Predicting Individual Overall Placement in Collegiate Waterski Tournaments using a Neural Network

Charles Rodenkirch

University of Wisconsin – Madison

ECE 539: Introduction to Artificial Neural Networks and Fuzzy Logic
ABSTRACT

I am currently president of the UW-Madison Waterski and Wakeboard Team. Collegiate waterski tournaments consist of three events (Jump, Slalom, and Trick) for which each team submits 5 men and 5 women competitors. Each water skier is given final overall ranking at each tournament. This is often considered the ultimate classifier of water skier skill. However due to the complexities in determining final ranking in tournaments it is very hard for a water skier to predict how they will place before the tournament concludes. This is because there is no absolute scale for final ranking. In each event skiers are scored based on a standardized point guide [1], these scores are then ranked against other competitor’s scores in the same event and then the water skier is given a final point value for that event based on how their scores rank against the other competing water skiers scores. The awarded points from each event are then summed to find a final composite score which is used for the overall placement. This means that any water skier’s overall placement depends on how well other water skiers perform which is often hard to predict, as each competitor only gets one opportunity in each event (jump, trick, and slalom). Water skiers often know what scores they hope to receive in each event due to their performance in the week beforehand; however, any attempts to estimate what their resulting final ranking would be in the tournament based on these scores is nontrivial due to having to predict how the other water skiers will perform in that tournament. I would like to investigate using a multilayer perceptron as a tool to predict final skier ranking in a tournament based on prediction criteria a competitor would know in the weeks leading up to the competition. Historical
tournament results and archived weather data will be used to train the multilayer perceptron.

PROBLEM STATEMENT

It is extremely difficult, if not impossible, for a collegiate water skier to estimate what his final ranking will be in an upcoming tournament. A water skier can certainly predict his scores in each individual event with decent accuracy. However, the final ranking is not based directly off these scores and is instead a measurement of how well you performed compared to the other competitors in each event. The water skier, who is attempting to predict what their final ranking for the weekend will be, then not only has to predict their own scores, but also the scores of all the other competing water skiers. This is further complicated by the fact that although each tournament has a set number of competitors per team, the number of teams per tournament can still vary widely, resulting in different pools of competitors at each tournament. Although historical scores of all competitors are available for tournaments [2] and the skill level and competitor number at each tournament tends to be similar year to year, the data is just too large and complex for a water skier to easily process. Due to being presented with too much information for an individual to predict accurately, most water skiers are unable to comment on what place they expect to receive in an upcoming competition.
MOTIVATION

Overall ranking in a tournament is considered the best classification of a water skier’s skill level, and is therefore of great interest to any water skier. Placing in the top ten at any tournament would be considered quite exceptional, and is a common goal for collegiate water skiers. However, it is very hard for a water skier to determine what scores he needs in each event to reach this top echelon of water skiers in the upcoming tournament. Therefore while training for the upcoming tournament, a water skier can only guess at which of the three events to concentrate their training efforts. They must then hope that improving that certain event will have the greatest effect on their chance at receiving a higher placement in the tournament. However, if the water skier possessed a prediction system to which they could feed in certain data that was available to them before the tournament, such as their predicted scores in each event, the number of competitors, etc., they could determine their predicted overall ranking based on their scores. This would allow the water skier to test hypothetical event scores based on potential improvements that they could obtain during the week of training preceding the tournament. Then by comparing the predicted placements based off of their hypothetical event scores it would be easy for the water skier to determine how an improvement in each event would affect their overall placement at the upcoming tournament. They can then focus their training on the event whose improved score would give them the best chance at increasing their overall tournament ranking. This selective training could certainly give them an edge over the rest of the competition.
BACKGROUND

Collegiate Water Ski Tournaments

Collegiate waterskiing is governed by the national collegiate waterski association [3] which is a subsidiary of USA Waterski [4], the governing body for all amateur and professional waterski tournaments in the United States. Therefore collegiate waterski tournaments follow very similar tournament rules compared to USA Water Ski’s tournaments, and the slight differences can be found in the national collegiate waterski association’s official tournament rules [1]. These differences do not effect this prediction system, allowing for its future expansion of use beyond collegiate tournaments.

Collegiate water skiers are considered, “three event skiers”, which should not be confused with “show skiers.” In a show ski tournament, a team comprised of individual show skiers performs a show, and the entire team is judged based on criteria that estimate how entertaining their presented show was. Teams are then ranked accordingly and no individual scores are given. In contrast, three event tournaments are much more akin to a conventional track meet. Each collegiate team enters their top five female and top five male water skiers into the tournament. Each water skier then individually competes in three separate events: slalom, trick, and jump. The water skier’s performance in each event is then compared to the other competitors performances in that event, and based on the results of this comparison a water skier’s team is awarded a set amount of points for that water skier’s performance. At the end of the tournament, not only does each team get awarded an overall ranking compared to the other teams, but each water skier gets an overall ranking compared to the other
individual competitors. This individual overall ranking is extremely important as it is not only a classifier of water skier skill, but also a predictor of how many points you provided your team in the competition.

Every collegiate ski tournament takes place over the course of a weekend, with events occurring on Saturday and Sunday. Traditionally, for male competitors, the Slalom competition occurs on Saturday morning, Trick takes place on Saturday night, and the final event, Jump, happens on Sunday morning. There are two seasons of collegiate competition, the spring semester and the fall semester, with ski tournaments ceasing during the winter months due to cold weather.

The slalom skiing competition is held within a slalom course consisting of 6 buoys on alternating sides of the boat path with an entrance gate and exit gate at their respective ends of the course. A tournament approved towboat drives through the entrance gate, down the center of the course, and leaves through the exits gate all while towing the skier behind. The skier must round the outside of each buoy as he travels through the course. The easiest pass consists of a water skier being towed by a full length rope at 19 mph. After each successful pass the boat speed is increased up to a max speed of 36 mph causing the course to be more difficult to run. After running a 36 mph pass, the water skier’s rope is then shortened after each successful run; again make each pass more difficult. A skier’s final score in the Slalom competition is the number of buoys he can round before falling or missing a buoy during a pass.

The trick competition is simpler in comparison. A skier is towed behind a boat through a trick course which consists of a starting buoy and end buoy. Once the water
skier passes the starting buoy, they have 20 seconds to perform as many tricks as they can before the time runs out or they fall into the water. Specific tricks and their resulting point values are designated by USA Waterski [5] and each unique trick adds points to the water skiers total score in this competition.

The jump competition is based purely off the distance the skier travels in the air. A skier is pulled towards a five foot jump by a towboat and must cut towards the jump on their approach, with a harder cut towards the jump increases the skiers speed resulting in them traveling a further distance. Distance is measured in feet and reported by a camera scoring system. Each water skier receives three attempts at the jump with their longest distance being used as their final score.

There are many factors that affect a skier’s performance in a tournament; however, as skiing is an outdoor event the largest factor is that weekend’s weather. Specifically three features of the weather can make skiing more difficult. Windy days are considered a water skier’s worst fear. The most visual effect of high winds is seen on the surface of the water. A water skier performs best in water conditions commonly referred to as “glass”, during which the surface of the water is perfectly flat, allowing for a consistent resistance to the ski. High winds cause the water surface to become choppy and unpredictable. These winds also exert a force on the skier’s body, and due to their high rate of speed this force can be considerable. Precipitation and temperature also effect skiers as both factors can lead to unfavorable conditions to be traveling through water at high speeds in only a swimsuit and competition vest.
Predicting Sporting Events Using Neural Networks

There have already been quite a few papers on using neural networks to predict the outcome of sporting events, so there is a clear precedent that this is an effective method. In 1999 researchers at the University of Maryland used a neural network to predict the success of nations at the upcoming Summer Olympics [6]. Their data found that the best neural network model they could create outperformed the best regression model, indicating neural networks as a good prediction tool. Another set of researchers at the Air Force Institute of Technology was able to successfully predict NBA games using neural networks [7]. These previous papers support the idea of predicting water ski tournaments in the same way.

K Nearest Neighbor Clustering

The k-nearest neighbor algorithm is a method for classification and regression commonly employed in pattern recognitions [8]. A program employing the method can be fed data that does not have any characteristic structure or parameters. The program then processes the data in its feature space and is able to determine classification categories based on cluster centers that it defines. Then new data is classified based on the classification of its nearest neighbors, hence the name of the algorithm.

Multi-Layer Perceptron

A multi-layer perceptron is a type of feed-forward neural network of threshold units [9]. Multi-layer perceptrons are composed of an input layer of neurons, successive layers of intermediate units, and a layer of output neurons. The output of each layer is connected to the input of the next layer. A synaptic weight is associated with the each
unique connection between neurons in neighboring layers. Each neuron itself is associated with a hyper plane, and classifies its input based on which side of the hyper plane the input falls, this classification is then passed on to neurons in the next layer. To be used for classification, the weights and activation functions of each neuron must be calibrated so that when feature vectors are inputted to the input layer of neurons, the correct classification vector is outputted from the output neurons.

**Back Propagation**

Back propagation is a commonly used method for training neural networks. It works off the premise that if shown multiple examples and given these examples classifications a neural network can learn what features correspond with which classification. When the neural network is then presented a new example and not given its classification, the neural network should be able to classify it based on its previously learned patterns [10]. This is akin to how any individual can easily classify a newly met individual as a child, even though he/she has only seen other examples of children and never met that specific child before. Back propagation trains a feed forward neural network, by passing it a feature vector, and comparing the networks output with the known correct output. Then based on the discrepancies it finds, it is able to update the weights within the neural network to more properly reflect the correct classification. Back propagation is deemed a supervised learning method, as while the network is learning it is able to check its output against the correct given classification [10].
DATA GATHERING AND PROCESSING

The general idea behind this prediction program is that a water skier could enter in predictions about an upcoming tournament and then receive his predicted placement. Therefore it was important to not use any data that could not be determined until after a tournaments completion. In regards to the individual skiers performance, their trick score in points, their jump score in feet, and their slalom score in buoy count were used as feature vectors. A skier could predict these values based on his performance in preceding practice sessions. In regards to the tournament, the number of competitors and season of competition (spring/fall) were used. These values could be easily determined after the registration period of the tournament and before the tournament takes place. In regards to weather conditions, Saturday and Sunday’s wind speed in miles per hour, precipitation in inches, and temperature where used. This data could be predicted using any online weather forecast utility.

The data from previous tournaments that links event scores with overall placement was pulled from the archived scorebooks of the national collegiate water ski association. For the scope of this project, male competitors’ scores for all tournaments in the Midwest conference from the last three years of tournaments were compiled. As this data is not organized within the scorebooks to be easily processed, this data accumulation was an arduous process. The Matlab file “ImportData.m” was written to facilitate this process. Each event, trick, slalom, and jump, has its own scorebook. There is also a fourth scorebook for overall placement. As it was necessary to link each skiers score in each event to his final placement, the data from these four scorebooks needed to be combined. “ImportData.m” takes in data from four different text documents, each
containing one of the four scorebooks. It then processes this data and outputs a matrix with each row containing the skiers overall placement, trick score, jump score, and slalom score. This process was repeated for each tournament. The data separated by each tournament can be viewed in the Excel file “Rough Data.”

Next the weather conditions for both Saturday and Sunday for each tournament were found by searching through an archive of historical weather conditions [11]. The weather data from each tournament was then added to the scores from that tournament, completing each feature vectors. All feature vectors were then combined into a mass listing of 401 unique feature vectors which can be viewed in the excel document “Combined Data.” A statistical analysis of each feature vector was performed and can be found in the appendix under Figure A. Finally the feature vectors where given labels based on their overall ranking. This resulted in the data that will be fed into the multi layer perceptron and can be viewed in the Excel file “Feature Vectors with Labels.”

**Clustering Classification of Scores**

It was hypothesized that by classifying the scores in each event using a k nearest neighbors classifier before feeding it into the multi layer perceptron, that the prediction accuracy of the multi-layer perceptron could be improved. To decide how many clusters to use for each event, trick, jump, and slalom, the matlab program “WaterskiCluster.m” was employed. This program calls the program “kmeansWaterski.m” to perform cluster classification on data that was inputted. Cluster values from 2 through 40 were tested on for each event and the resulting data was compiled in the excel file “Clustering Event
Scores.” This data was then graphed and analyzed. The resulting graphs can be viewed in the appendix Figure B. The final distortion values for each number of clusters were compared, and the number of clusters that produced the lowest distortion value was chosen. This value was 9 cluster points for trick and jump, and 7 cluster points for slalom. Once the optimal cluster point values were determined, the corresponding weights of these cluster centers were processed using the program “clusterToRanges.m” which was written to intake a set of cluster coordinates and output ranges of which each ranges center is a cluster point. The output of this code can be viewed in appendix Figure C. Finally the scores for all feature vectors for each event where encoded using the program “encodeSkiScores.m” which takes a vector of range points and a vector containing the ski scores themselves, and outputs a vector of the corresponding classification label for each ski score. After encoding all of the scores, the feature vectors were again compiled and can be viewed in the Excel file “Feature Vectors with Labels Scores Encoded.”

TESTING

Yue Hu Hen’s code for a back propogation multil layer perceptron was used as a basis for the construction of the testing program. The modified code used in this paper was saved as “bpMultiConfigTest” and the configuration function called by this program was edited and saved as “bpconfigWaterski”. The program bpMultiConfigTest takes in all available feature vectors. For each trial it runs, it randomly partitions the feature vectors into a testing and training set. The training set is then partitioned into training and tuning set. The program then runs trials for every unique combination of neuron layers (2 through 10), number of neurons per hidden layer (1 through 5), activation
functions of hidden layers (Sigmoidal, hyperbolic tangent, and linear), learning factor
(0.1 through 0.9 in 0.1 increments), and momentum (0.1 through 0.9 in 0.1 increments).
It then outputs the classification rate of the resulting 12,149 unique trials.

The program was ran first using data which had not been encoded using the
clustering method, and the results can be viewed in the excel file “Classification
Performance Unsorted Data.” Then the program was ran again with the data that had
been encoded using the clustering method and the results for this trial can be viewed in
the excel file “Classification Performance Clustered Data.” With both methods, many of
the trials resulted in extremely high classification rates. Figure D in the appendix shows
the statistical analysis of the configurations with only extremely high classifier rates for
both the encoded and nonencoded data. The number of occurrences for each varied
parameter within results that had great classification rates was tallied. The values with
high occurrences were deemed quality parameters for use in the final configuration.

FINAL RESULTS

For the final configuration, it was chosen to use non-encoded data. This was due
to the fact that the non-encoded data had more trials with an extremely high
classification rate than did the encoded data. Based off the data in Figure D, the
following parameters where chosen. The network consists of an input layer, a hidden
layer with 4 neurons that uses a linear activation function and an output layer that uses
a sigmoidal activation function. The learning rate was set to 0.4 and momentum was set
to 0.7.
The final application, “FinalWaterskiPredictionApp,” asks the user to input all necessary data to complete a feature vector. It then trains itself using the entire set of feature vectors with labels. After training it tests the feature vector created from the users input and display the predicted range that he will place in the upcoming tournament.

FUTURE WORK

If more time and effort were to be expended on this project, the first order of business would be to collect all the available tournament data. Working individually I was only able to gather and process the tournament data for male competitors spanning the last 3 years of collegiate ski tournaments taking place in the Midwest conference. However, there is over 10 years of recorded ski tournament data for each region in the nation. Having access to this complete database would allow for a much better trained neural network prediction system.

Another issue that could be targeted is the programs calculation of classification rate. I had not edited this function from the original code. However, I believe that it may only be accounting for true positives while not accounting for false positives. I believe this due to the fact that the final system often outputs classifications were every value in the vector is 1, whereas the system should in fact output classification vectors where only one value in the vector is 1. If the classification function could be improved, the original tests could be re ran and a better configuration could be determined.
REFERENCES

2. http://www.usawaterski.org/rankings/view-tournamentsHQ.asp?sl=on&tr=on&ju=on&sTourLevel=Collegiate&sTourRange=5 – An archive containing collegiate waterski tournament records
3. www.NCWSA.com – National Collegiate Waterskiing Association
5. Trick Descriptions and Points Values, USA Waterski
11. http://weather.org/weatherorg_records_and_averages.htm - an easy to use online application that provides historical weather data for any given city and date

APPENDIX

Figures

- Figure A – Feature Vectors Statistical Analysis

<table>
<thead>
<tr>
<th></th>
<th>Placement</th>
<th>Jump Score</th>
<th>Trick Score</th>
<th>Slalom Score</th>
<th># Competitors</th>
<th>Precip Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>45</td>
<td>156</td>
<td>2730</td>
<td>104</td>
<td>45</td>
<td>2.64</td>
</tr>
<tr>
<td>Range</td>
<td>44</td>
<td>156</td>
<td>2730</td>
<td>104</td>
<td>40</td>
<td>2.64</td>
</tr>
<tr>
<td>Average</td>
<td>14.05721333</td>
<td>54.95522388</td>
<td>376.1567164</td>
<td>23.575249</td>
<td>27.3955224</td>
<td>0.102089552</td>
</tr>
<tr>
<td>Variance</td>
<td>108.3136121</td>
<td>1177.505453</td>
<td>163440.5152</td>
<td>1031.661</td>
<td>138.8759</td>
<td>0.23313096</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>10.43645532</td>
<td>34.3147394</td>
<td>404.2777639</td>
<td>32.11936</td>
<td>11.7845618</td>
<td>0.482807515</td>
</tr>
</tbody>
</table>
### Figure B – Determining Optimal Number of Clusters

- **Trick Scores Clustering Results**

![Trick Scores Clustering Results](image1)

- **Jump Scores Clustering Results**

![Jump Scores Clustering Results](image2)

- **Slalom Scores Clustering Results**

![Slalom Scores Clustering Results](image3)
- Figure C – Converting Cluster Centers to Classification Ranges
  - Trick

<table>
<thead>
<tr>
<th>Cluster Centers (Trick Points)</th>
<th>Corresponding Range Points</th>
<th>Corresponding Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>53.71</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>200.83</td>
<td>127.27</td>
<td>1</td>
</tr>
<tr>
<td>348.64</td>
<td>274.735</td>
<td>2</td>
</tr>
<tr>
<td>525.95</td>
<td>437.295</td>
<td>3</td>
</tr>
<tr>
<td>656.67</td>
<td>591.31</td>
<td>4</td>
</tr>
<tr>
<td>796.06</td>
<td>726.365</td>
<td>5</td>
</tr>
<tr>
<td>1022.31</td>
<td>909.185</td>
<td>6</td>
</tr>
<tr>
<td>1428.00</td>
<td>1225.155</td>
<td>7</td>
</tr>
<tr>
<td>2151.43</td>
<td>1789.715</td>
<td>8</td>
</tr>
</tbody>
</table>

- Jump

<table>
<thead>
<tr>
<th>Cluster Centers (Jump Distances)</th>
<th>Corresponding Range Points</th>
<th>Corresponding Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>42.25</td>
<td>21.125</td>
<td>1</td>
</tr>
<tr>
<td>56.56</td>
<td>49.405</td>
<td>2</td>
</tr>
<tr>
<td>68.14</td>
<td>62.35</td>
<td>3</td>
</tr>
<tr>
<td>80.95</td>
<td>74.545</td>
<td>4</td>
</tr>
<tr>
<td>93.89</td>
<td>87.42</td>
<td>5</td>
</tr>
<tr>
<td>105.33</td>
<td>99.61</td>
<td>6</td>
</tr>
<tr>
<td>120.67</td>
<td>113</td>
<td>7</td>
</tr>
<tr>
<td>142.70</td>
<td>121.085</td>
<td>8</td>
</tr>
</tbody>
</table>

- Slalom
- **Figure D – Classification Results**
  - **Non-Encoded Data**
    - | # Layers Including Output | # of Occurrences | # of Neurons per Hidden Layer | # of Occurrences |
      |---------------------------|------------------|------------------------------|-----------------|
      | 1                         | 667              | 1                            | 192             |
      | 2                         | 209              | 2                            | 207             |
      | 3                         | 60               | 3                            | 217             |
      | 4                         | 40               | 4                            | 237             |
      | 5                         | 19               | 5                            | 235             |
      | 6                         | 25               |                              |                 |
      | 7                         | 12               |                              |                 |
      | 8                         | 21               |                              |                 |
      | 9                         | 18               |                              |                 |
      | 10                        | 17               |                              |                 |
  - **Encoded Data**
    - | Activation Function | # of Occurrences | Learning Rate | # of Occurrences | Momentum | # of Occurrences |
      |---------------------|------------------|---------------|------------------|----------|-----------------|
      | Sigmoid             | 227              | 0.1           | 65               | 0.1      | 71              |
      | Hyperbolic Tangent  | 214              | 0.2           | 83               | 0.2      | 82              |
      | Linear              | 647              | 0.3           | 102              | 0.3      | 95              |
      |                     |                  | 0.4           | 120              | 0.4      | 110             |
      |                     |                  | 0.5           | 120              | 0.5      | 111             |
      |                     |                  | 0.6           | 147              | 0.6      | 129             |
      |                     |                  | 0.7           | 144              | 0.7      | 142             |
      |                     |                  | 0.8           | 146              | 0.8      | 154             |
      |                     |                  | 0.9           | 161              | 0.9      | 194             |
Matlab Code

CODE CREATED SOLEY BY CHARLES RODENKIRCH

clusterToRanges.m

% clusterToRanges - takes in cluster points, returns ranges with cluster
% points at the center between each range point given
% (C) copyright 2013 by Charles Rodenkirch

clusters = importdata('clusters.txt');
min = 0;
max = 3000;

for n = 1:(size(clusters)+1)
    if(n == 1)
        range(1) = min;
    elseif (n > (size(clusters)))
        range(n) = max;
    else
        center = clusters(n-1) + ((clusters(n)-clusters(n-1))/2);
        range(n) = center;
    end
end

disp(range);
encodeSkiScores.m

% takes in a column vector of ski scores and a column vector of range points then determines which range the ski score falls in and puts the label for this range into another column vector at the same position of the ski score in its original column vector
% Copyright Charles Rodenkirch 2013

clear all, close all

rangePoints = importdata('rangePoints.txt');
skiScores = importdata('skiScores.txt');

labels(1) = 0;

for n = 1:size(skiScores)
    for t = size(rangePoints):-1:1
        disp(t);
        if (skiScores(n) <= rangePoints(t))
            labels(n,1) = t-1;
        end
    end
end

disp(labels);

ImportData.m

overall = importdata('Overall.txt', '   ');
jump = importdata('Jump.txt', '   ');
trick = importdata('Trick.txt', '   ');
slalom = importdata('Slalom.txt', '   ');
results = zeros(200,4);

[m,n] = size(overall);
z = 0;
i = 0;

for x = 1:m
    z = 0;
i = 1;
    results(x,1) = overall(x,1);
    while z == 0;
        if (overall(x,4)== jump(i,2))
            results (x,2) = jump(i,1);
            z = 1;
        end
    i = i+1;
end
i = 1;
z = 0;
while z == 0;
    if (overall(x,3)== trick(i,2))
results (x,3) = trick(i,1);
z = 1;
end
i = i+1;
end
i = 1;
z = 0;
while z == 0;
    if (overall(x,2)== slalom(i,2))
        results (x,4) = slalom(i,1);
z = 1;
    end
    i = i+1;
end
end

% bpconfigWaterski.m
% configure the MLP. Called by bp.m
% mfiles used: partunef.m, scale.m
% Allow user to set up various parameters.
% Time saving procedure: save the parameters into a file, and load the same
% file next time if no parameter changes are needed.
% 9/23/2001: add testing set
% 10/15/2003: allow testing set not to have labels
% (C) copyright 2001 by Yu Hen Hu
% created: 3/17/2001
% Last modification: 9/29/2001, 10/15/2003

restart= 1;
train0=trainData;
[Kr,MN]=size(train0);
M= 11;
N=MN-M;

testys=1;
if testys==1,
    test0= testData;
    [Kt,MNt]=size(test0);
    % determine if the test set has labels
    if MNt == MN, labeled=1; else labeled=0; end
end

% scale feature vectors
% INPUT: if input of training file is scaled, the same scaling factor will
% be applied to scale the input of the tuning file, or other testing file
% ===============================================================================
scalein= 1; %input('If want to scale input to range of -5 to 5, enter 1 (default): ');
if isempty(scalein), scalein=1; end
if scalein==1,
    if testys==1,
        [tmp,xmin,xmax]=scale([train0(:,1:M); test0(:,1:M)],-5,5); % scale input
        train0(:,1:M)=tmp(1:Kr,:); test0(:,1:M)=tmp(Kr+1:Kr+Kt,:);
    else
        [train0(:,1:M),xmin,xmax]=scale(train0(:,1:M),-5,5); % scale input
    end
end
% output scaling will be performed after activation functions of output nodes
% have been decided.

% ================================================================
% start configure MLP: # of layers, # hidden neurons / layer,
% initialize weight matrices
% ==============================================================
%disp('=====================================================================');
L1 = numLayers; %input('Excluding input layer, enter total # of layers (default = 2), L = ');
L=L1+1;
%disp(['The output layer has ' int2str(N) ' neurons.']);
n(1) = M;
n(L) = N; % # of output neurons = output dimension
% w is a cell array with i-th element being a weight matrix
% of the i-th layer neurons.
if L > 2,
    for i=2:L-1,
        n(i)= numNeurons; %input(['Number of neurons in hidden layer # ' int2str(i-1) ' = ']);
        w{i}=0.001*randn(n(i),n(i-1)+1); % first column is the bias weight
        dw{i}=zeros(size(w{i})); % initialize dw
    end
end
w{L}=0.005*randn(n(L),n(L-1)+1); % first column is the bias weight
dw{L}=zeros(size(w{L}));
% ==============================================================
% choose types of activation function
% default: hidden layers, tanh (type = 2), output layer, sigmoid (type = 1)
% default parameter T = 1 is used.
% ==============================================================
atype=2*ones(L,1); atype(L)=1; % default
%disp('By default, hidden layers use tanh activation function, output use sigmoidal');
chostype= 1; %input('Enter 1 to change default values (default 0): ');
if chostype==1,
    %disp('=====================================================================');
    %disp('activation function type 1: sigmoidal');
%disp('activation function type 2: hyperbolic tangent');
%disp('activation function type 3: linear');
for l=2:L,
    atype(l)= functionChoice; %input(['Layer #' int2str(l) ' activation function type = ']);
end
atype(L) = 1;
end

% ==============================================================
% next load a tuning set file to help determine training errors
% or partition the training file into a training and a tuning file.
% ==============================================================
%disp('Enter 0 (default) if a pattern classification problem, ');
classreg= 0; %input('      1 for approximation problem: ');
% ==============================================================
%disp('=============================================================
%msg_1=[...
%'To estimate training error, choose one of the following:             '
%'1 - Use the entire training set to estimate training error;        '
%'2 - Use a separate fixed tuning data file to estimate training error;'
%'3 - Partition training set dynamically into training and tuning sets;'
%'   (This is for pattern classification problem)                     '
%disp(msg_1);
chos= 3; %input('Enter your selection (default = 1): ');
if chos==3, % partition the training file into a training and tuning
    % according to a user-specified percentage
    prc= percentTune; %input('Percentage (0 to 100) of training data reserved
    % for tuning: ');
    [tune,train0]=partunef(train0,M,prc);
    [Kr,MN]=size(train0); % this train0 is only a subset of original train0
    % this train0 is only a subset of original train0 and hence
    % Kr must be updated.
end
[Ktune,MN]=size(tune);

% ==============================================================
% scaling the output of training set data
% normally the output will be scaled to [outlow outhigh] = [0.2 0.8]
% for sigmoidal activation function, and [-0.8 0.8] for hyperbolic tangent
% or linear activation function at the output nodes.
% However, the actual output of MLP during testing of tuning file or testing
% file will be handled differently:
% a) Pattern classification problem: since we are concerned only with the
%    maximum among all output, the output of MLP will not be changed even it
%    ranges only
%    between [outlow outhigh] rather than [0 1]
% b) Approximation (regression) problem: the output of MLP will be scaled
%    back
%    for comparison with target values
% ==============================================================
Output from output nodes for training samples may be scaled to: 
[0.2 0.8] for sigmoidal activation function or 
[-0.8 0.8] for hyperbolic tangent or linear activation function.

scaleout=1; %input('Enter 1 (default) to scale the output: ');

if atype(L)==1, outlow = 0.2; elseif atype(L)==2 | atype(L) == 3, outlow = -
0.8; end 
outhigh=0.8;
if scaleout==1,
    [train0(:,M+1:MN),zmin,zmax]=scale(train0(:,M+1:MN),outlow,outhigh);
    % scale output
    [tune(:,M+1:MN),zmin,zmax]=scale(tune(:,M+1:MN),outlow,outhigh);
    % scale target value of tuning set
    if testys==1 & labeled==1 % if testing set specifies output
        [test0(:,M+1:MN),zmin,zmax]=scale(test0(:,M+1:MN),outlow,outhigh);
    end 
end

% now, we have a training file and a tuning file

% learning parameters
% learning parameters
% disp('=============================================================');
alpha= learningRate; %input('learning rate (between 0 and 1, default = 0.1)
alpha = ');
mom= momInput; %input('momentum constant (between 0 and 1, default 0.8) mom = ');

% termination criteria
% A. Terminate when the max. # of epochs to run is reached.
% disp('=============================================================');
nepoch= 500; %input('maximum number of epochs to run, nepoch = '); 
K = Kr; %input(['epoch size (default = ' int2str(min(64,Kr)) '), <= ' int2str(Kr) ')= ']);

% disp(['total # of training samples applied = ' int2str(nepoch*K)]);

% B. Check the tuning set testing result periodically. If the tuning set testing 
% results is reducing, save the weights. When the tuning set testing results 
% start increasing, stop training, and use the previously saved weights. 
% disp('=============================================================');
nck= 20; %input(['# of epochs between convergence check (> ' ... 
    %int2str(ceil(Kr/K)) ')= ']);
%disp(' ');%disp('If testing on tuning set meets no improvement for n0');maxstall=30; %input('iterations, stop training! Enter n0 = ');
nstall=0; % initialize # of no improvement count. when nstall > maxstall, quit.
if classreg==0,
    bstrate=0; % initialize classification rate on tuning set to 0
elseif classreg==1,
    bstss=1; % initialize tuning set error to maximum
    ssthresh=0.001; % initialize thres
end

% training status monitoring
E=zeros(1,nepoch); % record training error

%disp(' ');%disp(['the training error is plotted every ' int2str(ndisp) ' iterations']);
%disp('Enter <Return> to use default value. ')
chos1= 10000; %input('Enter a positive integer to set to a new value: ');
if isempty(chos1),
    ndisp=5;
elseif chos1>0,
    ndisp=chos1;
else
    ndisp=input('You must enter a positive integer, try again: ');
end

% intialization for the bp iterations

% initialization for the bp iterations
% t = 1; % initialize epoch counter
% ptr=1; % initialize pointer for re-sampling the training file
% not_converged = 1; % not yet converged

bpMultiConfigTest.m

% bp.m - Implementation of backpropagation algorithm
% (C) copyright 2001 by Yu Hen Hu
% created: 3/17/2001
% call bpconfig.m, cvgtest.m, bpdisplay.m, bptest.m
% rsample.m, randomize.m, actfun.m, actfunp.m
% partunef.m
%Modified by Charles Rodenkirch on 12/2013

clear all, close all
timeC = 0;
percentTune = 20;
allFeatureVectors = importdata('allFeatureVectorsClustered.txt');

for a = 1:10
numLayers = a;
for c = 1:5
    numNeurons = c;
    for f = 1:3
        functionChoice = f;
        for h = 1:9
            learningRate = h/10;
            for j = 1:9
                momInput = j/10;
                disp(numLayers);
                testData, trainData]=fsplit(allFeatureVectors,20,2);

                bpconfigWaterski; % configure the MLP network and learning parameters.

                timeC = timeC +1;
                % BP iterations begins
                while not_converged==1,
                    % start a new epoch
                    % Randomly select K training samples from the training set.
                    [train,ptr,train0]=rsample(train0,K,Kr,ptr); % train is K by M+N
                    z{1}=(train(:,1:M))'; % input sample matrix M by K
                    d=train(:,M+1:MN)'; % corresponding target value N by K

                    % Feed-forward phase, compute sum of square errors
                    for l=2:L, % the l-th layer
                        u{l}=w{l}*[ones(1,K);z{l-1}]; % u{l} is n(l) by K
                        z{l}=actfun(u{l},atype(l));
                    end
                    error=d-z{L}; % error is N by K
                    E(t)=sum(sum(error.*error));

                    % Error back-propagation phase, compute delta error
                    delta{L}=actfunp(u{L},atype(L)).*error; % N (=n(L)) by K

                    if L>2,
                        for l=L-1:-1:2,
                            delta{l}=(w{l+1}(:,2:n(l)+1))'*delta{l+1}.*actfunp(u{l},atype(l));
                        end
                    end

                    % update the weight matrix using gradient, momentum and random perturbation
                    for l=2:L,
                        dw{l}=alpha*delta{l}*[ones(1,K);z{l-1}]'+... % mom
                        mom*dw{l}+randn(size(w{l}))*0.005;
                        w{l}=w{l}+dw{l};
                    end

                    % display the training error
% Test convergence to see if the convergence condition is satisfied,
cvgtest;
t = t + 1; % increment epoch count
end % while loop

% disp('Final training results:')
if classreg==0,
    [Cmat,crate]=bptest(wbest,tune,atype),
elseif classreg==1,
    SS=bptestap(wbest,tune,atype),
end

if testys==1, % disp('Apply trained MLP network to the testing data. The results are: ');
    if classreg==0,
        [Cmat,crate,cout]=bptest(wbest,test0,atype,labeled,N);
        %if labeled==1,
        %disp('Confusion matrix Cmat = '); disp(Cmat);
        %disp(['classification = ' num2str(crate) '%'])
        %elseif labeled==0,
        % print out classifier output only if there is no label
        %disp('classifier outputs are: ')
        %disp(cout);
    end
    elseif classreg==1,
        SS=bptestap(wbest,test0,atype),
    end
end

keepTrack(timeC,1) = numLayers;
keepTrack(timeC,2) = numNeurons;
keepTrack(timeC,3) = functionChoice;
keepTrack(timeC,4) = learningRate;
keepTrack(timeC,5) = momInput;
keepTrack(timeC,6) = crate;
end
end
end

FinalWaterskiPredictionApp.m

% bp.m - Implementation of backpropagation algorithm
% (C) copyright 2001 by Yu Hen Hu
% created: 3/17/2001
clear all, close all

timeC = 0;
percentTune = 20;

trainData = importdata('allFeatureVectors.txt');

numLayers = 2;
numNeurons = 4;
functionChoice = 3;
learningRate = 0.4;
momInput = 0.7;
done = 0;

while done == 0
    testData(1,1) = input('What is your predicted jump score (Feet) ?')
    if isempty(testData(1,1))
        disp('error no value entered');
    else
        done = 1;
    end
end

done = 0;
while done == 0
    testData(1,2) = input('What is your predicted trick score (Points)?')
    if isempty(testData(1,2))
        disp('error no value entered');
    else
        done = 1;
    end
end

done = 0;
while done == 0
    testData(1,3) = input('What is your predicted Slalom score (Buoy Count)?')
    if isempty(testData(1,3))
        disp('error no value entered');
    else
        done = 1;
    end
end

done = 0;
while done == 0
    testData(1,4) = input('Number of competitors at the tournament?')
    if isempty(testData(1,4))
        disp('error no value entered');
    else
        done = 1;
    end
end
done = 0;
while done == 0
testData(1,5) = input('Rain predicted for Saturday (Inches)?)
if isempty(testData(1,5))
    disp('error no value entered');
else
done = 1;
end
end

done = 0;
while done == 0
testData(1,6) = input('Rain predicted for Sunday (Inches)?)
if isempty(testData(1,6))
    disp('error no value entered');
else
done = 1;
end
end

done = 0;
while done == 0
testData(1,7) = input('Wind predicted for Saturday (MPH)?)
if isempty(testData(1,7))
    disp('error no value entered');
else
done = 1;
end
end

done = 0;
while done == 0
testData(1,8) = input('Wind predicted for Sunday (MPH)?)
if isempty(testData(1,8))
    disp('error no value entered');
else
done = 1;
end
end

done = 0;
while done == 0
testData(1,9) = input('What is the predicted temp for Saturday (Degrees)?)
if isempty(testData(1,9))
    disp('error no value entered');
else
done = 1;
end
end
done = 0;
while done == 0
testData(1,10) = input('What is the predicted temp for Sunday (Degrees)?')
if isempty(testData(1,10))
    disp('error no value entered');
else
done = 1;
end
end

done = 0;
while done == 0
testData(1,11) = input('Is this spring season (enter 0) or fall season (enter 1)?')
if isempty(testData(1,11))
    disp('error no value entered');
else
done = 1;
end
end

bpconfigWaterski; % configure the MLP network and learning parameters.

timeC = timeC +1;
% BP iterations begins
while not_converged==1,
    % start a new epoch
    % Randomly select K training samples from the training set.
    [train,ptr,train0]=rsample(train0,K,Kr,ptr); % train is K by M+N
    z{1}=(train(:,1:M))'; % input sample matrix  M by K
    d=train(:,M+1:MN)'; % corresponding target value  N by K

    % Feed-forward phase, compute sum of square errors
    for l=2:L, % the l-th layer
        u{l}=w{l}*ones(1,K);z{l-l}'; % u{l} is n(l) by K
        z{l}=actfun(u{l},atype(l));
    end
    error=d-z(L); % error is N by K
    E(t)=sum(sum(error.*error));

    % Error back-propagation phase, compute delta error
    delta(L)=actfunp(u(L),atype(L)).*error; % N (=n(L)) by K
    if L>2,
        for l=L-1:-1:2,
            delta{l}=(w{l+1}(;2:n(l)+1))'*delta{l+1}.actfunp(u{l},atype(l));
        end
    end

    % update the weight matrix using gradient, momentum and random perturbation
for l=2:L,
    dw{l} = alpha * delta{l} * [ones(1,K); z{l-1}]
    mom * dw{l} + randn(size(w{l}))*0.005;
    w{l} = w{l} + dw{l};
end

% display the training error
% bpdisplay;

% Test convergence to see if the convergence condition is satisfied,
% cvgtest;
t = t + 1; % increment epoch count
end % while loop

% disp('Final training results:')
if classreg == 0,
    [Cmat, crate] = bptest(wbest, tune, atype),
elseif classreg == 1,
    SS = bptestap(wbest, tune, atype),
end
if testys == 1,
    % disp('Apply trained MLP network to the testing data. The results are: ') ;
    if classreg == 0,
        [Cmat, crate, cout] = bptest(wbest, test0, atype, labeled, N);
        % if labeled == 1,
        % disp('Confusion matrix Cmat = '); disp(Cmat);
        % disp(['classification = ' num2str(crate) '%'])
        % elseif labeled == 0,
        % print out classifier output only if there is no label
        % disp('classifier outputs are: ')
        % disp(cout);
        % end
        elseif classreg == 1,
            SS = bptestap(wbest, test0, atype),
    end
end

disp('Based on our calculations, you have a chance at placing in the
following ranges: ')
if cout(1,1) == 1
    disp('1st through 9th place')
end
if cout(2,1) == 1
    disp('10th through 19th place')
end
if cout(3,1) == 1
    disp('20th through 29th place')
end
if cout(4,1) == 1
    disp('30th through 39th place')
end
if cout(5,1) == 1
    disp('40th through 49th place')
end

disp('GOOD LUCK SKIING THIS WEEKEND, SWERVE HARD!');

KmeanWaterski.m

function [W,iter,finalDist]=kmeansWaterski(X,W,er,itmax)
% Usage: [W,iter,finalDist]=kmeansWaterski(X,W,er,itmax,tp)
% k-means clustering algorithm (function call version)
% Input - W: initial weight vectors c by N matrix, c: # of clusters
% - X: K by N input vectors to be clustered
% - er: 0<er<1, fractional error between successive
%       distortion for convergence. If not specified,
%       default = 0.01
% - itmax: maximum iterations before terminate iterations
% if not specified, default value = c: # of clusters
% - tp: a parameter vector.
%   2/23/2001: tp(1) = 0 using fixed # of clusters
%   = 1 may delete some empty clusters at the end
%   9/21/2001: tp(2) = 0 L2 norm (default)
%   = 1 L1 norm
%   = 2 L_inf norm
% Output - W: final weight vectors (code book)
% - iter: actual number of iterations to converge
% - finalDist: final distortion
% Original = Yue hu Hen
% created: 1/2/94
% Modified by = Charles Rodenkirch
% Last modified 12/19/2013
finalDist = -1;

[c,N]=size(W);
[K,N]=size(X);

tp=[0 0];
dtype=tp(2);

if c==1,
  iter=1; Sb=zeros(N); D=0;
  if N==1,
    W=X;
    Sw=0;
  elseif N > 1,
    W=mean(X);
    tmp=X-ones(K,1)*W;
    Cova=tmp'*tmp/K;
    Sw=K*Cova;
  end
  return
end % the case of c > 1 will continue

converged=0; % reset convergence condition to false
Dprevious=0;
iter=0;
while converged==0, iter=iter+1;

% step A. evaluation of distortion using dtype norm
tmp=mydist(X,W,dtype); % K x C
[tmp1,ind]=sort(tmp'); % first row of ind gives new cluster assignment
% of each data vector, tmp1, ind: 1 X K

% step B. compute total distortion with present assignment and check for
% convergence. If converged, we still update weights one more
time!
if dtype==0, % L2 norm
    Dpresent=sum(tmp1.*tmp1); % distortion before weight is adjusted.
elseif dtype==1, % L1 norm
    Dpresent=sum(tmp1);
elseif dtype==2, % L_Inf norm
    Dpresent=max(tmp1);
end

if abs(Dpresent-Dprevious)/abs(Dpresent) < er | iter == itmax,
    finalDist = mean(sum(tmp.*tmp));
    converged=1;
end

% Step C. update weights (code words) with new assignment
if tp(1)==1, cidx=[1:c]; end
for i=1:c,
    nc(i)=sum([ind(1,:)==i]);
    if nc(i)>1,
        W(i,:)=sum(X(ind(1,:)==i,:))/nc(i);
    elseif nc(i)==1,
        W(i,:)=X(ind(1,:)==i,:);
    elseif nc(i)==0,
        if tp(1)==0, % if must have n non-
            [tmp1,midx]=sort(-tmp1); % sort samples according to negative
            % from their current center. The most remote ones come first
            W(i,:)=X(midx(i,:)); % if an empty cluster reassign it
            % to the i-th most remote samples
        elseif tp(1)==1, % if empty clusters can be eliminated,
            cidx=setdiff(cidx,i);
        end
    end
end % i-loop

if tp(1)==1, % remove clusters that are empty if instructed so
    W=W(cidx,:);
    c=length(cidx);
end

Dprevious=Dpresent;
end % while loop
WaterskiCluster.m

% clusterdemo.m - clustering algorithm demonstration program
% (C) copyright 2001 by Yu Hen Hu
% mfiles used: datagen.m
clear all, close all, clf
% generate some data using datagen.m, kmeansf.m
% Nvec: (1xclass) # data in each of the c gaussian distr.
% mean_var:
% 3 x class: mean (1st 2 rows) and variance of each class.
% 4 x class: mean (1st 2 rows) and variance 1 and variance 2
% 5 x class: mean (1st 2 rows), var 1, var 2, and rotation angle
% rotation angle is in [0 90)

x= importdata('slalomScores.txt');

% apply kmeansf.m to the data
% Input - W: initial weight vectors  c by N matrix, c: # of clusters
% - X: K by N input vectors to be clustered
% - er: 0<er<1, fractional error between successive distortion for convergence. If not specified, default = 0.01
% - itmax: maximum iterations before terminate iterations
% if not specified, default value = c: # of clusters
% Output - W: final weight vectors (code book)
% - iter: actual number of iterations to converge

finalDist = 0;

%for n = 3:40
n = 7;

c=n;
[K,N]=size(x);
er = 1e-5;
itmax=2000;
done=0;
xmean=mean(x);

while done==0,  % while not done yet,
    W0=0.1*randn(c,N)+ones(c,1)*xmean;
    [W,iter, finalDist]=kmeansWaterski(x,W0,er,itmax);
    if iter<itmax,
        reportData(n,1) = n;
        reportData(n,2) = finalDist;
        done = 1;  % converged
    end
end
%end

disp(reportData);
function y=actfun(x,type,par)
% Usage: y=actfun(x,type,par)
% Compute activation functions
% x: net function, a K x N matrix
% y: activation function, a K x N matrix
% type: 1 - sigmoid, 2 - tanh, 3- linear, 4 - radial
% par: parameter list
%   sigmoid: T,  y=1/(1+exp(-x/T)), yp=y*(1-y)/T
%   tanh: T,     y=(exp(x/T)-exp(-x/T))/(exp(x/T)+exp(-x/T))
%         yp=(1-y*y)/T
%   linear:m,b   y=ax+b, yp=a
%   radial:m,sig y=exp(-(x-m)^2/sig^2), yp=-2(x-m)*y/sig^2
% (C) copyright 2001 by YU HEN HU
% created: 2/5/2001
if nargin<=2, % if omitted from input vari list, set default
    if type==3,
        par=[1 0];
    elseif type==4,
        par=[0 1];
    else
        par=1;
    end
end
if nargin==1, % no type info either
    type=1;
end
switch type
% sigmoid
case 1
    T=par(1);
    y = 1./(1+exp(-x/T));
% tanh
case 2
    T=par(1);
    tmp=exp(x/T);
    y=(tmp-1./tmp)./(tmp+1./tmp);
% linear
case 3
    a=par(1); b=par(2);
    y=a*x+b;
% radial
case 4
    m=par(1); sig=par(2);
    s=sig^2;
    tmp=(x-m).*'(x-m);
    y=exp(-tmp/s);
end

function yp=actfunp(x,type,par)
% Usage: yp=actfunp(x,'type',par)
% Compute activation functions and their derivatives
% x: net function, a K x N matrix
% y: activation function, a K x N matrix
% type: 1 - sigmoid, 2 - tanh, 3 - linear, 4 - radial
% par: parameter list
%   sigmoid: T,  y=1/(1+exp(-x/T)), yp=y*(1-y)/T
%   tanh: T,    y=(exp(x/T)-exp(-x/T))/(exp(x/T)+exp(-x/T))
%         yp=(1-y*y)/T
%   linear: m,b y=ax+b, yp=a
%   radial: m,sig y=exp(-(x-m)^2/sig^2), yp=-2(x-m)*y/sig^2
% (C) copyright 2001 by YU HEN HU
% created: 2/5/2001

if nargin<=2, % if omitted from input vari list, set default
    if type==3,
        par=[1 0];
    elseif type==4,
        par=[0 1];
    else
        par=1;
    end
end
if nargin==1, % no type info either
    type=1;
end

switch type
    case 1 % sigmoid
        T=par(1);
        y = 1./(1+exp(-x/T));
        yp = y.*(ones(size(y))-y)/T;
    case 2 % tanh
        T=par(1);
        tmp=exp(x/T);
        y=(tmp-1./tmp)/(tmp+1./tmp);
        yp=(ones(size(y))-y*y)/T;
    case 3 % linear
        a=par(1); b=par(2);
        y=a*x+b;
        yp=a*ones(size(y));
    case 4 % radial
        m=par(1); sig=par(2);
        s=sig^2;
        tmp=(x-m).*(x-m);
        y=exp(-tmp/s);
        yp=(-2/sig^2)*(x-m).*y;
end

bptest.m

function [Cmat,crate,cout]=bptest(w,test,atype,labeled,N)
% Usage: [Cmat,crate,cout]=bptest(w,test,atype,labeled,N)
% This routine is used for pattern classification problems only.
% testing MLP network for classification rate and confusion matrix
% test (K by M+N, if labeled=1, K by M if labeled = 0)
% the feature vectors (K by M) and if labeled=1, target vectors (K by N)
% w{1}, 1<= l <= L consists of the L-1 weight matrices, w{1}=[].
% each w{l} is n(l) x (n(l-1)+1)
% assume the output uses 1 of N encoding. Thus, class #1 is [1 0 ... 0]
% class #2 is [0 1 0 .. 0], etc.
% atype(1:L): specify activation function types for each layer
% if ignored, type(1) = 1
% labeled=1 (default) if test data set has labels, = 0 otherwise
% N: label dimension, will not be needed if labeled =1
% Cmat: N by N the confusion matrix
% crate = sum(diag(Cmat))/K: classification rate
% cout: classifiers output wrt each testing data, K by N
% used by cvgtest.m
% copyright (c) 1996-2001 by Yu Hen Hu
% Last modified: 3/20/2001, 10/15/2003

if nargin<4, labeled=1; end  % default the test set has labels
if nargin<3, atype=ones(length(w),1); end

[K,MN]=size(test); % find # of samples and input+output dimension
[m1,M1]=size(w{2}); % find input dimension
M=M1-1;
if labeled ==1, N=MN-M; end
L=length(w);

z{1}=(test(:,1:M))'; % input sample matrix  M by K
if labeled==1,
    target=test(:,M+1:MN)'; % corresponding target value  N by K
end

% Feed-forward phase,
for l=2:L, % the l-th layer
    z{l}=actfun(w{l}*[ones(1,K);z{l-1}],atype(l)); % z{l} is n(l) by K
end

% To compute confusion matrix, we need at least two outputs. If the MLP has
% only one output to encode two classes, we change it to a 1 out of 2
% encoding
if N==1,
    cout=[[z{L}>0.5];[z{L}<0.5]]+zeros(2,K); % convert into one of N encoding
    if labeled==1, % if testing set has labels
        target=[target;ones(1,K)-target];
    end
    N=2; % change size of confusion matrix to 2 by 2
else
    cout = [(z{L} - ones(N,1)*max(z{L})) == 0]+zeros(N,K); % N x K
end

if labeled==1,
    % Then we calculate the NxN confusion matrix.
    Cmat=((target-ones(N,1)*max(target))==0)+zeros(N,K));...
        *cout';
    crate=sum(diag(Cmat))*100/K;
elseif labeled==0, % if the testing set has no labels, only provide cout
    Cmat=eye(N); crate=1;
end

cvgtest.m

% cvgtest.m - convergence test.
% A subroutine called specifically by bp.m
% mfile used: bptest.m, partunef.m, scale.m
% (C) copyright 2001 by Yu Hen Hu
% created: 3/19/2001
% modified: 10/16/2001: add line 26 bstrate=crate;

% ==============================================================
% termination criteria
% A. Terminate when the max. # of epochs to run is reached.
% ==============================================================
if t>=nepoch;
    not_converged=0;
    %disp('Terminate training since Max # of epochs has been reached');
end
% ==============================================================
% B. Check the tuning set testing result periodically. If the tuning set
%    testing results is reducing, save the weights. When the tuning set testing
%    results start increasing, stop training, and use the previously saved weights.
% ==============================================================
if rem(t,nck)==0, % time to check tuning set testing result
    %disp([\'Epoch # \ ' int2str(t)])
    if classreg==0, % if pattern classification problem
        [Cmat,crate]=bptest(w,tune,atype),
        if crate > bstrate, % if the classification of tuning set is improving
            bstrate=crate;
            wbest=w; % memorize the best weights
            nstall=0; % reset the no-improvement count
        else
            nstall=nstall+1; % if no improvement, increment the no-improvement count
        end
    end % if crate
elseif classreg==1, % for regression problems
    SS=bptestap(w,tune,atype) % sum of square error/(# samples*# output)
    if SS < bstss,
        bstss=SS;
        wbest=w;
        nstall=0; % reset stall count
    else
        nstall=nstall+1;
    end % if SS
end % if classreg

if nstall>maxstall, % continuous maxstall no improvement, terminate
not_converged=0;
%disp([ 'Terminate because no improvement for ' int2str(nstall) ' consecutive checks' ]); end % if nstall

% for pattern classification problem, if % tuning set classification rate = 100%, terminate
if classreg==0 & crate == 100,
   not_converged=0;
   %disp('Terminate because classification rate of tuning set is 100%')
% for approximation problem, if tuning set error < ssthresh, terminate
elseif classreg==1 & SS < ssthresh,
   not_converged=0;
   %disp(['Terminate as approx. error of tuning set < ' num2str(ssthresh)])
end

if chos == 3, % repartition the training and tuning set
   [tune,train0]=partunef([train0;tune],M,prc);
   [Kr,MN]=size(train0);  % this train0 is only a subset of original train0
end

deidgrad.m

%possible decision gradient code

x1=-10:10;x2=-10:10;
[xlg,x2g]=meshgrid(x1,x2);
z{1}=horzcat(x1,x2);
y=actfun(w{l}*[ones(1,2);z{1}],atype(l));
surf(xlg,x2g,y)

fsplit.m

function [x1,x2]=fsplit(x,ratio,mode)
% Usage: [x1,x2]=fsplit(x,ratio,mode)
% Random partition a matrix x row-wise according to ratio:1
% If x is K by N, x1 is K1 by N, x2 is K2 by N
% K1+K2 = K,
% mode = 1 (default)  0 < ratio < 1, K1/K2 = ratio:1
% mode = 2, ratio is percentage. K1/K2 =ratio:100-ratio
% call randomize.m
% (C) copyright 2001 by Yu Hen Hu
if nargin<3
    mode=1;
end

[k,n]=size(x);
if k==1 & n>1,
    x=x'; % if x is a row vector, convert it to a column vector
    k=n;
end
if k>1
    if mode==1,
        k1=min(k-1,max(1,round(ratio*k/(1+ratio)))); % 1 <=k2<=k-1
    elseif mode==2
        k1=min(k-1,max(1,round(ratio*k*0.01))); % 1 <=k2<=k-1
    end

    k2=k-k1;
    x=randomize(x);
    x1=x(1:k1,:); x2=x(k1+1:k,:);
end

mydist.m

function d=mydist(X,W,type,para)
% Usage: d=mydist(X,W,type,para)
% X: K x M
% W: C x M
% d: K x C distance between K 1 X M vectors and
%      C 1 X M prototypes
% type = 0 (default) L2 norm, para=[]
%     = 1 L1 norm, para=[]
%     = 2 L_inf norm (box distance), para=[]
%     = 3 distance taking covariance matrix into account
% para: c x 1, para{i}: M x M inverse of cov. matrix
% (C) 2001 by Yu Hen Hu
% created: 4/5/2001,
% modified: 9/5/2001 to add type 3 distance

if nargin <3,
    type=0; % default is L2 norm
end
if type == 3,
    if nargin < 4, error('must give covarince matrices'); end
end

[K,M]=size(X);
[C,M1]=size(W);
if M~=M1, error('X and W dimension should be the same!'); end
if type==0, % L2 norm
    if M >1,
wnorm=sum((W.*W)'); % 1 by C vector
xnorm=sum((X.*X)'); % 1 by K vector

elseif M==1,
    wnorm=(W.*W)'; % 1 by C vector
    xnorm=(X.*X)'; % 1 by K vector
end
d=sqrt(xnorm'*ones(1,C)-2*X*W'+ones(K,1)*wnorm); % K by C matrix

elseif type==1, % L1 norm
d=[];
    if C <=K, % compute by column
        for i=1:C,
            d=[d sum(abs(X'-W(i,:)'*ones(1,K))')];
        end
    elseif K < C, % compute by row
        for i=1:K,
            d=[d; sum(abs(X(i,:)'*ones(1,C)-W'))];
        end
    end

elseif type==2, % L_inf norm
    d=[];
    if C <=K, % compute by column
        for i=1:C,
            d=[d max(abs(X'-W(i,:)'*ones(1,K))')];
        end
    elseif K < C, % compute by row
        for i=1:K,
            d=[d; max(abs(X(i,:)'*ones(1,C)-W'))];
        end
    end

elseif type==3, % L2 norm with covariance matrix
    % (x-w)*Cinv*(x-w)'
    % calculate x-w
    d=[];
    for i=1:C,
        dxw=X-ones(K,1)*w(i,:); % K by M
        d = [d sum((para{i}*dxw').*dxw')'];
    end
end

partunef.m

function [x1,x2]=partunef(x,M,prc)
% Usage: [x1,x2]=partunef(x,M,prc)
% callable version of partune.m
% called by: bpconfig.m, cvgtest.m
% mfiles used: fsplit.m
% partition the training data samples x into a training
% set and a tuning set according to a user specified percentage ratio
% # rows in x1: # rows in x2 ~ prc: 100-prc
% then save them back in ASCII format into two files.
% each column of x can have only two values xlow, xhigh
% and these values has to be figured out from x
% (C) copyright 2001 by Yu Hen Hu
[Kr,MN]=size(x);
x1=[]; x2=[];
for i=M+1:MN,
   xhigh=max(x(:,i));
   idx=find(x(:,i)==xhigh);
   [ridx,tidx]=fsplit(idx,prc,2);
   x1=[x1;x(ridx,:)];
   x2=[x2;x(tidx,:)];
end

function [B,I]=randomize(A,rowcol)
% Usage: [B,I]=randomize(A,rowcol)
% randomize row orders or column orders of A matrix
% rowcol: if =0 or omitted, row order (default)
%         if = 1, column order
% copyright (C) 1996-2001 by Yu Hen Hu
% Last modified: 8/30/2001

rand('state',sum(100*clock))
if nargin == 1,
   rowcol=0;
end
if rowcol==0,
   [m,n]=size(A);
   p=rand(m,1);
   [p1,I]=sort(p);
   B=A(I,:);
elseif rowcol==1,
   Ap=A';
   [m,n]=size(Ap);
   p=rand(m,1);
   [p1,I]=sort(p);
   B=Ap(I,:)';
end

function [train,ptr,x]=rsample(x,K,Kr,ptr)
% Usage: [train,ptr]=rsample(x,K,Kr,ptr)
% random sample K out of Kr rows from matrix x.
% ptr: current row count pointer. K rows will be taken from
% row # ptr+1 to ptr + K
% if ptr + K > Kr, the x matrix will be randomized by row
% and then the first K rows will be taken.
% Each time ptr will be updated
% call: randomize.m
% (C) copyright 2001 by Yu Hen Hu
if ptr+K > Kr,
x=randomize(x);
train=x(1:K,:);
ptr=K;
else
    train=x(ptr+1:ptr+K,:);
    ptr=ptr+K;
end

rscale.m

function [x,xmin,xmax]=scale(x,low,high,type)
% Usage: [x,xmin,xmax]=scale(x,low,high,type)
% % linearly scale x into the range of low to high
% it must be that high - low > 0
% if type = 0 (default), low and high are scalars and scaling is uniform in each dim
%     = 1, scaling is done at each individual dimension in the range
% vectors low and high
% output: x - scaled x, xmax: original maximum, xmin: original min
% input: x - original x, low: desired min, high: desired max.
% copyright (c) 1996 by Yu hen Hu
% created 9/21/96
% modified 9/22/96: add xmin, xmax to output
% modified 4/5/2001: add type to allow scaling to indivisual ranges in each dim.
% if nargin==3,
%     type=0;
end

[K,M]=size(x);
range=high-low; % type 0, scalar, type 1, may be a 1 X M vector
if type==0,
    xmax=max(max(x));xmin=min(min(x));xrange=xmax-xmin;
x=(x-xmin)*range/xrange+low;
elseif type==1,
    if size(high)==size(low) == [1 1];
        high=ones(1,M)*high; low=ones(1,M)*low;
    end
    low=diag(diag(low))'; range=diag(diag(range))'; % make them 1 x M vectors
    xmax=max(x); xmin=min(x); xrange=xmax-xmin; % each 1 x M vector
    mask=[xrange<1e-3]; % maskout elements too small, 1 x M
    range=(1-mask).*range+mask; xrange=(1-mask).*xrange+mask;
    x=(x-ones(K,1)*xmin)*diag(range./xrange)+ones(K,1)*low;
end