect: [0]
 outputConnect: [1]
 targetConnect: [0]

    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
 numInputDelays: 0  (read-only)
 numLayerDelays: 0  (read-only)

  subobject structures:

       inputs: {1x1 cell} of inputs
       layers: {1x1 cell} of layers
      outputs: {1x1 cell} containing 1 output
      targets: {1x1 cell} containing no targets
       biases: {1x1 cell} containing 1 bias
  inputWeights: {1x1 cell} containing 1 input weight
  layerWeights: {1x1 cell} containing no layer weights
```

functions:

adaptFcn: 'trains'
initFcn: 'initlay'
performFcn: (none)
trainFcn: 'trainr'

parameters:

adaptParam: .passes
initParam: (none)
performParam: (none)
trainParam: .epochs, .goal, .show, .time

weight and bias values:

IW: {1x1 cell} containing 1 input weight matrix
LW: {1x1 cell} containing no layer weight matrices
b: {1x1 cell} containing 1 bias vector

other:

userdata: (user stuff)

ans =

trainr

a =

(1,1) 1
(2,2) 1
(2,3) 1
(1,4) 1

ac =

1 2 2 1

result =

6

R =

```
    5    6    6    0    0    0    0    0    0    0

p =

  1.0e+003 *

    0.7470    0.2700    0.0410    1.0700
         0    0.0010    0.0083    0.0160


net =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

       numOutputs: 1  (read-only)
       numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

           inputs: {1x1 cell} of inputs
           layers: {1x1 cell} of layers
          outputs: {1x1 cell} containing 1 output
          targets: {1x1 cell} containing no targets
           biases: {1x1 cell} containing 1 bias
     inputWeights: {1x1 cell} containing 1 input weight
     layerWeights: {1x1 cell} containing no layer weights

    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
         trainFcn: 'trainr'

    parameters:
```

```
    adaptParam: .passes
     initParam: (none)
   performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

          IW: {1x1 cell} containing 1 input weight matrix
          LW: {1x1 cell} containing no layer weight matrices
           b: {1x1 cell} containing 1 bias vector

  other:

     userdata: (user stuff)


wts =

  0.5000    0.5000
  0.5000    0.5000


net =

  Neural Network object:

  architecture:

     numInputs: 1
     numLayers: 1
   biasConnect: [1]
  inputConnect: [1]
  layerConnect: [0]
 outputConnect: [1]
 targetConnect: [0]

    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
 numInputDelays: 0  (read-only)
 numLayerDelays: 0  (read-only)

  subobject structures:

      inputs: {1x1 cell} of inputs
      layers: {1x1 cell} of layers
     outputs: {1x1 cell} containing 1 output
     targets: {1x1 cell} containing no targets
      biases: {1x1 cell} containing 1 bias
  inputWeights: {1x1 cell} containing 1 input weight
  layerWeights: {1x1 cell} containing no layer weights
```

```
    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
         trainFcn: 'trainr'

    parameters:

        adaptParam: .passes
         initParam: (none)
       performParam: (none)
        trainParam: .epochs, .goal, .show, .time

    weight and bias values:

                IW: {1x1 cell} containing 1 input weight matrix
                LW: {1x1 cell} containing no layer weight matrices
                 b: {1x1 cell} containing 1 bias vector

    other:

            userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
TRAINR, Epoch 225/300
TRAINR, Epoch 250/300
TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.


net =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
```

```
    inputConnect: [1]
    layerConnect: [0]
   outputConnect: [1]
   targetConnect: [0]

     numOutputs: 1  (read-only)
     numTargets: 0  (read-only)
  numInputDelays: 0  (read-only)
  numLayerDelays: 0  (read-only)

  subobject structures:

         inputs: {1x1 cell} of inputs
         layers: {1x1 cell} of layers
        outputs: {1x1 cell} containing 1 output
        targets: {1x1 cell} containing no targets
         biases: {1x1 cell} containing 1 bias
   inputWeights: {1x1 cell} containing 1 input weight
   layerWeights: {1x1 cell} containing no layer weights

  functions:

       adaptFcn: 'trains'
        initFcn: 'initlay'
      performFcn: (none)
       trainFcn: 'trainr'

  parameters:

     adaptParam: .passes
      initParam: (none)
    performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

           IW: {1x1 cell} containing 1 input weight matrix
           LW: {1x1 cell} containing no layer weight matrices
            b: {1x1 cell} containing 1 bias vector

  other:

       userdata: (user stuff)

ans =

trainr
```

```
a =

  (1,1)       1
  (2,2)       1
  (2,3)       1
  (1,4)       1


ac =

    1    2    2    1


result =

    6


R =

    5    6    6    6    0    0    0    0    0    0
p =

  219.0000   -1.0000   12.0000   -1.0000
         0   -1.0000    2.5000   -1.0000


net =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

      numOutputs: 1  (read-only)
      numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

          inputs: {1x1 cell} of inputs
          layers: {1x1 cell} of layers
```

```
        outputs: {1x1 cell} containing 1 output
        targets: {1x1 cell} containing no targets
         biases: {1x1 cell} containing 1 bias
   inputWeights: {1x1 cell} containing 1 input weight
   layerWeights: {1x1 cell} containing no layer weights

  functions:

      adaptFcn: 'trains'
       initFcn: 'initlay'
     performFcn: (none)
      trainFcn: 'trainr'

  parameters:

     adaptParam: .passes
      initParam: (none)
   performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

           IW: {1x1 cell} containing 1 input weight matrix
           LW: {1x1 cell} containing no layer weight matrices
            b: {1x1 cell} containing 1 bias vector

  other:

      userdata: (user stuff)


wts =

  0.5000   0.5000
  0.5000   0.5000


net =

  Neural Network object:

  architecture:

     numInputs: 1
     numLayers: 1
   biasConnect: [1]
  inputConnect: [1]
  layerConnect: [0]
 outputConnect: [1]
 targetConnect: [0]
```

```
    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
  numInputDelays: 0  (read-only)
  numLayerDelays: 0  (read-only)

  subobject structures:

         inputs: {1x1 cell} of inputs
         layers: {1x1 cell} of layers
        outputs: {1x1 cell} containing 1 output
        targets: {1x1 cell} containing no targets
         biases: {1x1 cell} containing 1 bias
   inputWeights: {1x1 cell} containing 1 input weight
   layerWeights: {1x1 cell} containing no layer weights

  functions:

      adaptFcn: 'trains'
       initFcn: 'initlay'
     performFcn: (none)
      trainFcn: 'trainr'

  parameters:

     adaptParam: .passes
      initParam: (none)
   performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

           IW: {1x1 cell} containing 1 input weight matrix
           LW: {1x1 cell} containing no layer weight matrices
            b: {1x1 cell} containing 1 bias vector

  other:

       userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
TRAINR, Epoch 225/300
```

TRAINR, Epoch 250/300
TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.

net =

Neural Network object:

architecture:

numInputs: 1
numLayers: 1
biasConnect: [1]
inputConnect: [1]
layerConnect: [0]
outputConnect: [1]
targetConnect: [0]

numOutputs: 1 (read-only)
numTargets: 0 (read-only)
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)

subobject structures:

inputs: {1x1 cell} of inputs
layers: {1x1 cell} of layers
outputs: {1x1 cell} containing 1 output
targets: {1x1 cell} containing no targets
biases: {1x1 cell} containing 1 bias
inputWeights: {1x1 cell} containing 1 input weight
layerWeights: {1x1 cell} containing no layer weights

functions:

adaptFcn: 'trains'
initFcn: 'initlay'
performFcn: (none)
trainFcn: 'trainr'

parameters:

adaptParam: .passes
initParam: (none)
performParam: (none)
trainParam: .epochs, .goal, .show, .time

weight and bias values:

```
        IW: {1x1 cell} containing 1 input weight matrix
        LW: {1x1 cell} containing no layer weight matrices
         b: {1x1 cell} containing 1 bias vector

   other:

      userdata: (user stuff)

ans =

trainr


a =

   (2,1)      1
   (1,2)      1
   (1,3)      1
   (1,4)      1


ac =

     2     1     1     1


result =

     5


R =

     5     6     6     6     5     0     0     0     0     0


p =

   491   158    25    -1
     0    45     7    -1


net =

   Neural Network object:

   architecture:
```

```
    numInputs: 1
    numLayers: 1
  biasConnect: [1]
 inputConnect: [1]
 layerConnect: [0]
outputConnect: [1]
targetConnect: [0]

    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
numInputDelays: 0  (read-only)
numLayerDelays: 0  (read-only)

subobject structures:

        inputs: {1x1 cell} of inputs
        layers: {1x1 cell} of layers
       outputs: {1x1 cell} containing 1 output
       targets: {1x1 cell} containing no targets
        biases: {1x1 cell} containing 1 bias
  inputWeights: {1x1 cell} containing 1 input weight
  layerWeights: {1x1 cell} containing no layer weights

functions:

      adaptFcn: 'trains'
       initFcn: 'initlay'
    performFcn: (none)
      trainFcn: 'trainr'

parameters:

    adaptParam: .passes
     initParam: (none)
  performParam: (none)
    trainParam: .epochs, .goal, .show, .time

weight and bias values:

           IW: {1x1 cell} containing 1 input weight matrix
           LW: {1x1 cell} containing no layer weight matrices
            b: {1x1 cell} containing 1 bias vector

other:

      userdata: (user stuff)

wts =
```

```
    0.5000    0.5000
    0.5000    0.5000


net =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

       numOutputs: 1  (read-only)
       numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

           inputs: {1x1 cell} of inputs
           layers: {1x1 cell} of layers
          outputs: {1x1 cell} containing 1 output
          targets: {1x1 cell} containing no targets
           biases: {1x1 cell} containing 1 bias
     inputWeights: {1x1 cell} containing 1 input weight
     layerWeights: {1x1 cell} containing no layer weights

    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
         trainFcn: 'trainr'

    parameters:

       adaptParam: .passes
        initParam: (none)
     performParam: (none)
      trainParam: .epochs, .goal, .show, .time

    weight and bias values:

              IW: {1x1 cell} containing 1 input weight matrix
```

```
        LW: {1x1 cell} containing no layer weight matrices
         b: {1x1 cell} containing 1 bias vector

  other:

      userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
TRAINR, Epoch 225/300
TRAINR, Epoch 250/300
TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.


net =

  Neural Network object:

  architecture:

     numInputs: 1
     numLayers: 1
   biasConnect: [1]
  inputConnect: [1]
  layerConnect: [0]
 outputConnect: [1]
 targetConnect: [0]

    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
 numInputDelays: 0  (read-only)
 numLayerDelays: 0  (read-only)

  subobject structures:

       inputs: {1x1 cell} of inputs
       layers: {1x1 cell} of layers
      outputs: {1x1 cell} containing 1 output
      targets: {1x1 cell} containing no targets
       biases: {1x1 cell} containing 1 bias
 inputWeights: {1x1 cell} containing 1 input weight
```

```
    layerWeights: {1x1 cell} containing no layer weights

    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
         trainFcn: 'trainr'

    parameters:

        adaptParam: .passes
         initParam: (none)
       performParam: (none)
         trainParam: .epochs, .goal, .show, .time

    weight and bias values:

              IW: {1x1 cell} containing 1 input weight matrix
              LW: {1x1 cell} containing no layer weight matrices
               b: {1x1 cell} containing 1 bias vector

    other:

        userdata: (user stuff)


ans =

trainr


a =

  (1,1)        1
  (2,2)        1
  (2,3)        1
  (2,4)        1


ac =

     1     2     2     2


result =

     7
```

R =

```
    5    6    6    6    5    7    0    0    0    0
```

p =

```
  540.0000  226.0000   29.7000   -1.0000
         0    0.8000    7.8000   -1.0000
```

net =

```
    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

      numOutputs: 1  (read-only)
      numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

          inputs: {1x1 cell} of inputs
          layers: {1x1 cell} of layers
         outputs: {1x1 cell} containing 1 output
         targets: {1x1 cell} containing no targets
          biases: {1x1 cell} containing 1 bias
     inputWeights: {1x1 cell} containing 1 input weight
     layerWeights: {1x1 cell} containing no layer weights

    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
         trainFcn: 'trainr'

    parameters:

      adaptParam: .passes
```

```
    initParam: (none)
  performParam: (none)
    trainParam: .epochs, .goal, .show, .time

  weight and bias values:

          IW: {1x1 cell} containing 1 input weight matrix
          LW: {1x1 cell} containing no layer weight matrices
           b: {1x1 cell} containing 1 bias vector

  other:

     userdata: (user stuff)


wts =

  0.5000   0.5000
  0.5000   0.5000


net =

  Neural Network object:

  architecture:

    numInputs: 1
    numLayers: 1
  biasConnect: [1]
 inputConnect: [1]
 layerConnect: [0]
outputConnect: [1]
targetConnect: [0]

    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
numInputDelays: 0  (read-only)
numLayerDelays: 0  (read-only)

  subobject structures:

      inputs: {1x1 cell} of inputs
      layers: {1x1 cell} of layers
     outputs: {1x1 cell} containing 1 output
     targets: {1x1 cell} containing no targets
      biases: {1x1 cell} containing 1 bias
 inputWeights: {1x1 cell} containing 1 input weight
 layerWeights: {1x1 cell} containing no layer weights
```

functions:

adaptFcn: 'trains'
initFcn: 'initlay'
performFcn: (none)
trainFcn: 'trainr'

parameters:

adaptParam: .passes
initParam: (none)
performParam: (none)
trainParam: .epochs, .goal, .show, .time

weight and bias values:

IW: {1x1 cell} containing 1 input weight matrix
LW: {1x1 cell} containing no layer weight matrices
b: {1x1 cell} containing 1 bias vector

other:

userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
TRAINR, Epoch 225/300
TRAINR, Epoch 250/300
TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.

net =

Neural Network object:

architecture:

numInputs: 1
numLayers: 1
biasConnect: [1]
inputConnect: [1]

```
    layerConnect: [0]
   outputConnect: [1]
   targetConnect: [0]

     numOutputs: 1  (read-only)
     numTargets: 0  (read-only)
  numInputDelays: 0  (read-only)
  numLayerDelays: 0  (read-only)

  subobject structures:

          inputs: {1x1 cell} of inputs
          layers: {1x1 cell} of layers
         outputs: {1x1 cell} containing 1 output
         targets: {1x1 cell} containing no targets
          biases: {1x1 cell} containing 1 bias
    inputWeights: {1x1 cell} containing 1 input weight
    layerWeights: {1x1 cell} containing no layer weights

  functions:

       adaptFcn: 'trains'
        initFcn: 'initlay'
      performFcn: (none)
       trainFcn: 'trainr'

  parameters:

     adaptParam: .passes
      initParam: (none)
    performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

            IW: {1x1 cell} containing 1 input weight matrix
            LW: {1x1 cell} containing no layer weight matrices
             b: {1x1 cell} containing 1 bias vector

  other:

      userdata: (user stuff)

ans =

trainr


a =
```

```
(2,1)     1
(2,2)     1
(1,3)     1
(1,4)     1

ac =

    2    2    1    1

result =

    6

R =

    5    6    6    6    5    7    6    0    0    0

p =

  274.0000  112.0000   14.2000   -1.0000
         0   40.0000    3.0000   -1.0000

net =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

        numOutputs: 1  (read-only)
        numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

          inputs: {1x1 cell} of inputs
```

```
        layers: {1x1 cell} of layers
       outputs: {1x1 cell} containing 1 output
       targets: {1x1 cell} containing no targets
        biases: {1x1 cell} containing 1 bias
  inputWeights: {1x1 cell} containing 1 input weight
  layerWeights: {1x1 cell} containing no layer weights

  functions:

      adaptFcn: 'trains'
       initFcn: 'initlay'
    performFcn: (none)
      trainFcn: 'trainr'

  parameters:

    adaptParam: .passes
     initParam: (none)
  performParam: (none)
    trainParam: .epochs, .goal, .show, .time

  weight and bias values:

          IW: {1x1 cell} containing 1 input weight matrix
          LW: {1x1 cell} containing no layer weight matrices
           b: {1x1 cell} containing 1 bias vector

  other:

     userdata: (user stuff)


wts =

  0.5000    0.5000
  0.5000    0.5000


net =

  Neural Network object:

  architecture:

    numInputs: 1
    numLayers: 1
  biasConnect: [1]
  inputConnect: [1]
  layerConnect: [0]
  outputConnect: [1]
```

```
 targetConnect: [0]

    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
  numInputDelays: 0  (read-only)
  numLayerDelays: 0  (read-only)

  subobject structures:

         inputs: {1x1 cell} of inputs
         layers: {1x1 cell} of layers
        outputs: {1x1 cell} containing 1 output
        targets: {1x1 cell} containing no targets
         biases: {1x1 cell} containing 1 bias
   inputWeights: {1x1 cell} containing 1 input weight
   layerWeights: {1x1 cell} containing no layer weights

  functions:

      adaptFcn: 'trains'
       initFcn: 'initlay'
     performFcn: (none)
       trainFcn: 'trainr'

  parameters:

     adaptParam: .passes
      initParam: (none)
    performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

           IW: {1x1 cell} containing 1 input weight matrix
           LW: {1x1 cell} containing no layer weight matrices
            b: {1x1 cell} containing 1 bias vector

  other:

       userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
```

TRAINR, Epoch 225/300
TRAINR, Epoch 250/300
TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.

net =

Neural Network object:

architecture:

numInputs: 1
numLayers: 1
biasConnect: [1]
inputConnect: [1]
layerConnect: [0]
outputConnect: [1]
targetConnect: [0]

numOutputs: 1 (read-only)
numTargets: 0 (read-only)
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)

subobject structures:

inputs: {1x1 cell} of inputs
layers: {1x1 cell} of layers
outputs: {1x1 cell} containing 1 output
targets: {1x1 cell} containing no targets
biases: {1x1 cell} containing 1 bias
inputWeights: {1x1 cell} containing 1 input weight
layerWeights: {1x1 cell} containing no layer weights

functions:

adaptFcn: 'trains'
initFcn: 'initlay'
performFcn: (none)
trainFcn: 'trainr'

parameters:

adaptParam: .passes
initParam: (none)
performParam: (none)
trainParam: .epochs, .goal, .show, .time

weight and bias values:

IW: {1x1 cell} containing 1 input weight matrix
LW: {1x1 cell} containing no layer weight matrices
b: {1x1 cell} containing 1 bias vector

other:

userdata: (user stuff)

ans =

trainr

a =

(1,1) 1
(1,2) 1
(2,3) 1
(2,4) 1

ac =

1 1 2 2

result =

6

R =

5 6 6 6 5 7 6 6 0 0

p =

320.0000 -1.0000 17.6000 -1.0000
0 -1.0000 3.6000 -1.0000

net =

Neural Network object:

architecture:

```
    numInputs: 1
    numLayers: 1
  biasConnect: [1]
 inputConnect: [1]
 layerConnect: [0]
outputConnect: [1]
targetConnect: [0]

  numOutputs: 1  (read-only)
  numTargets: 0  (read-only)
numInputDelays: 0  (read-only)
numLayerDelays: 0  (read-only)

subobject structures:

      inputs: {1x1 cell} of inputs
      layers: {1x1 cell} of layers
     outputs: {1x1 cell} containing 1 output
     targets: {1x1 cell} containing no targets
      biases: {1x1 cell} containing 1 bias
 inputWeights: {1x1 cell} containing 1 input weight
 layerWeights: {1x1 cell} containing no layer weights

functions:

    adaptFcn: 'trains'
     initFcn: 'initlay'
  performFcn: (none)
    trainFcn: 'trainr'

parameters:

  adaptParam: .passes
   initParam: (none)
 performParam: (none)
  trainParam: .epochs, .goal, .show, .time

weight and bias values:

        IW: {1x1 cell} containing 1 input weight matrix
        LW: {1x1 cell} containing no layer weight matrices
         b: {1x1 cell} containing 1 bias vector

other:

    userdata: (user stuff)


wts =
```

```
    0.5000    0.5000
    0.5000    0.5000


net =

    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

      numOutputs: 1  (read-only)
      numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

          inputs: {1x1 cell} of inputs
          layers: {1x1 cell} of layers
         outputs: {1x1 cell} containing 1 output
         targets: {1x1 cell} containing no targets
          biases: {1x1 cell} containing 1 bias
     inputWeights: {1x1 cell} containing 1 input weight
     layerWeights: {1x1 cell} containing no layer weights

    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
         trainFcn: 'trainr'

    parameters:

      adaptParam: .passes
       initParam: (none)
    performParam: (none)
      trainParam: .epochs, .goal, .show, .time

    weight and bias values:
```

```
        IW: {1x1 cell} containing 1 input weight matrix
        LW: {1x1 cell} containing no layer weight matrices
         b: {1x1 cell} containing 1 bias vector

   other:

       userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
TRAINR, Epoch 225/300
TRAINR, Epoch 250/300
TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.


net =

   Neural Network object:

   architecture:

      numInputs: 1
      numLayers: 1
    biasConnect: [1]
   inputConnect: [1]
   layerConnect: [0]
  outputConnect: [1]
  targetConnect: [0]

     numOutputs: 1  (read-only)
     numTargets: 0  (read-only)
  numInputDelays: 0  (read-only)
  numLayerDelays: 0  (read-only)

   subobject structures:

         inputs: {1x1 cell} of inputs
         layers: {1x1 cell} of layers
        outputs: {1x1 cell} containing 1 output
        targets: {1x1 cell} containing no targets
         biases: {1x1 cell} containing 1 bias
```

```
    inputWeights: {1x1 cell} containing 1 input weight
    layerWeights: {1x1 cell} containing no layer weights

  functions:

      adaptFcn: 'trains'
       initFcn: 'initlay'
     performFcn: (none)
      trainFcn: 'trainr'

  parameters:

     adaptParam: .passes
      initParam: (none)
   performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

          IW: {1x1 cell} containing 1 input weight matrix
          LW: {1x1 cell} containing no layer weight matrices
           b: {1x1 cell} containing 1 bias vector

  other:

      userdata: (user stuff)

ans =

trainr


a =

  (1,1)        1
  (2,2)        1
  (2,3)        1
  (2,4)        1


ac =

     1     2     2     2


result =

     7
```

R =

```
    5    6    6    6    5    7    6    6    7    0
```

p =

```
  365.0000  141.0000   20.2000  590.0000
         0   45.0000    4.0000   36.0000
```

net =

```
    Neural Network object:

    architecture:

        numInputs: 1
        numLayers: 1
      biasConnect: [1]
     inputConnect: [1]
     layerConnect: [0]
    outputConnect: [1]
    targetConnect: [0]

      numOutputs: 1  (read-only)
      numTargets: 0  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)

    subobject structures:

          inputs: {1x1 cell} of inputs
          layers: {1x1 cell} of layers
         outputs: {1x1 cell} containing 1 output
         targets: {1x1 cell} containing no targets
          biases: {1x1 cell} containing 1 bias
    inputWeights: {1x1 cell} containing 1 input weight
    layerWeights: {1x1 cell} containing no layer weights

    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
      performFcn: (none)
        trainFcn: 'trainr'

    parameters:
```

```
    adaptParam: .passes
     initParam: (none)
   performParam: (none)
    trainParam: .epochs, .goal, .show, .time

  weight and bias values:

          IW: {1x1 cell} containing 1 input weight matrix
          LW: {1x1 cell} containing no layer weight matrices
           b: {1x1 cell} containing 1 bias vector

  other:

     userdata: (user stuff)


wts =

  0.5000    0.5000
  0.5000    0.5000


net =

  Neural Network object:

  architecture:

     numInputs: 1
     numLayers: 1
    biasConnect: [1]
   inputConnect: [1]
   layerConnect: [0]
  outputConnect: [1]
  targetConnect: [0]

    numOutputs: 1  (read-only)
    numTargets: 0  (read-only)
  numInputDelays: 0  (read-only)
  numLayerDelays: 0  (read-only)

  subobject structures:

       inputs: {1x1 cell} of inputs
       layers: {1x1 cell} of layers
      outputs: {1x1 cell} containing 1 output
      targets: {1x1 cell} containing no targets
       biases: {1x1 cell} containing 1 bias
  inputWeights: {1x1 cell} containing 1 input weight
  layerWeights: {1x1 cell} containing no layer weights
```

```
    functions:

        adaptFcn: 'trains'
         initFcn: 'initlay'
       performFcn: (none)
        trainFcn: 'trainr'

    parameters:

       adaptParam: .passes
        initParam: (none)
     performParam: (none)
      trainParam: .epochs, .goal, .show, .time

    weight and bias values:

              IW: {1x1 cell} containing 1 input weight matrix
              LW: {1x1 cell} containing no layer weight matrices
               b: {1x1 cell} containing 1 bias vector

    other:

        userdata: (user stuff)

TRAINR, Epoch 0/300
TRAINR, Epoch 25/300
TRAINR, Epoch 50/300
TRAINR, Epoch 75/300
TRAINR, Epoch 100/300
TRAINR, Epoch 125/300
TRAINR, Epoch 150/300
TRAINR, Epoch 175/300
TRAINR, Epoch 200/300
TRAINR, Epoch 225/300
TRAINR, Epoch 250/300
TRAINR, Epoch 275/300
TRAINR, Epoch 300/300
TRAINR, Maximum epoch reached.


net =

    Neural Network object:

    architecture:

       numInputs: 1
       numLayers: 1
      biasConnect: [1]
```

```
    inputConnect: [1]
    layerConnect: [0]
   outputConnect: [1]
   targetConnect: [0]

     numOutputs: 1  (read-only)
     numTargets: 0  (read-only)
  numInputDelays: 0  (read-only)
  numLayerDelays: 0  (read-only)

  subobject structures:

          inputs: {1x1 cell} of inputs
          layers: {1x1 cell} of layers
         outputs: {1x1 cell} containing 1 output
         targets: {1x1 cell} containing no targets
          biases: {1x1 cell} containing 1 bias
    inputWeights: {1x1 cell} containing 1 input weight
    layerWeights: {1x1 cell} containing no layer weights

  functions:

       adaptFcn: 'trains'
        initFcn: 'initlay'
      performFcn: (none)
       trainFcn: 'trainr'

  parameters:

     adaptParam: .passes
      initParam: (none)
    performParam: (none)
     trainParam: .epochs, .goal, .show, .time

  weight and bias values:

            IW: {1x1 cell} containing 1 input weight matrix
            LW: {1x1 cell} containing no layer weight matrices
             b: {1x1 cell} containing 1 bias vector

  other:

       userdata: (user stuff)


ans =

trainr
```

```
a =

   (1,1)     1
   (2,2)     1
   (2,3)     1
   (1,4)     1


ac =

    1    2    2    1


result =

    6


R =

    5    6    6    6    5    7    6    6    7    6

>>
```

Bibliography

BIBLIOGRAPHY

AccuTraffic http://www.accuweather.com/www/accutraffic/traffic.html Date last referred: 5.7.2002

Artificial Neural Networks. Saint Louis University School Of Business & Adminstration http://hem.hj.se/~de96klda/NeuralNetworks.htm Date last referred : 12.02.2003

Artificial Neural Networks Technology. http://psychology.about.com/gi/dynamic/offsite.htm?site=http%3A%2F%2Fwww.dacs.dtic.mil%2Ftechs%2Fneural%2Fneural_ToC.html last date referred : 12.2.2003

Attrasoft digital nervous system. http://attrasoft.com/whatisnn.htm Date last referred: 31.8.2002

Azree Idris (1999). *MATLAB for Engineering Students.* Prentice Hall

Artificial Neural Networks Technology http://www.dacs.dtic.mil/techs/neural/neural_ToC.html Date last referred: 5.7.2002

Competitive learning http://www.shef.ac.uk/psychology/gurney/notes/l7/section3_2.html#SECTION00020000000000000000 Date last referred : 12.02.2003

Danaco, Eric (1991). *Neural Networks.* MacMillan, London.

Fausette, Laurene (1994), *Fundamentals of Neural Networks : Architecture, Algorithms and Applications*. Prentice Hall International, Inc. New Jersey.

Frenzel, Louis E. (1987). *Crash Course in Artificial Intelligence and Experts Systems*. Howard W. Sams & Co. Indianapolis.

Gordon, V. S., and Bieman, J. M(1995). Rapid Prototyping: Lessons Learned. IEEE Software. (pp 86-87)

Hawryszkiewyez, Igor (1998), *Introduction to System Analysis and Design*, 4th Edition, Prentice Hall Australia Pty. Ltd. Sydney.

Heng Soon Leng (2001), *Network Traffic Controller Using Artificial Neural Network*. B.Comp.Sc. Thesis, Faculty of Computer Science & Information Technology, University of Malaya, Kuala Lumpur.

J Scheikoff, Robert (1997) , *Artificial Neural Networks*. McGraw-Hill, New York.

Kendall, Kenneth E. and Kendall, Julie E(1998). *System Analysis and Design* 4th Edition. Prentice Hall Inc.

Lim Kian Sinn, Vincent (2002), *Smart Electronic Travel Agency Back-End System*. B.Comp. Sc. Thesis, Faculty of Computer Science & Information Technology, University of Malaya, Kuala Lumpur.

Mari Mari.com

http://www.marimari.com/content/malaysia/transport/main.html Date last referred: 6.9.2002

MATLAB Help (2001) The MathWork Inc.

McCarthy, John (2000). *Basic Question. What is Artificial Intelligent ?* URL : http://www.formal.stanford.edu/jmc/whatisai/node1.html.Date last referred: 3.12.2002

My Web Travel Channel

http://www.myweb.com.my/travel/16malaysia/malaysia.html Date last referred: 30.9.2002

Neural Networks

http://www.accurate-automation.com/products/nnets.htm Date last referred: 3.12.2002

Neural Networks : What are Neural Networks ?

http://www.emsl.pnl.gov:2080/proj/neuron/neural/neural.ann.html Date last referred: 3.12.2002

Pfleeger, Shari Lawrence (1998), *Software Engineering – Theory and Practice*, Prentice Hall International, Inc, New Jersey.

Road Transport Department Homepage http://www.jpj.gov.my/hiway.htm Date last referred: 6.9.2002

Smith, Leslie (1998), *An Introduction to Neural Networks*. URL : http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html Date last referred: 30.9.2002

Subject: What does unsupervised learning *learn?*

ftp://ftp.sas.com/pub/neural/FAQ2.html#A_unsup *Date last referred : 12.2.2003*

TrafficCast - The Power of Prediction

URL:http://www.trafficcast.com/corporate/about.html Date last referred: 30.9.2002

TrafficLinq - Traffic Information

URL:http://www.trafficlinq.com/trafficinformation.htm Date last referred: 6.9.2002

Turban, Efram ( ), *Expert System and Applied Artificial Intelligence.* MacMillan Publishing Company, New York.

Winston, Patrick Henry ( ), *Artificial Intelligence*, 3rd Edition. Addison-Wesley Publishing Co, Massachusetts.

W.S. Sarle, editor. *Neural Network FAQ. Newsgroup* : comp.ai.neural-nets, 1998. URL:ftp://ftp.sas.com/pub/neural/FAQ.html.Date last referred: 5.1.2002

Yap Hui Sun (2001), *Intelligent Agent For Electronic Commerce.* B.Comp. Sc. Thesis, Faculty of Computer Science & Information Technology, University of Malaya, Kuala Lumpur.