A Computational Intelligence (CI) Approach to the Precision Mars Lander Problem

Brian Birge
Gerald Walberg

Performance Report submitted under NASA Grant #NAG-1-01111 for the period 5/16/01 to 5/15/02

Abstract

A Mars precision landing requires a landed footprint of no more than 100 meters. Obstacles to reducing the landed footprint include trajectory dispersions due to initial atmospheric entry conditions such as entry angle, parachute deployment height, environment parameters such as wind, atmospheric density, parachute deployment dynamics, unavoidable injection error or propagated error from launch, etc. Computational Intelligence (CI) techniques such as Artificial Neural Nets and Particle Swarm Optimization have been shown to have great success with other control problems. The research period extended previous work on investigating applicability of the computational intelligent approaches. The focus of this investigation was on Particle Swarm Optimization and basic Neural Net architectures. The research investigating these issues was performed for the grant cycle from 5/15/01 to 5/15/02. Matlab 5.1 and 6.0 along with NASA's POST were the primary computational tools.

Nomenclature

Activation Function
Artificial Neural Net
Backpropagation
Bank Angle
Computational Intelligence
Epoch
Evolutionary Computation
Firewire
Generalization
Genetic Algorithms
Hyperspace
Inverse Control
Iris Data Set
LM
Mars Sample Return Mission
MarsGram
Matlab
Multilayer
Open Loop
Particle Swarm Optimization
POST
Sigmoid

Important Files Used

Data & input:

bkb3guess.dat
chutestates.dat
fulliris.txt
prntblk.bak
prntblk.dat
pvstates.dat
bc4batch.mat
chutestatplot.mat
copy_of_bc1871.mat
copy_of_lc.mat
copy_of_lc1747_18.mat
copy_of_m2001newbase.mat
copy_of_rwp1rps.mat
forward.mat
forward3.mat
inverse.mat
IrisWtsLM.mat
lc4batch.mat
lcsaveas.mat
nettrain-76init-bc17baseline.mat
nettrain-76init-bc17baselineMOD.mat
nettrain-nominit-bc17baseline.mat
nettrain-nominit-bc17baselineMOD.mat
nettrainer-weights.mat
nettrainer-wts-1layer.mat
nettrainer2-wts-1layer-inertialBPX.mat
nettrainer2-wts-1layer-inertiallm.mat
nettrainer2-wts-1layer-inertialPSO.mat
pathdat1.mat

Program & Control:

chutestatplot.m
demonormalize.m
f6.m
learnga.m
learnlm.m
load_DATA_mat.m
m2001viewsite.m
MatPathData.m
nettrainer2.m
normalize.m
p2check.m
p3dbatchSPACE2000.m
p3dbatchspace2000PSO.m
pso.m
testf6.m
testnettrainer.m
testpso.m
tga2.m
tlm2.m
tpso2.m
trainbpx.m
trainga.m
trainlm.m
trainpso.m
unwrapmat.m
wrapmat.m

Introduction

The Mars Rover Sample Return Mission (MRSR) requirements make the development of an accurate descent control system challenging. A Mars precision landing requires a landed footprint of no more than 1 kilometer. In contrast, the Viking mission had a landing footprint of over 100 kilometers (excellent for that particular mission requirement). The current state of the art's footprint is still in the magnitude of kilometers.

Obstacles to reducing the landed footprint include trajectory dispersions due to initial atmospheric entry conditions (entry angle, parachute deployment height, etc.), environment (wind, atmospheric density, etc.), parachute deployment dynamics, unavoidable injection error (propagated error from launch on), etc.

In recent years, a number of computational techniques that lend themselves particularly well to control problems have become available. In broad categories they are Fuzzy Systems (FS), Artificial Neural Networks (ANN), and Evolutionary Computation (EC). Fuzzy Systems are based on fuzzy logic which is in turn based on the way our mind deals with incomplete and/or inaccurate information. Neural Nets are modeled after the spatial structure of the brain and allow 'connectionist' learning properties. Evolutionary computation paradigms such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) are loosely modeled after biological evolution and optimization. These techniques have been shown individually to work very well in solving a large number of problems in linear and nonlinear system identification, modeling, and control.

Until recently FS, ANN, and EC were totally separate fields with very little interaction. A growing number of researchers and practical engineers are discovering that a combination of two or more of these methods offers advantages that a single one lacks.

Using an ANN in a control system we add fault tolerance, distributed (connectionist) representation properties, and the ability to learn optimal responses to new input. If we add an FS 'shell' we include high level rule/decision abilities as well as comparative reasoning. Combining techniques this way is called Computational Intelligence (CI).

This paper describes ongoing research into using CI strategies for the MRSR mission as well as other Mars Lander missions.

Problem Setup

The problem examined is that of the re-entry of the proposed Mars 2001 lander. The lander specifications and landing site are all taken from the Mars 2001 conditions as related by Powell, Striepe, and Queen. The initial conditions are defined as within the cloud set developed by Powell. These initial conditions reflect a point of parachute deployment of the lander after it has completed the first leg of re-entry and consist of approximately 2000 possible starts within 10km dispersion. The 2000 points were reduced to 76 points by taking the convex hull of both the position and velocity subspaces. This results in a data set of initial conditions reflecting the extremes in handoff position and velocity. In addition, one initial point was computed from the averages of the complex hull and taken to be the nominal initial condition. These initial conditions represent the currently expected diversion of any Mars re-entry.

Lander Scenarios

The input deck setup to run with these initial conditions was a ballistic parachute deceleration followed by 'fire-wired' steerable parachute descent. This scenario has been determined from previous work to have the best tradeoffs between practicality, stability, and performance.

POST was used to target and optimize to desired end states. Because this input deck has been examined in the previous work only a brief overview is presented here.

The lander simulation starts off with one of the 77 representative initial conditions. This includes all position and velocity data along with a time of day and basic weather model slightly modified from the MarsGRAM model. The modification is only in the wind strengths, using the Braun multipliers.

The Post deck deploys the ballistic chute and at some controllable time it turns the ballistic parachute into a steerable parachute. Then several events follow where the controller is given the opportunity to change bank angle of the lander, all with a goal of minimizing distance to target. At a height 1000m above surface, where surface is taken to be at 2500m, the parachute is jettisoned and the lander performs a reverse gravity turn descent on thrusters to achieve desired end velocity.

Controller Training

This section describes motivation and previous research.

For this deck, the POST optimizer was turned off. The initial guesses for the controller (bank angles and times of occurrence) were generated in an offline data file to be fed to the deck on simulation start. For the nominal initial condition, which was the average of the extreme initial conditions, a set of controls was found by trial and error that would achieve fairly good results. These 'nominal initial' controls were then perturbed over a fairly large range for each initial condition, taking into account limits on bank angle, etc.

For a situation where there was only one bank angle command at a controller specified time, we had 2 controls:

1) time to turn parachute into steerable, also same as first bank angle command time
2) bank angle command

From previous work it was seen that a series of commanded bank angle turns yielded better results, so for a deck with 3 events we have 6 controls. Each set of these controls is perturbed parametrically and re-run on the same deck, i.e. for a deck with 2 control parameters the initial controls are run, then control 1 is changed slightly, deck is run again, then control 1 is changed again, and deck is run again. This is done until we get the whole range of possible (or at least reasonably expected) values for the control 1. Then control 2 is slightly changed and the whole parametric study of control 1 is re-done for the new control 2. The whole thing is repeated until we get a database reflecting the entire ranges of control 1 and 2.

At the end of each run, all starting and ending trajectory data is saved. So for a deck with 6 controls and reasonably non-limiting control ranges, we have an immense amount of data to sift through. This data was used to train an artificial neural net as an inverse controller. The ending conditions along with initial conditions (inertial or relative position and velocity – both were tried) were the net's inputs and the set of controls needed were the output. So for our 6 control deck we have a net with 12 inputs and 6 outputs.

Not all of the collected data was used for training the net, about 20% was held back for testing after training was done. The trick to training is to make sure the data is well prepared.

To this end a variety of things were tried. First, all the training data was normalized. This is a standard practice when using the traditional training algorithms. This required developing an algorithm that would normalize whatever data was given to it and save the normalizing factors so future unknown data could also be normalized in exactly the same way. This was coded as a matlab function where you could choose several types of normalization of scalars, vectors, or matrices. Also the inverse function was written, to get the normalized data (usually a large matrix) back into the original form.

The above processes yielded several data sets that were all extremely large. These data sets were fed to an algorithm that built up an artificial neural net. The algorithm started with a single hidden layer and a single hidden neuron and attempted to train. If it didn't train successfully within a user adjustable time frame (usually was set to 10000 epochs) then another neuron was added to the hidden layer and the process started over. There were several training methods employed also. The control was standard backpropagation, essentially gradient descent. The problem with this is that it is slow and prone to local minima in the problem space. Since the data set was nonlinear and hugely multi-dimensional and also just huge in amount, there were a lot of chances to run into local minima. And that is just what happened. Each training run took several days to complete using the NASA VAB computers because of the size of the sets but almost never did the runs train to the minimum error criterion. Rather they stopped at the maximum

number of hidden neurons. Each training run showed a quick drop in error initially to a plateau that just would not go away, no matter which training set was tried.

The training sets were then pared down in size by about 50%. These seemed to train well but generalization was a distinct problem. That is, feeding in data that the net had not seen yet (from the test set) did not yield the desired outputs. In fact, the output was extremely sensitive to very slight changes in input parameters and would output unusable controls. For example, an input set might yield a desired control of >90 degrees in the bank angle, a sure way to crash the simulation.

The above was then tried with the LM training method. We were still using the same neural net architecture and training sets (normalized full, non-normalized full, normalized subset, non-normalized subset) but the training of the net was performed differently.

LM is a method that can be considered as an improved backpropagation method. It is much faster and is also better at avoiding local minima in the problem space (those plateaus we ran into during previous training). But it still suffers from the heritage of being basically a gradient descent algorithm. That is all background on the previous work. After the above it was decided to take a step back and do an exhaustive examination on the computational intelligence paradigm itself. Was it the huge amount of data that was the problem, was it a flaw in the whole approach, or is it something else or a combination of other things?

Generalized Study

To this end, several strategies were examined without using the POST generated data, rather more generic test sets provided by the literature were used. The basic architecture was examined including the activation functions, as were the training methods, all with a view to a more basic understanding of the process. Generalization, training time, and processing power were the important parameters under consideration.

Several small parametric studies examined parameter modification in artificial neural nets. The goals were to understand how these parameters affect:

1) Generalization, how does the net handle inputs its never seen?
2) Training Time, epochs needed to drive error to desired
3) Processing Load, number of neurons needed to successfully train

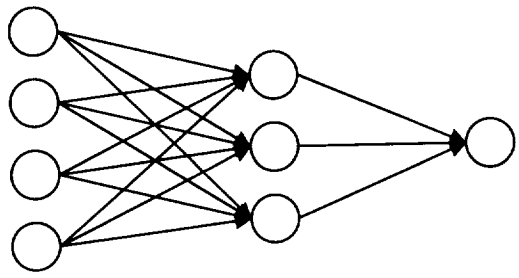
In brief, the following areas were examined:

1) Neural Net Architectures
2) Test Sets
3) Data Representation
4) Training Methods

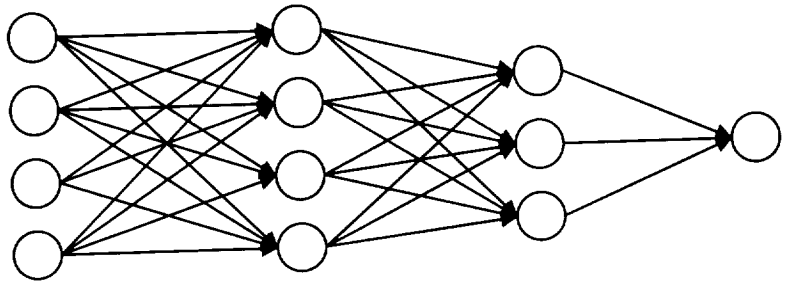
Neural Net Architectures

There are 2 basic architectures that were looked at, a single hidden layer network and a dual hidden layer network. The effect of one hidden layer vs. two hidden layers was examined with respect to: Generalization, Training Time, and Processor Load.

1) Single hidden layer network



2) Dual hidden layer network



Each architecture has a varying number of nodes in each layer, above is just for illustration.

The bare minimum of nodes for each permutation was calculated in Matlab. This was done by starting with a single neuron in each hidden layer and attempting to train. If the error goal was not met then a neuron was added incrementally and training started again. For the dual layer case, the second hidden layer would be incremented first and then the first layer. This was so that one hidden layer would not be much larger than the other. The neuron numbers grew until the error goal was reached within 5000 epochs. For the single hidden layer, a maximum of 20 hidden layer neurons was specified. For the dual hidden layer, a maximum of 10 neurons for each layer was specified. If incrementing to these numbers still did not reach the error goal then that 'architecture/activation function/training set %/data format' combination was not trainable.

This 'growth' of neurons was implemented specifically to find the minimum number of neurons needed to train each parameterized combination. Since the eventually designed controller is meant to be onboard the lander, the processor load must be as streamlined as possible. Finding the minimum number of neurons to train is a big step in this.

The hidden layer for the networks consisted of sigmoid functions using either the Matlab 'logsig' function and 'tansig'. When they were used is discussed in the data representation section. All output layers were linear using the Matlab 'purelin' function.

The classic activation function used for the hidden layer neurons is called the 'logistic sigmoid' function and is given by:

$$f(x) = \frac{1}{1+e^{-x}}$$

This function has an output between 0 and 1 and has a characteristic curve giving it the common name: "S-function". The Matlab name for this function is 'logsig'.

The second function is a variation on the sigmoid function called the 'hyperbolic tangent' and looks like:

$$f(x) = \tanh(x)$$

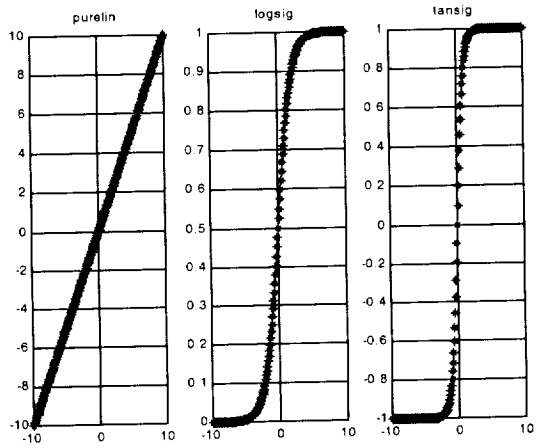
The function has an output between [–1, 1] with the Matlab function name of 'tansig'.

The standard linear function was used for the output layers:

$$f(x) = x$$

Notice the difference between the logsig and the tansig functions are only in the output range. Tansig is symmetric about zero while logsig is symmetric about one.

The input/output graphs of the activation functions used look like the following figure.



Both the sigmoid and hyberbolic tangent are non-linear equations and are known to work well with non-linear problems. The linear function is traditionally used in the output layer of a pattern classification network.

Test Sets

Before re-trying the POST data on the neural net, three standard test sets were examined.

1) XOR
2) Iris
3) Schaffer f6

The test sets were chosen because they represent a complexity increasing standard non-linear test bed for computational intelligence algorithms. That is, if the methods work on these 3 known data sets then they should also work on the POST generated data sets or POST generated real-time data. If they don't work on the POST data at that point then logic would suggest that the method of gathering data should be fine tuned rather than the training algorithm.

XOR

The XOR test set is the standard modified OR logic, shown by the following,

Input	Output
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

Iris

The 'Iris' data set is a famous set of parameters that describes 3 separate species of Iris flowers. They are classified by sepal width, sepal length, petal width, and petal length. There are 150 states in the set, each of the 3 flowers gets 50 states. So, this set can be thought of as a 4 input, single output system. The four inputs would be the petal/sepal parameters and the output is a number that corresponds to the flower classification. In this case, flower 1 = 1, flower 2 = 2, and flower 3 = 3. The Iris data set is a non-linear mapping, hence it is an excellent test base for artificial neural network and more general computational intelligence research. The full non-normalized Iris Data Set is presented in the appendix.

For example, the first 6 entries of the set in its raw form look like:

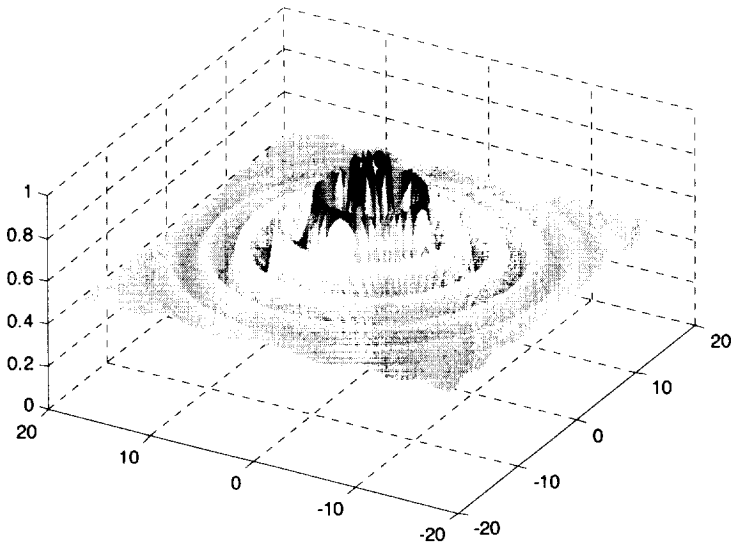
Species	Sepal length	Sepal width	Petal length	Petal width
1	49	30	14	2
1	51	38	19	4
1	52	41	15	1
1	54	34	15	4
1	50	36	14	2
1	57	44	15	4

The data was split up into three parts, representing each flower classification. Then a percentage of each of these three subsets was taken for training the neural nets and the rest was taken for testing. Splitting the full set up into 3 subsets based on each species guaranteed that each species would get equally represented training. 50% of the training set was used leaving 50% for testing respectively.

Schaffer f6

The f6 test set is derived from the f6 equation. It is highly nonlinear with a single known global minimum at (0,0). There are plenty of opportunities for computational intelligence algorithms to get stuck in local minima, so this test set is very good for testing that aspect.

$$z = 0.5 + (\sin^2(\mathrm{sqrt}(x^2+y^2)) - 0.5)/((1+0.01*(x^2+y^2))^2)$$



Data Representation

The Iris data set represents characteristics for 3 different species of flowering Iris plant. This data can be represented two separate ways easily in a neural network.

First, the net can be trained to output a single number. This number will represent one of the three flowers. In this project the specific numbers are: flower 1 = 0.33, flower 2 = 0.66, flower 3 = 1.0. If the net output a number within +/-20% of the target, that was deemed close enough to consider as a match. For example, anything between .80 and 1.2 would equal species 3.

Secondly, the net can be trained to output a single state. For this, three outputs were used instead of one. The target was:

[1,0,0] = flower 1

[0,1,0] = flower 2

[0,0,1] = flower 3

The output was considered a match if the magnitude of one of the columns was larger than the other two. For example, a result of [0.6,0.9,-0.1] would be interpreted as indicating flower 2 since 0.9 is larger than the other 2 columns. Then we just filtered for this condition with basic logic.

In addition we can choose to normalize or not. Traditional methods call for normalizing all data. The Matlab implementations used in the coding of this project do not require normalization in order to produce results. So a run was performed with normalization and without and then compared for the properties of generalization, training time, and processor load.

The first six entries of normalized Iris data look like:

0.3333	0.6203	0.6818	0.2029	0.0800
0.3333	0.6456	0.8636	0.2754	0.1600
0.3333	0.6582	0.9318	0.2174	0.0400
0.3333	0.6835	0.7727	0.2174	0.1600
0.3333	0.6329	0.8182	0.2029	0.0800
0.3333	0.7215	1.0000	0.2174	0.1600

It is the same data, just normalized. Similar concepts were applied to the XOR data while the f6 dataset was left alone since it has one minimum and is already normalized. We can relate this to the MRSR lander problem by recognizing that if we don't have to normalize we are better off computationally, especially since normalization requires us to know the extremes if the raw data which we may not know. But normalization may make the net (hence the controller) train to new data quicker and/or with less computational power. It's a trade off we need to know more about. Also we would like to know if the controller will work better if we had the output neurons hold a single value for bank angle or perhaps 2 output neurons each with a bank angle, one representing a negative bank.

For the normalized data sets the 'logsig' function was used in the hidden layers as we only need to worry about the range [0, 1] for the data. For the non-normalized data sets the 'tansig' function was used in the hidden layers.

Training Algorithms

The training algorithms investigated have been:

1) Gradient Descent, i.e. Standard Backpropagation
2) LM (modified gradient descent)
3) Particle Swarm Optimization (PSO)

Gradient Descent
Standard Backpropagation algorithm, weights of the net are changed according to a difference in error between previous and present values. This is the control case. Prone to local minima.

LM
A much improved version of gradient descent. It is faster, more robust, and less computationally intensive overall but suffers from some of the same handicaps such as it can train any network as long as its weight, net input, and transfer functions have derivative functions. Less prone to local minima but still easily caught.

Particle Swarm Optimization
In theory the PSO method is much faster and not prone to local minima. In practice of this study it is much slower than LM but much better at driving down to convergence though it still seems to have some problems with minima. An advantage of PSO is that neural trained with this method seem to be better at generalization than nets trained with traditional methods. The heart of the PSO algorithm is the velocity equation:

$$Vel(k+1)=rand1*vel(k)+rand2*(pbest-pos(k))+rand3*(gbest-pos(k)),\quad k \text{ is the discrete time step}$$

$$Pos(k+1)=Pos(k)+Vel(k+1)$$

Pbest is the personal best valued position for the particular particle and Gbest is the global best position so far for entire population of particles. The variables rand1, rand2, and rand3 are random numbers on the interval [0,1]. Pos(k) is present position of a single particle, and has as many dimensions as the problem. This is the basic form of the PSO equation. The most common form is to include an inertia term and acceleration constants:

$$\begin{aligned} Vel(k+1)= &[inertia*rand1*vel(k)] \\ &+[ac1*rand2*(pbest-pos(k))] \\ &+[ac2*rand3*(gbest-pos(k))] \end{aligned}$$

The acceleration constants ac1 and ac2 are fixed and have been determined by Dr. Eberhart and Dr. Shi to be best for most problems when both are set to 2. For our problem these values actually degrade performance and it seems the best values are 2 for ac1 and 1 for ac2. This weighs the influence of local best particles more than global. There are lots of rules of thumb but not much in the way of heuristics for this method as it is still relatively new. That isn't necessarily a big handicap since the algorithm is stochastic in nature anyway. The recommended inertia term is a linear function with time that decreases from 0.9 to 0.4 or a static term set at 1.4. Performance is definitely better with the linear term rather than the constant. Essentially as training progresses the influence of past velocity becomes smaller. I have tried many sets of the above parameters with varying degrees of success.

For neural net training there are another couple of important parameters. The initial weight size (corresponds to initial positions of particles) and the maximum velocity components allowed are very important. Literature mentions that a maximum velocity component of 4 seems to work well but in this study that is 3 orders of magnitude too large. Also, you would logically think that a large initial weight spread would make it easier to find the global best but experience in this research has shown the opposite is the case. It seems that for neural net training, if we start out with a particle population tightly clustered together they like to travel outward to find the solution. Why this happens is an active area of investigation. Depending on the error over time, the maximum velocity component is modified. Sometimes during what seems like a hang in a local minimum well the velocity components will saturate to the max values. If this happens over a series of training passes (various lengths of time have been tried) the max velocity will get a small decrease or increase (decrease seems to work better). This seems to jitter the problem out of local minima.

Results

Here is the output on the Matlab workspace for a particular run (training set 50%, hidden layers as shown, Iris data set, normalized, training method LM):

****

for 50% of the training set:

1) Single Output/Single Hidden layer: hidden=logsig, output=purelin
   Architecture from left to right: 2 1
   Epochs to train: 33
   Generalization result: 62/75 passed test
   or 82.6667%

2) Triple Output/Single Hidden layer: hidden=logsig, output=purelin
   Architecture from left to right: 2 3
   Epochs to train: 27
   Generalization result: 70/75 passed test
   or 93.3333%

3) Single Output/Double Hidden layer: hidden1=purelin hidden2=logsig, output=purelin
   Architecture from left to right: 3 3 1
   Epochs to train: 49

Generalization result: 71/75 passed test
or 94.6667%

4) Triple Output/Double Hidden layer: hidden1=purelin hidden2=logsig, output=purelin
Architecture from left to right: 6 5 3
Epochs to train: 56
Generalization result: 72/75 passed test
or 96%

All training sets followed a similar trend with virtually no difference for these data sets between using logsig and tansig in the hidden layers. LM outperformed Gradient Descent in all areas while PSO outperformed LM slightly except in time to train where it was very slow. This goes against the literature and warrants further study.

Conclusion

A network can have quite good results by recasting the data into multiple outputs instead of one, using only one hidden layer, and using either the logsig/linear or tansig/linear architectures. Overall these have the best performance in terms of an inductive average of the three metrics: generalization, training time, and processing load.

For the MRSR problem, generalization is the most important criteria. This paper shows that a high degree of generalization is possible with a simple non-linear artificial neural network that uses very little processing power and takes almost no time to train. For any application that has the potential to injure people, generalization must be the overriding factor and percentages even higher than the 96% shown possible with this study would need to be achieved.

If processing power and time to train (for example in a real time application) are important, Gradient Descent may not be a smart choice. Both LM and PSO outperformed it in those areas.

This study confirmed that a CI approach may be a valid and preferred approach which leads to the conclusion that the initial problem formulation of the MRSR along with the data preparation and mining need to be re-examined.

References

1. Bennett, T., Fox, R., "Testing and Development of the NASA X-38 Parafoil Upper Surface Energy Modulator (patent pending)", AIAA-99-1753

2. Birge, B.K., Walberg, G.D.,"An Investigation of Terminal Guidance and Control Techniques for a Robotic Mars Lander", under NAG-1-2086

3. Brauer, G.L., Cornick, D.E., Olson, D.W., Petersen, F.M., Stevenson, R., "Volume I, Formulation Manual, Six-Degree-of-Freedom Program To Optimize Simulated Trajectories (6D POST)", NAS1-18147

4. Braun, R.D., Powell, R.W., Cheatwood, F.M., Spencer, D.A., Mase, R.A., "The Mars Surveyor 2001 Lander: A First Step Toward Precision Landing", IAF-98-Q.3.03

5. Braun, Robert D. et. al., "The Mars Surveyor 2001 Lander: A First Step Toward Precision Landing", IAF-98-Q.3.03, 49th IAF Congress, Melbourne Australia, Sept 28-Oct 2, 1998

6. Carman, G., Ives, D., Geller, D., "Apollo-Derived Mars Precision Lander Guidance", AIAA Atmospheric Flight Mechanics Conference, August 10-12, 1998, Boson, MA, AIAA 98-4570

7. Carter, P., Smith, R., "Mars Rover Sample Return – Lander Performance", AIAA 89-0633

8. Fogiel, Dr. M., Staff of Research and Education Association, "Handbook of Mathematical, Scientific, and Engineering Formulas, Tables, Functions, Graphs, Transforms", REA,1994

9. Haykin, Simon, "Neural Networks, A Comprehensive Foundation second edition", Prentice Hall, Inc. 1999

10. Ingoldby, R.N., "Guidance and Control System Design of the Viking Planetary Lander", J. Guidance and Control, Vol. 1, No. 3, May-June 1978, pp. 189-196

11. Justus, C.G., James, B.F., Johnson, D.L., "Mars Global Reference Atmospheric Model (Mars-GRAM 3.34): Programmer's Guide", NASA Technical Memorandum 108509

12. Knacke, T.W., "Parachute Recovery Systems Design Manual", Para Publishing, PO Box 4232, Santa Barbara, CA, 93140-4232, USA, ISBN 0-915516-85-3

13. Knacke, T.W., "Parachute Recovery Systems", NWC TP 6575

14. Krishnamurthy, Karthik and Ward, Donald T., "An Intelligent Flight Director for Autonomous Aircraft", AIAA 2000-0168

15. Lin, Chin-Teng, George Lee, C.S., "Neural Fuzzy Systems, A Neuro-Fuzzy Synergism to Intelligent Systems", Prentice Hall P T R, 1996

16. Machin, R., Stein, J., and Muratore, J., "An Overview of the X-38 Prototype Crew Return Vehicle Development and Test Program", AIAA-99-1703

17. Ro, T., Queen, E., "Study of Martian Aerocapture Terminal Point Guidance", AIAA Atmospheric Flight Mechanics Conference, August 10-12, 1998, Boson, MA, AIAA 98-4571

18. McEneaney, W.M., Mease, K.D., "Error Analysis for a Guided Mars Landing", J. Astronautical Sciences, Vol. 39, No. 4, Oct-Dec 1991, pp. 423-445

Appendix

Steerable Parachute POST input deck:

```
l$search
c*************************************************************************
c       m2001newbase2-liftchute-targref.inp
c
c
c   NEW altitude target: 2700m (was 2500m), this allows relaxation of altitude
c   targeting tolerances (+/- 150m instead of 1) and simulates about +200m above surface where we should
c   probably have a seperate terminal descent simulation anyway.
c
c
c   events 27 has been taken out and event 26 is a criterion
c   to stop bankangle maneuvers
c
c   Mars Lander from LaRC hand-off to 2500 m
c   no optimization, fixed bank on steerable chute (0 deg)
c
c   Parachute diameter = 13.0 m
c
c   Marsgram atmosphere - Feb. 3, 2002; 00 hr, 00 min, 0.0 s
c   ln(pres) and atem input as tables
c   Marsgram winds*Braun multipliers input as tables
c
c   This deck optimizes (min dprng1)
c
c   Parachute is lifting with 4 steering events
c     1st guesses are input from 'bankanglestudy.dat'
c     and are just constant turns allowing a study of achievable
c     range using this steerable chute model
c
c   Only aoa and thrust level are controls during rev grav turn
c
c   This deck is a modified version of EMQ-grvtn-newref.inp
c   Includes most recent M2001 chute handoff conditions & ref (south site)
c   Includes more up to date engine
c   (from CPIA/M5 Liquid propellant engine manual - unit 187):
c       Viking lander(max throttle cond):
c           Isp=210sec, Thrust=632lbf, exit area=1.588in^2
c           Flow rate=3.10 lb/s., thrust coeff 1.53
c
c   Constraints:
c    At event 80 (critr = wr = 2.0) -
c    2550 < gdalt < 2850
c    -1.1 < ur   < 1.1
c    -1.1 < vr   < 1.1
c
c*************************************************************************
   ioflag   = 3,    / metric input, metric output
   ipro     = -1,   / trajectory print flag
c*************************************************************************
c Optimization variable - 5.b
c - general optimization inputs
   irscl    = 3,    / default value (flag)
   isens    = 0,    / forward finite differences, default value (flag)
   maxitr   = 50,   / maximum number of iteration
```

```
   opt    = -1,
   optph  = 80,
   optvar = 'dprng1',
   srchm  = 4,    / projected gradient
   wopt   = 1.0,  / weighting for optimization variable (default val)
c - projected gradient specific inputs, all defaults except coneps(2)
   coneps = 89.9, 4*1.0e-04,
   consex = 1.0e-6, 0.001,
   fiterr = 1.0e-6, 0.001,
   gamax  = 10,
   ideb   = 0,
   npad   = 9.0, 4.0, 14.4494,
   pctcc  = 0.3,
   pdlmax = 2.0,
   pgeps  = 1.0,
   prntpd = 0,
   p2min  = 1.0,
   stminp = 0.1, 0.1,
   stpmax = 1.0e+10,
c*************************************************************************
c Constraint variable - 5.c
c - general Dependent Variable  inputs
   depph  = 80,80,80,80,80,/80,80,
   depvr  = 'gdalt','ur','vr','ur','vr',/'long','gdlat',
   ifdeg  = 0,0,0,0,0,/1,1,
   indxd  = 1,2,3,4,5,/6,7,
   ndepv  = 5,/7,
c - projected gradient specific inputs
   deptl  = 150.0,0.1,0.1,0.1,0.1,/0.01688179,0.01688179, / target within 1km
   depval = 2700.0,1.0,1.0,-1.0,/-1.0,93.8023,-15.8384,
   idepvr = 0,1,1,-1,-1,/0,0,    / constraint types
c*************************************************************************
c Controls - 5.d
c - general Independent Control inputs
   indph(1) = 25,
   indph(2) = 26,26,
   indph(4) = 27,27,
   indph(6) = 50,50,
   indph(8) = 28,28,
   indph(10) = 29,29,
c
   indvr(1) = 'bnkpc1',
   indvr(2) = 'bnkpc1','critr',
   indvr(4) = 'bnkpc1','critr',
   indvr(6) = 'etapc1','alppc1',
   indvr(8) = 'bnkpc1','critr',
   indvr(10) = 'bnkpc1','critr',
c
   nindv  = 11,
   pert(1)  = 1.0e-4,
   pert(2)  = 1.0e-4,1.0e-4,
   pert(4)  = 1.0e-4,1.0e-4,
   pert(6)  = 1.0e-4,1.0e-4,
   pert(8)  = 1.0e-4,1.0e-4,
   pert(10)  = 1.0e-4,1.0e-4,
c
```

```
*include '../../bankangleguess.dat', / just covers u(1)-u(5), i.e. chute events
   u(6)    = 1.0,0.0,
   u(8)    = 5.0,5.0,
   u(10)    = -5.0,5.0,
c
c - projected gradient specific input
   modew   = 1,
 $
c*****************************************************************************
c    Trajectory Simulation Inputs
c*****************************************************************************
l$gendat
   title=0h*EMQ-liftchute-grvtn.inp*,
   event      = 1,0.0,     / first event number (primary event)
   fesn       = 500,       / final event number
c - NUMERICAL INTEGRATION METHODS p. 6.a.15-1
   npc(2) =1,  / integration method (using RK) [flag]
   dltmax =1,  / max step size when using variable steps [s]
   dltmin =0.05,  / min step size  "    "    "      "   [s]
   dt=1,/ integration time step [s]
   kstpmx=5,/ max # of integration steps for each integration
   npinc=5,/ # of integration steps on each cycle
c*****************************************************************************
c  Initial Event Conditions/Setup
c
c - INITIAL POSITION AND VELOCITY p. 6.a.12-1
   npc(3)=1,/ initial position in xi, yi, zi
   npc(4)=1,/ initial velocity in vxi, vyi, vzi
   npc(40)=3,/ reference plane for azimuth and FPA [flag] 3=?
c - ---- ---- ---- ---- ---- ----
c - initial conditions for batch runs, includes 'timeo' (via Matlab)
*include '../../pvstates.dat',
c - ---- ---- ---- ---- ---- ----
c
c - RANGE CALCULATIONS p. 6.a.19-1
   npc(12)=1,/ cross/down range option [flag] (3=?)
  / lonref=93.6484317,/ from Scott (new M2001 south site)
  / latrefgd=-15.8108183,
   lonref   = 93.8023,  / developed from looking at averages of various
   latrefgd = -15.8384,  / lifting chute non-optimized cases (showellipsebanks2.m)
c
c - PARACHUTE MODEL p. 6.a.28-1
   npc(32)=2,
   diamp(1)=0.0,/ init val of chute diam, unfurl to 13m
   drgpk(1)=1,
   idrgp(1)=0,
   parif(1)=70.0,
c
c - AERODYNAMIC INPUTS p. 6.a.1-1
   npc(8)=3,/aerodynamic coefficient [flag]
   sref=4.5238934,/aerodynamic reference area [m^2] (from M98)
c
c - AEROHEATING CALCULATIONS p. 6.a.2-1
   npc(15)=1,/ calculate aeroheating rate & tot. heat using Chapman
   npc(26)=0,/ no special aeroheating calculations
   rn= 0.6638,/ nose radius for Chapman heating (M98nom.inp)
```

```
c
c - ATMOSPHERE PARAMETERS p. 6.a.4-1
   npc(5)= 6,/ 2/3/02, 0 hr Marsgram atmosphere input as tables
   npc(6)= 2,/ Marsgram winds* Braun multipliers input as tables
   atmosk(1)=241.0,
   atmosk(2)=5.335e-03,
c
c - CONIC CALCULATION OPTION p. 6.a.5-1
   npc(1)= 3,/Keplerian conic option [flag]
   mre=1hu,/value of mean radius to be used [m](1hu = [re+rp]/2)
c
c - GRAVITY MODEL p. 6.a.10-1
   npc(16)=0,/ spherical or oblate model (oblate) [flag]
   j2=0.1958616e-02,/ spherical harmonics of gravity potential function
   j3=0.3144926e-04,
   j4=-0.1889437e-04,
   j5=0.2669248e-05,
   j6=-0.1340757e-05,
   j7=0.0d0,
   j8=0.0d0,
   re=3393940.0,/ equatorial radius [m]
   rp=3376780.0,/ polar radius [m]
   mu=4.28282868534e+13,/ gravitational constant (mars) [m^3/s^2]
   omega=7.088218e-05,/ rate of rotation of planet [rad/s]
   go=3.718,/ weight to mass factor (Mars surface)
c - VELOCITY LOSSES p. 6.a.25-1
   npc(25)=2,/ velocity loss calculation
c*****************************************************************************
c  Initial Guidance Inputs
c*****************************************************************************
   iguid(1)=0,0,1,/ atm.rel. aero angle guidance
   alppc(1)=180.0,/ initial alpha
   maxtim=2000.,/ maximum time
   altmax=550000.,/ maximum altitude
   altmin=-3000.0,/ minimum altitude
c*****************************************************************************
c Vehicle Model
c*****************************************************************************
   wgtsg=2176.811,/veh. wt. at parachute deploy, N (585.479 kg)
   wpropi=372.0,/initial propellant weight, N (100 kg)
   npc(30)=0,/enhanced (component) weight model
   npc(9)=1,/rocket engine
   npc(27) = 1, / integrate flow rate of specified engines
   npc(22)=2,/input all four coef's in throttling parameter
   neng=2,/2 engines
   ispv(1)=553.9,553.9,/ Mars Isp (Earth Isp = 210 sec)
   iwdf(1)=2,2,/flow rate = vac. thrust/ispv
   iwpf(1)= 0,0,
   iengmf(1)=0,0,/engine off initially
   iengt(1)=0,0,/fixed engine angles (in tables) w.r.t body
c*****************************************************************************
c Print Block
c - PRINT VARIABLE REQUESTS p. 6.a.16-1 --->>
   npvl=0,/ # of print variables per line [flag]
   pinc=10,/ print interval
   prnc=0,/ make profile for plotting
```

```
*include '../../prntblk.dat',  / printing variables
 $
l$tblmlt
 vwum = 1.0,
 vwvm = 1.0,
 $
l$tab
 table = 'denkt',1,'gdalt',3,1,1,1,
 0.0,1.0,30000,1.0,130000,1.0,
 $
l$tab
 table = 'prest',1,'gdalt',6,1,1,1,
 0,6.52553,1859,6.36383,4217,6.15610,6387,5.96265,8515,5.77084,
 8816,5.74359,
 $
l$tab
 table = 'atemt',1,'gdalt',6,1,1,1,
 0,226.1419,1859,223.766,4217,220.502,6387,217.854,8515,215.292,
 8816,214.936,
 $
l$tab
 table = 'vwut',1,'gdalt',6,1,1,1,
 0,-0.1248,1859,0.02672,4217,0.19915,6387,0.55803,8515,0.86006,
 8816,0.85601,
 $
l$tab
 table = 'vwvt',1,'gdalt',6,1,1,1,
 0,0.200574,1859,0.42458,4217,0.68911,6387,1.5575,8515,2.2659,
 8816,2.2508,
 $
l$tab
 table = 'vwwt',0,0.0,
 $
l$tab
 table = 'tvc1t',0,2646.0,
 $
l$tab
 table = 'tvc2t',0,2646.0,
 $
l$tab
 table = 'ae1t',0,0.001,
 $
l$tab
 table = 'ae2t',0,0.001,
 $
l$tab
 table = 'pi1t',0,180.0,  / engine #1 gimbal pitch angle
 $
l$tab
 table = 'yi1t',0,0.0,    / engine #1 gimbal yaw angle
 $
l$tab
 table = 'pi2t',0,0.0,    / engine #2 gimbal pitch angle
 $
l$tab
 table = 'yi2t',0,0.0,  / engine #2 gimbal yaw angle
```

```
 $
l$tab
 table = 'wd1t',0,12.98, / engine #1 flow rate
 $
l$tab
 table = 'wd2t',0,12.98,   / engine #2 flow rate
 $
l$tab
 table='cdt',0,1.7,
 $
l$tab
 table='clt',0,0.0,
 $
l$tab
 table='cdp1t',0,0.41,
 endphs=1,
 $
c
c Parachute fully deployed
l$gendat
 event=22.,0.0, / primary event
 critr='diamp1',
 value=13.0,
 parif(1)=0.0,
 wgtsg = 1937.297,
 endphs=1,
 $
c - - - Convert parachute to 'airplane', simulates lifting/steerable chute
l$gendat
  event=25.,0.0, /primary event
  critr='tdurp',
  value=0.0,
  sref=132.73, / surface area of parachute about 13m (pi*r^2) incorporated into lander
  npc(32)=0,     / don't calculate parachute drag
  iguid(1)=0,    / aero-guidance
  iguid(2)=1,    / individual component steering
  iguid(6)=0,    / alpha carried over
  iguid(7)=0,    / beta carried over
  iguid(8)=1,    / bnkang input (from targeting algorithm)
 $
l$tblmlt
 $
l$tab
 table='cdt',0,0.41,
 $
l$tab
 table='clt',0,.41, /lift coeff includes parachute, i.e. =1*0.41
 endphs = 1,
 $
c - - 2nd chance to steer 'airplane'
l$gendat
  event=26.,
  critr='tdurp',
  endphs = 1,
 $
c - - 3rd chance to steer 'airplane'
```

```
l$gendat
 event=27.,
 critr='tdurp',
 endphs = 1,
 $
c - - 4th chance to steer 'airplane'
l$gendat
 event=28.,
 critr='tdurp',
 endphs = 1,
 $
c - - 5th chance to steer 'airplane'
l$gendat
 event=29.,
 critr='tdurp',
 endphs = 1,
 $
c - - - jetison parachute; turn on engine #1 (start of reverse gravity turn)
l$gendat
 event = 50,0.0,
 critr = 'gdalt', / roving event (in case event 26 never happens)
 value = 3500.0,
 npc(32)=0,              / jettison parachute
 diamp(1)=0,
 wjett = 276.702,        / weight of parachute lost (N)
 sref=2.0,              / new surface area
 iengmf(1) = 1,0,
 iwpf(1) = 1,0,
 iguid(1) = 0,     / aero-guidance
 iguid(2) = 1,     / individual component steering
 iguid(6) = 1,     / alpha input by targeting algo
 iguid(7) = 0,     / beta carried over
 iguid(8) = 0,     / bnkang carried over
 $
l$tblmlt
 $
l$tab
 table='cdt',0,2.0,
 $
l$tab
 table='clt',0,0.0,
 $
l$tab
 table='cdp1t',0,0.0,
 endphs = 1,
 $
cc - marks 2500 meter mark (Martian surface)
l$gendat
 event=80,0.0, / primary event - this must happen or targeting failed
 critr='wr',
 value = 2.0,
 iengmf(1) =  0,0,
 iwpf(1) = 0,0,
 endphs=1,
 $
cc
```

```
c
c*************************************************************************
c    This event marks arrival at the  Martian surface
c*************************************************************************
l$gendat
 event=500,0.0, / primary event
 critr='tdurp',
 value=0,
 endphs=1,
 endjob=1,
 endprb=1,
 $
```

Complete list of all files modified/used during grant cycle

TreePrint listing of: C:\school\NASA\Docs\NASAfiles2001to2002

```
C:\school\NASA\Docs\NASAfiles2001to2002
|| treeprint.txt
||
+--dat
| | bkb3guess.dat
| | bkb3guess2.dat
| | chutestates.dat
| | chutestates2.dat
| | fulliris.txt
| | hoverinitcond.dat
| | prntblk.bak
| | prntblk.dat
| | pvstates.dat
| |
+--docs
| | List of all papers for PHD research.doc
| | PHD TIMELINE.doc
| | Progress Update 1.doc
| | Proposal sum2001.doc
| | Research Update.doc
| | summer.txt
| | Thesistemplate1.doc
| |
+--Mat
| || bc4batch.mat
| || chutestatplot.mat
| || copy_of_bc1871.mat
| || copy_of_lc.mat
| || copy_of_lc1747_18.mat
| || copy_of_m2001newbase.mat
| || copy_of_rwp1rps.mat
| || forward.mat
| || forward3.mat
| || inverse.mat
| || IrisWtsLM.mat
| || lc4batch.mat
| || lcsaveas.mat
| || nettrain-76init-bc17baseline.mat
| || nettrain-76init-bc17baselineMOD.mat
```

```
| || nettrain-nominit-bc17baseline.mat
| || nettrain-nominit-bc17baselineMOD.mat
| || nettrain1.mat
| || nettrain2.mat
| || nettrain3.mat
| || nettrainer-weights.mat
| || nettrainer-wts-1layer.mat
| || nettrainer2-wts-1layer-inertialBPX.mat
| || nettrainer2-wts-1layer-inertiallm.mat
| || nettrainer2-wts-1layer-inertialPSO.mat
| || nettrainer2-wts-2layers-unmodified-inertial-lm.mat
| || pathdat1.mat
| || pickstates4.mat
| || pickstates5.mat
| || weightsf6.mat
| || weightsxor.mat
| ||
| \--oldMAT
|    | allptsCHUTEno25.mat
|    | lc4batch.mat
|    |
+--outs
| | copy_of_lc.out
| | copy_of_m2001newbase.out
| | lc4batch.out
| | lcsaveas.out
| |
| \--oldout
|    | EMQ-grvtn-newref.out
|    | hover30deg.out
|    | m2001newbase-liftchute-noopt-bnk0.out
|    | m2001newbase-liftchute-noopt-bnk25.out
|    | m2001newbase-liftchute-noopt-bnkneg25.out
|    | m2001newbase-liftchute-noopt-bnkstudy.out
|    | m2001newbase-liftchute15LD-noopt-bnkstudy.out
|    | m2001newbase-liftchute1LD-noopt-bnkstudy.out
|    | m2001newbase-liftchute2.5LD-noopt-bnkstudy.out
|    | m2001newbase-liftchute25LD-noopt-bnkstudy.out
|    | m2001newbase-liftchute2LD-noopt-bnkstudy.out
|    | m2001newbase-liftchute75LD-noopt-bnkstudy.out
|    | m2001newbase-thrustchute-noopt-bnkstudy.out
|    | m2001newbase-thrustchute12kg-noopt-bnkstudy.out
|    | m2001newbase.out
|    | m2001newbase2-liftchute-noopt-bnkstudy.out
|    | m2001newbase2-liftchute-targref.out
|    | m2001newbase2-liftchute.out
|    | m2001newbase2-liftchute15LD-noopt-bnkstudy.out
|    | m2001newbase2-liftchute15LD-targref.out
|    | m2001newbase2-liftchute15LD.out
|    | m2001newbase2-liftchute1LD-noopt-bnkstudy.out
|    | m2001newbase2-liftchute1LD-targref.out
|    | m2001newbase2-liftchute1LD.out
|    | m2001newbase2-liftchute25LD-noopt-bnkstudy.out
|    | m2001newbase2-liftchute25LD-targref.out
|    | m2001newbase2-liftchute25LD.out
|    | m2001newbase2-liftchute2LD-noopt-bnkstudy.out
```

```
|   |  m2001newbase2-liftchute2LD-targref.out
|   |  m2001newbase2-liftchute2LD.out
|   |  m2001newbase2-liftchute75LD-noopt-bnkstudy.out
|   |  m2001newbase2-liftchute75LD-targref.out
|   |  m2001newbase2-liftchute75LD.out
|   |  m2001newbase2-thrustchute-noopt-bnkstudy.out
|   |  m2001newbase2-thrustchute.out
|   |  m2001newbase2-thrustchute12kg-noopt-bnkstudy.out
|   |  m2001newbase2-thrustchute12kg.out
|   |  m2001newbase2.out
|   |  tstnet.out
|   |
+--post3d_source
|   aero.f
|   aero.o
|   balland.f
|   balland.o
|   calspe.f
|   calspe.o
|   guidance.f
|   guidance.o
|   Makefile
|   Makefile.log
|   Makefile_calspe
|   Makefile_calspe_new
|   Makefile_calspe_old
|   Makefile_dbg
|   Makefile_dbg_calspe
|   mars_entry5a.inp
|   master.o
|   p3d
|   p3d_calspe
|   p3d_dbg
|   p3d_dbg_calspe
|   phzxm.o
|
\--support
    chutestatplot.m
    demonormalize.m
    extho.m
    f6.m
    f6pso.m
    learnga.m
    learnlm.m
    load_DATA_mat.m
    m2001viewsite.m
    MatPathData.m
    nettrainer.m
    nettrainer2.m
    normalize.m
    p2check.m
    p3dbatchSPACE2000.m
    p3dbatchspace2000PSO.m
    p3dSPACE2000.m
    p3dSPACE2000b.m
    pso.m
```

startup.m
tbpx1.m
tbpx2.m
tbpx3.m
testf6.m
testnettrainer.m
testpso.m
tga1.m
tga2.m
tlm1.m
tlm2.m
tlm3.m
tpso1.m
tpso2.m
tpso2mod.m
tpso2old.m
tpso3.m
trainbpx.m
trainga.m
trainlm.m
trainpso.m
tstmov.m
unwrapmat.m
wrapmat.m

Iris Data Set

The Iris data set used here was downloaded from a website after a google search for the words "Iris Data Set'. It is presented here in the exact downloaded form:

Species	Sepal length	Sepal width	Petal length	Petal width
1	49	30	14	2
1	51	38	19	4
1	52	41	15	1
1	54	34	15	4
1	50	36	14	2
1	57	44	15	4
1	46	32	14	2
1	50	34	16	4
1	51	35	14	2
1	49	31	15	2
1	50	34	15	2
1	58	40	12	2
1	43	30	11	1
1	50	32	12	2
1	50	30	16	2
1	48	34	19	2
1	51	38	16	2
1	48	30	14	3
1	55	42	14	2
1	44	30	13	2
1	54	39	17	4
1	48	34	16	2
1	51	35	14	3
1	52	35	15	2

1	51	37	15	4
1	54	34	17	2
1	51	38	15	3
1	57	38	17	3
1	45	23	13	3
1	48	30	14	1
1	53	37	15	2
1	44	29	14	2
1	54	39	13	4
1	54	37	15	2
1	49	31	15	1
1	50	35	13	3
1	51	34	15	2
1	46	31	15	2
1	47	32	13	2
1	47	32	16	2
1	50	33	14	2
1	50	35	16	6
1	55	35	13	2
1	46	34	14	3
1	51	33	17	5
1	52	34	14	2
1	49	36	14	1
1	48	31	16	2
1	46	36	10	2
1	44	32	13	2
2	66	29	46	13
2	61	29	47	14
2	60	34	45	16
2	52	27	39	14
2	49	24	33	10
2	60	27	51	16
2	56	27	42	13
2	61	30	46	14
2	55	24	37	10
2	57	30	42	12
2	63	33	47	16
2	69	31	49	15
2	57	28	45	13
2	61	28	47	12
2	64	29	43	13
2	63	23	44	13
2	60	22	40	10
2	56	30	41	13
2	63	25	49	15
2	50	20	35	10
2	59	30	42	15
2	55	25	40	13
2	62	29	43	13
2	51	25	30	11
2	57	28	41	13
2	58	27	39	12
2	56	29	36	13
2	67	31	47	15
2	67	31	44	14
2	55	24	38	11
2	56	30	45	15

2	61	28	40	13
2	50	23	33	10
2	55	26	44	12
2	64	32	45	15
2	55	23	40	13
2	66	30	44	14
2	68	28	48	14
2	58	27	41	10
2	54	30	45	15
2	56	25	39	11
2	62	22	45	15
2	65	28	46	15
2	58	26	40	12
2	57	29	42	13
2	59	32	48	18
2	70	32	47	14
2	60	29	45	15
2	57	26	35	10
2	67	30	50	17
3	63	33	60	25
3	65	30	52	20
3	58	28	51	24
3	68	30	55	21
3	67	31	56	24
3	63	28	51	15
3	69	31	51	23
3	64	27	53	19
3	69	31	54	21
3	72	36	61	25
3	57	25	50	20
3	65	32	51	20
3	65	30	58	22
3	62	34	54	23
3	64	28	56	21
3	61	26	56	14
3	64	28	56	22
3	77	30	61	23
3	67	30	52	23
3	62	28	48	18
3	59	30	51	18
3	63	25	50	19
3	72	30	58	16
3	76	30	66	21
3	64	32	53	23
3	61	30	49	18
3	79	38	64	20
3	72	32	60	18
3	63	27	49	18
3	77	28	67	20
3	58	27	51	19
3	67	25	58	18
3	49	25	45	17
3	67	33	57	21
3	77	38	67	22
3	56	28	49	20
3	65	30	55	18
3	58	27	51	19

3	74	28	61	19
3	69	32	57	23
3	68	32	59	23
3	73	29	63	18
3	71	30	59	21
3	60	22	50	15
3	77	26	69	23
3	67	33	57	25
3	63	29	56	18
3	60	30	48	18
3	64	31	55	18
3	63	34	56	24