CT Slice Localization

Vibhor Jain

May 2013

Abstract

Radiologists and clinicians will face the problem of locating CT scans along the vertical axis of the body when trying to compare scans. This becomes especially challenging when trying to localize individual CT slices instead of CT volume scans. The proposed project builds on an existing feature extraction scheme and uses an artificial neural network to solve the regression problem. The experiments shows that the accuracy of the neural network to be reasonably comparable to the existing lazy learning approaches to the regression problem[2][3].

0.1 Background/Motivation

A radiologist or clinician is often faced with the prospects of comparing various CT (Computer Tomography) scans, but must first find their location along the body (vertical axis) before comparing them. The specialist could choose to compare volume scans by loading the query scans to a local machine and manually localizing the query scans, but this would take too much time and effort from the specialist. The specialist could also choose to compare various slices, but then needs to resolve the issue of localizing those slices to the body[2].

Current methods resolve the localization issue for single slices by creating various landmarks among various scans and interpolating to generate localization results. There are also methods that extract certain features from the CT slices and apply instance-based machine learning techniques to solve such a regression problem[3]. This project explores the possibility of applying a more eager-based learning technique (such as artificial neural networks) on the same feature extraction scheme. The goal of this project will be to apply a neural network architecture on the presented feature extraction scheme to solve the regression problem of localizing CT scan slices along the vertical axis of the body with the same order of accuracy of at least one of the methods proposed [2][3].

This project will first describe the dataset used and the feature extraction scheme that made the dataset. Next, the project describes the equations that will be used to model the neural network architecture being used. The experiments are then described to find an optimum neural network architecture and compare its performance (in terms of accuracy) to other proposed methods[3][2]. The project finally closes off with describing possible improvements to the experiments described before.

0.2 The Dataset

The dataset consists of 53,500 data points and 386 attributes. The data was surveyed over 96 different patients with no gender specified. The last attribute value is a real valued number between 0 and 180 representing the output to be predicted. The first attribute value represents the patient ID number, so it is discarded (it doesn't make sense to evaluate patient predictions based on a patient ID number). This is assuming a person is approximate 1.8 meters long (0 being at the top of the head while 180 being at the sole of the foot). The data in the current dataset actually gives values between 0 and 100 to reflect the vertical positions of the major organs in the body. The rest of the attributes in between are part of two separate histograms: one describing bone density around the body (attributes 2 to 241), and one describing soft tissue density around the body (attributes 242 385)[1].

Before extracting data from the images, the images are normalized by extracting the smallest square image I_c that bounds the body scan. The image is then partitioned into polar space. For the bone histogram, the images are partitioned into 24 radial lines and 11 arcs. For the soft tissue histogram, the images are partitioned into 18 radial lines and 8 arcs (Thus making 240 attributes/regions for the bone histogram and 144 attributes/regions for

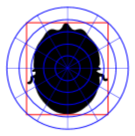


Figure 1: CT scan partitioned into polar regions[2]

the soft tissue histogram). The value in each bin is described as

$$v_i = \{ \begin{array}{ll} -0.25 & \text{if bin } i \notin I_c \\ \frac{p_i}{p_i+n_i} & \text{otherwise} \end{array} \tag{1}$$

where p_i describes the number of pixels in region i in I_c that are part of the bone (pixel values $\geq$ 300 HU (Hounsfiled unit)) or soft tissue (pixel values $\leq$ -500 HU) and n_i describes the number of pixels in region i in I_c that are part of neither bone nor soft tissue[3].

0.3 Network Architecture Specifications

In the neural network, the input vectors are fed into a network of perceptrons, which produce the output vectors. This network consits of a layer of input perceptrons, a single layer of hidden perceptrons, and a layer of output perceptrons. The equations for the perceptrons are give as below:

$$net = \sum_i w_i x_i \tag{2}$$

$$g(net) = \frac{a_{max} - a_{min}}{1 + e^{-net}} + a_{min} \tag{3}$$

where the x's represent the inputs for the preceptrons, the w's represent the weights associated with the connections going into the perceptrons (and the weight bias for the perceptron), and $g(net)$ represents the output of the perceptron. We use these equations to model the neural network

$$net_{pj} = \sum_i w_{ji} x_i \tag{4}$$

$$z_j = g(net_{pj}) \tag{5}$$

$$net_{pk} = \sum_j w_{kj} z_j \tag{6}$$

$$y_k = g(net_{pk}) \tag{7}$$

where x_i model the inputs, z_j model the inputs to the hidden units, and y_k model the outptus. We note that $g\prime(net)$ is given by:

$$g\prime(net) = \frac{\partial g(net)}{\partial net} = (g(net) - a_{min})(1 - \frac{1}{a_{max} - a_{min}}(g(net) - a_{min})) \tag{8}$$

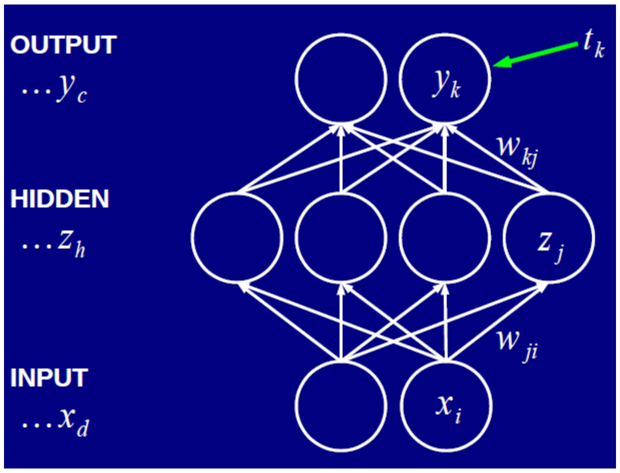


Figure 2: Model of Neural Net Architecture[6]

[4] If the output units are modeled by linear functions, $g(net_{pk}) = net_{pk}$ and $g\prime(net_{pk}) = 1$. To execute the backpropogation learning algorithm, we implement the training rule in the form

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \tag{9}$$

where η is the learning rate that is adjusted and E is the sum-squared error function being optimized. We also implement the addition of momentum to the network weights through the following

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1) \tag{10}$$

where $-\frac{\partial E}{\partial w_{ji}} = \delta_j x_{ji}$[5]. The momentum parameter α is varied with time so that

$$\alpha(n) = \alpha(n-1)c \tag{11}$$

for some $c > 1$. The weights w and partial derivatives $\frac{\partial E}{\partial w}$ are calculated based on the model seen in Figure 2. The error function E being used is given by:

$$E = \frac{1}{2}\sum_p \sum_k (t_{pk} - y_{pk})^2 + \frac{1}{2}\gamma \sum w^2 \tag{12}$$

where p iterates over all the given training or test examples and k iterates over the dimensionality of the output vectors[4]. The elements t_{pk} come from the elements of the target output vectors of the training and test data set, and the elements y_{pk} come from the elements of the output vectors produced by the neural network.The w's are added to implement a penalty term for the algorithm to seek small weight values[5].

If there are hidden units in the neural network, there are a set of weights for the output layer and a set of weights for the hidden layer in Figure 2.

The partial derivatives $\frac{\partial E}{\partial w}$ for the output layer weights are given by:

$$-\frac{\partial E}{\partial w_{kj}} = -\sum_p \frac{\partial E}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial w_{kj}} - \gamma w_{kj} \tag{13}$$

$$= \sum_p (t_{pk} - y_{pk}) g\prime(net_{pk}) z_{pj} - \gamma w_{kj} \tag{14}$$

$$= \sum_p \delta_{pk} z_{pj} - \gamma w_{kj} \tag{15}$$

The partial derivatives for the hidden layer weights are given by:

$$-\frac{\partial E}{\partial w_{ji}} = -\sum_p \frac{\partial E}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ji}} - \gamma w_{ji} \tag{16}$$

$$= -\sum_p \frac{\partial E}{\partial z_{pj}} \frac{\partial z_{pj}}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ji}} - \gamma w_{ji} \tag{17}$$

$$= -\sum_p (\sum_k \frac{\partial E}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial z_{pj}}) \frac{\partial z_{pj}}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ji}} - \gamma w_{ji} \tag{18}$$

$$= \sum_p (\sum_k (t_{pk} - y_{pk}) g\prime(net_{pk}) w_{kj}) g\prime(net_{pj}) x_{pi} - \gamma w_{ji} \tag{19}$$

$$= \sum_p (\sum_k \delta_{pk} w_{kj}) g\prime(net_{pj}) x_{pi} - \gamma w_{ji} \tag{20}$$

$$= \sum_p \delta_{pj} x_{pi} - \gamma w_{ji} \tag{21}$$

If there are no hidden units, the partial derivatives $\frac{\partial E}{\partial w}$ for the output layer weights are similar to the equation in (6):

$$-\frac{\partial E}{\partial w_{ki}} = \sum_p (t_{pk} - y_{pk}) g\prime(net_{pk}) x_{pi} - \gamma w_{ki} \tag{22}$$

The given equations for the partial derivatives up to this point are to be implemented for batch learning. For online learning, p simply iterates over one example[4].

0.4 Experiments

The experiments were run in MATLAB on the engapps00 server. The neural network architecture was assessed over the dataset through the hold-n-out cross validation technique (n=10,000) over the first 50,000 data points, where the 10,000 data points were taken consecutively through the 50,000 points. The last 3,500 data points are omitted from the experiment so that the error assessment over the 5 testing sets can be easily averaged. The average SSE (sum-squared error) was assessed for each hidden and output unit quantities given in Table 1 over the 5 sets of training and testing data set to determine the optimal neural network configuration. First the number of output units was set to 1 and the number of hidden units was varied to determine the optimum number of hidden units. The number of hidden units was then kept at this value and the number of output units was varied to determine the optimal number of output units (see Figure 4).

In addition to average SSE, the absolute difference between the target and network output vector over all training/test examples was analyzed for the optimal number of hidden units for one output unit and for the optimal number of output units for the optimal hidden units. With one output unit, the target value can be taken from the data as it's already recorded. For more output units, the target scalar values for each data point are converted to a binary valued vector (with a real-valued offset unit at the end). The binary units indicate the location of the target value along the number line (for example: between 0 and 50, between 50 and 100) These experiments were conducted over all the training and testing example sets using batch learning.The network's performance was compared with Graf's method by comparing the empirical CDF of the test data predictions of the network with Graf's empirical CDF[2].

The hidden units were modeled with a logistic function and were incremented in integer values. The output units were modeled with a linear function and a logistic function, and were incremented in factors of 100 (plus one additional unit). The last output unit is modeled with a linear function while the rest of the output units are modeled with a binary logistic funciton. For example: for an output layer with 3 units the first unit outputs 1 if the output is between 0 and 50, the second unit outputs 1 if the output is between 50 and 100, and the third unit is the offset value from the lower bound (0 or 50). For the network's output units, the output values from the logistic units are rounded to the nearest integer before comparing to the target values. Before comparing the network's output to the target values, the matrix of output values and target values are converted back to vectors of scalar values (by adding the offset to the starting value indicated by the binary vector).

The maximum and minimum values for the output logistic units were set to 1 and 0, while the maximum and minimum values for the hidden logistic units were arbitrarily set to .5 and -.5. The range of weight values was also arbitrarily set to be between -.5 and .5. The penalty term γ was set to 1×10^{-9} to avoid the extreme cases of overfitting the data. The initial value for the momentum parameter α was set to 1×10^{-5}, and the growth c for α (in 11) and the learning rate η chosen to be large enough to converge within a reasonable number of epochs but also small enough so the error from the backpropogation algorithm wouldn't diverge. The parameters c and η were adjusted according to the cases shown in Table 1. The network was trained through 300 epochs for each of the 5 training sets. In general, all the SSE vaues for the experiments stopped decreasing significantly beyond this point (as seen by Figure 3). Since the program spent approximately 1 second on each epoch, the number of training epochs was kept low enough to conduct experiments for each hidden and output unit configuration within a reasonable amount of time.

Table 1: Parameter Values

d_{hid}	d_{out}	η	c
0	1	5.0e-07	1.040
5	1	6.5e-06	1.035
10	1	6.5e-06	1.035
15	1	6.5e-06	1.035
17	1	6.5e-06	1.035
20	1	5.0e-06	1.040
25	1	5.0e-06	1.040
30	1	5.0e-06	1.040
35	1	5.0e-06	1.040
40	1	5.0e-06	1.040
40	3	6.0e-06	1.040
40	5	6.0e-06	1.040
40	6	6.0e-06	1.040
40	11	6.0e-06	1.040
40	21	6.0e-06	1.040
40	26	6.0e-06	1.040

THE SPECIFIED PARAMETER VALUES USED TO MAKE BACKPROPOGATION CONVERGE TO AN ERROR

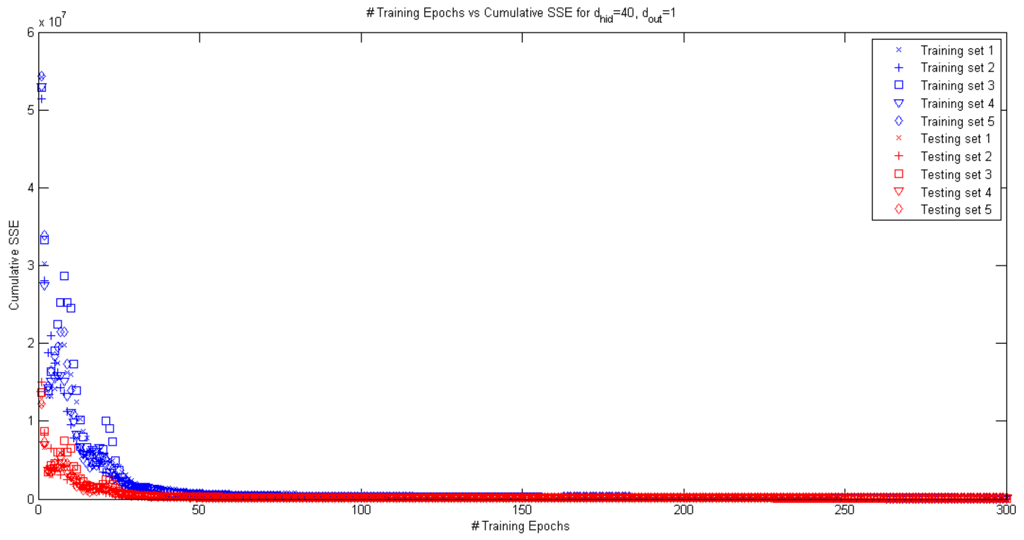
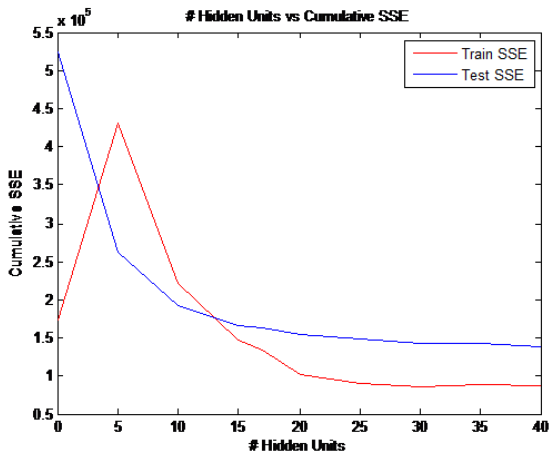
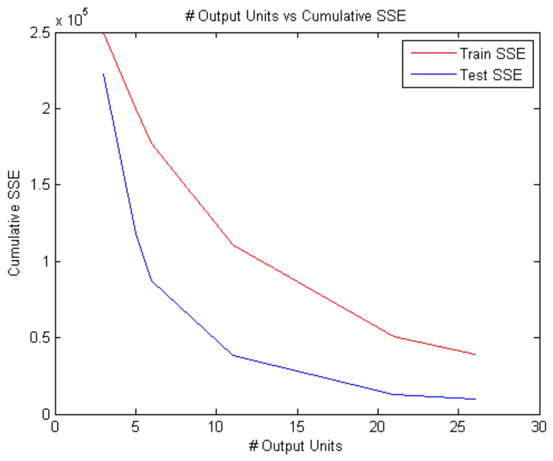


Figure 3: Error decay for this configuration is representative of the error decay for other configurations.



(a) Optimum number of hidden units is 40 for $d_{out} = 1$



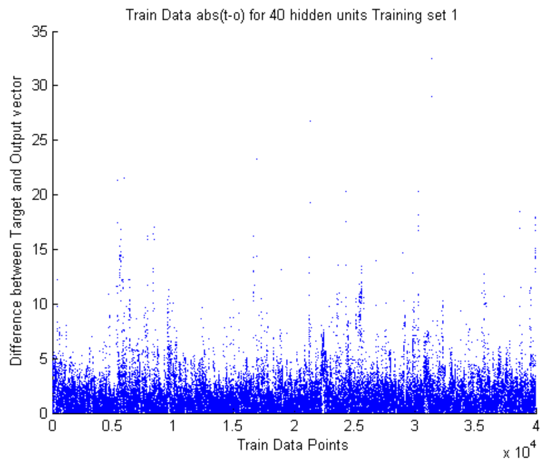
(b) Optimum number of output units is 26 for $d_{hid} = 40$

Figure 4: Optimum d_{hid} and d_{out} values

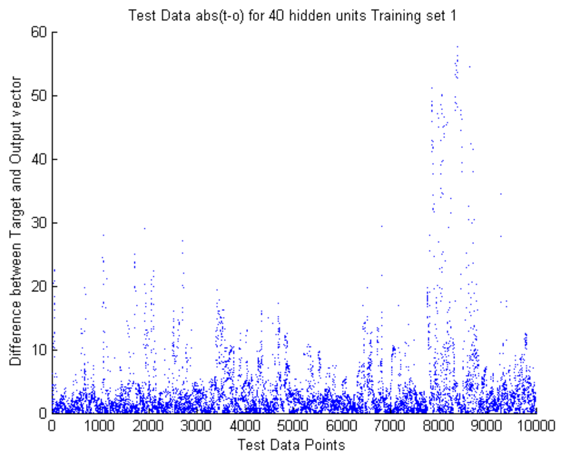
0.5 Results/Discussion

Results from Figure 4(a) indicate that the optimal network consists of 40 hidden units and 1 output unit. The results from Figure 4(b) would indicate that the optimal network consists of 40 hidden units and 26 output units. It can be inferred from Figure 4(a) that the improvement in accuracy of the network is not worth the increase in computational complexity beyond 40 hidden units. We see this same effect in Figure 4(b), as further increase in output units would yield little improvement over accuracy. The cumulative SSE value in Figure 4(b) (for $d_{hid} = 40, d_{out} = 26$) being less than the cumulative SSE value in Figure 4(a) (for $d_{hid} = 40, d_{out} = 1$) would indicate the true optimal network configuration consists of 40 hidden units and 26 output units.

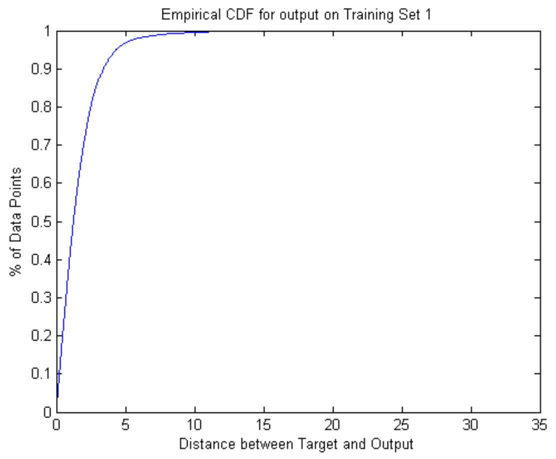
Because the optimal cumulative SSE values in Figure 4 are on the order of 10^5, it becomes necessary to analyze the absolute difference between the target and network output vectors for the training and testing data sets. Figure 5 shows this data ($d_{hid} = 40, d_{out} = 1$) for one of the training and testing subsets (the rest are given in the Appendix), as the results from this Figure are similar to Figures 7,8, and 9. Based on the data, we see that on average the network's predictions on the training data subset are accurate to within 0.91 cm (mean 1.64). It is seen that over 90 percent of the network's predictions on the training data subset are accurate to within 2.8 cm (5 units of distance). Figure 5 (b) also shows promising results, as on average the network's predictions on the testing subset are accurate to within 1.7 cm (mean 3.14). We also see that about 90 percent of the network's predictions on the testing data subset are accurate to within 5.6 cm (10 units of distance). It can be seen that the performance of this neural network is comparable to the method in Graf's implementation (90 percent of the predictions were accurate to within 4 cm, the average accuracy was 4.45 cm)[2].



(a) mean 1.64 median 1.24 variance 2.59



(b) mean 3.14 median 1.71 variance 25.9



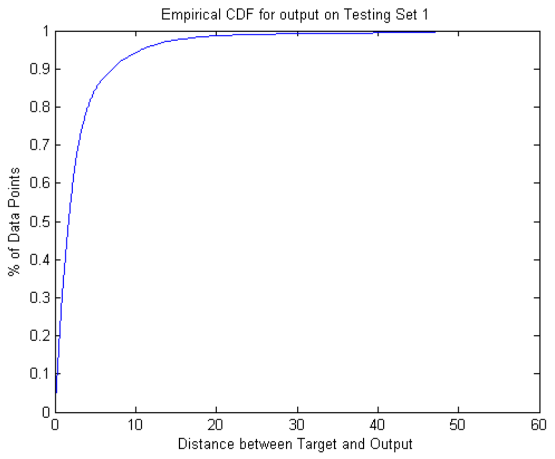
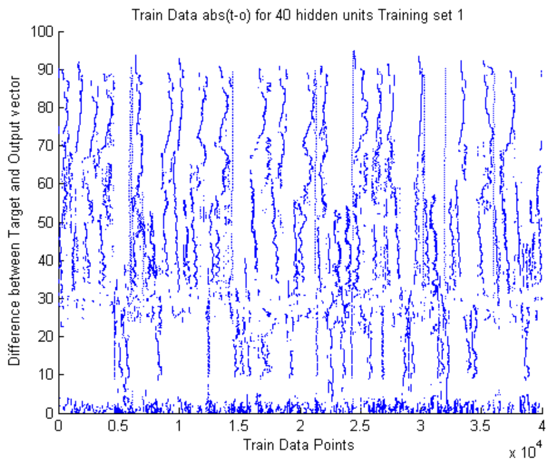
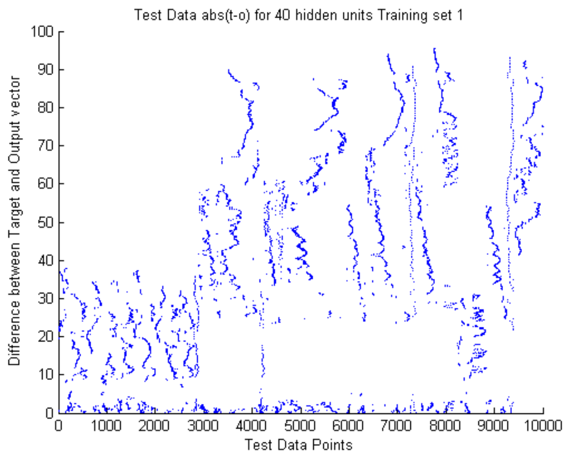


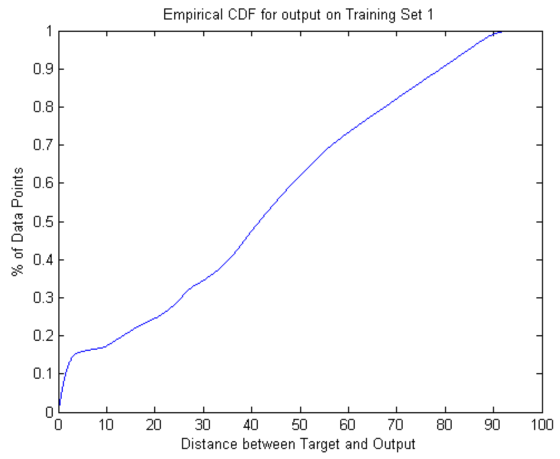
Figure 5: Comparing output and target values on training and testing subset 1 $d_{hid} = 40, d_{out} = 1$ (Representative of the other subset results)



(a) mean 41.3 median 41.6variance 708.6



(b) mean 38.3 median 35.8 variance 683.9



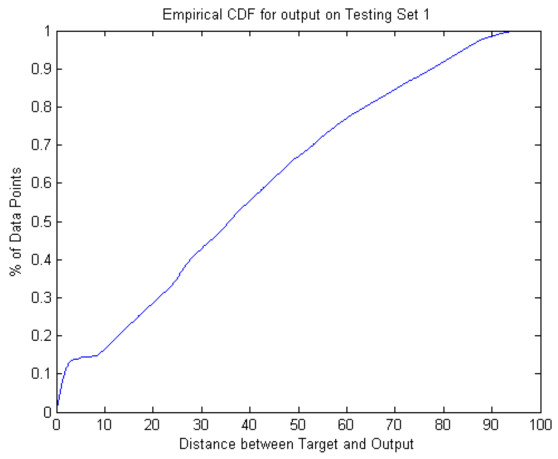


Figure 6: Comparing output and target values on training and testing subset 1 $d_{hid} = 40, d_{out} = 26$ (Representative of the other subset results)

When Figure 6 is analyzed, the performance of the network with 40 hidden units and 26 output units is found to be poor compared to the network with 40 hidden units and 1 output unit. On average, the network's predictions are accurate to within 23 cm (mean 41.3) on the training data subset and accurate to within 21.3 cm (mean 38.3) on the testing data subset. We see that only 15 percent of the network's predictions are accurate to within 5.6 cm (10 units of distance) on the training and testing data subset. A possible explanation for this poor performance could be the possibility that the error function E given in (12) does not correctly model the true difference between the network output values and target values. A weighted sum-squared error would need to be implemented instead of the version given in (12), imposing different multiplication weights on the binary bits corresponding to various locations in the number line:

$$E_{new} = \frac{1}{2} \sum_p \sum_k v_k (t_{pk} - y_{pk})^2 + \frac{1}{2} \gamma \sum w^2$$

For example: $\hat{v} = [0\ 4\ 8\ \ldots\ 96\ 1], \hat{v} \in \mathbb{R}^{26}$ for a network with 26 output units. The cumulative SSE was very large because even though the error was small for individual predictions, these errors aggregated over the 10,000 or 40,000 data points

0.6 Conclusion

In this project we demonstrated that we can use an eager learning approach such as utilizing neural networks to solve a regression problem with a reasonable amount of accuracy. We used Graf's extraction scheme on a data set of CT scans and were able to make predictions on the locations of CT scans (along the vertical axis of the body) within an accuracy on the same order of magnitude as Graf's method [2]. Even though increasing the number of output units dramatically increased the network's performance in terms of cumulative SSE, the network with more output units made poor predictions because the error function for multiple output units was not modeled correctly. This project can be further expanded by implementing the correct error function suggested in the end. Further analysis can be done by analyzing the weight values to see which input attributes significantly affect the predictions of the network.

0.7 Acknowledgements

This project was made possible by ideas from Prof. David C. Noelle and Prof. Harish Bhat.

0.8 Appendix

The neural network and its backpropogation algorithm for this project were written in MATLAB. The user runs the main program 'neural_net_projTest.m' in the MATLAB console window, making sure the CSV file associated with this problem is in the same directory as the main program and the attribute names on the top line are deleted. The program builds, runs, and learns a neural network to solve the regression problem associated witht this project. The program 'neural_net_projTest.m' makes a call to 'g_func.m,' which computes the function representing the output function of a given perceptron in the network. The neural network is 'learned' through the backpropogation algorithm, which is called in 'backpropogation.m.'

The program 'backpropogation.m' uses the error function in 'err.m' and makes a call to 'g_func_prime.m,' which computes the function representing the derivative of 'g_func.m.' When using multiple output units, the program also makes calls to 'g_func_linear.m' and 'g_func_linear_prime.m' to simulate the linear output units. The program takes in the configuration file (found in
http://archive.ics.uci.edu/ml/datasets/Relative+location+of+CT+slices+on+axial+axis, but the user must delete the top line from the CSV file (attribute names)), and the program prints the sum-squared errors over the training data set and testing data set over the number of epochs specified. Note: the code for this project is heavily based on the code implemented for Programming Assignment 4. The MATLAB files are given in the following page. To adjust the number of hidden units or output units in the program, the user modifies d_hid and partition (in factors of 100) in 'neural_net_projTest.m.' Other parameters can also be adjusted in the same file to accomodate for different experiments.

neural_net_projTest.m

```
clear all;

partition=1;
discretize=100/partition;%discretize is integer
dimen=385;

d_in=dimen-1;
if partition==1
    d_out=1;
else
    d_out=partition+1;
end
d_hid=40;
a_min=-.5;%a_min offset for g function
a_max=.5;%a_max offset for g function
w_range=.5;%range of the network weights
eta=.000005;%learning rate
alpha=0.0000001;%momentum term as seen in equation 4.18 of Tom Mitchell's Machine Learni
gamma=0.0000000001;%added penalty term to SSE function
T_total=100;%# total epochs to learn over
T_report=2;%report error every partition of iterations
n_train=40000;
n_test=10000;
num_vecs=53500;

train_pattern_set=struct('input_vecs',zeros(d_in,n_train),'output_vecs',zeros(d_out,n_tr
test_pattern_set=struct('input_vecs',zeros(d_in,n_test),'output_vecs',zeros(d_out,n_test

arr=csvread('slice_localization_data_use.csv');

arr=arr(1:n_train+n_test,2:end);

b=mod(arr(:,end),discretize);
a=(arr(:,end)-b)/discretize;
if partition>1
    c=zeros(size(arr,1),partition+1);
    for i=1:size(arr,1)
        c(i,a(i)+1)=1;
    end
else
    c=zeros(size(arr,1),partition);
end
c(:,end)=b;
```

```
arr=[arr(:,1:end-1) c];

for i=1:1
    test_pattern_set.input_vecs=arr((n_test*(i-1)+1):n_test*i,1:d_in)';
    test_pattern_set.output_vecs=arr((n_test*(i-1)+1):n_test*i,d_in+1:d_in+d_out)';
    train_pattern_set.input_vecs=[arr(1:(n_test*(i-1)),1:d_in); arr((n_test*(i)+1):end,1
    train_pattern_set.output_vecs=[arr(1:(n_test*(i-1)),d_in+1:d_in+d_out); arr((n_test*
    %no hidden units

    if d_hid>0
        w_hidden= -w_range/2+ w_range*rand(d_in,d_hid);%generate d_in by d_out matrix of
        w_output= -w_range/2+ w_range*rand(d_hid,d_out);

        w_hidden_bias=-w_range/2+ w_range*rand(d_hid,1);
        w_output_bias=-w_range/2+ w_range*rand(d_out,1);

        delta_w_hidden=zeros(d_in,d_hid)';
        delta_w_output=zeros(d_hid,d_out)';
        delta_w_hidden_bias=zeros(d_hid,1);
        delta_w_output_bias=zeros(d_out,1);
    %hidden units exist
    else
        w_hidden=[];
        w_output=-w_range/2+ w_range*rand(d_in,d_out);

        w_hidden_bias=[];
        w_output_bias=-w_range/2+ w_range*rand(d_out,1);

        delta_w_hidden=[];
        delta_w_output=zeros(d_in,d_out)';
        delta_w_hidden_bias=[];
        delta_w_output_bias=zeros(d_out,1);
    end

    t=train_pattern_set.output_vecs;
    t_test=test_pattern_set.output_vecs;

    s=train_pattern_set.input_vecs;
    s_test=test_pattern_set.input_vecs;
    siz_s=size(s);
    alpha=0.000001;
    fprintf('Train iteration %d\n',i);
    for j=1:T_total

        %run neural network over training and test data
        [y,z,x]=neural_net(s,w_hidden,w_output,w_hidden_bias,w_output_bias,a_min,a_max);
        [y_test,z_test,x_test]=neural_net(s_test,w_hidden,w_output,w_hidden_bias,w_outpu

        %produce error values
        err_train=err(y,t,w_output,w_output_bias,w_hidden,w_hidden_bias,gamma);
        err_test=err(y_test,t_test,w_output,w_output_bias,w_hidden,w_hidden_bias,gamma);


        %use network to determine delta w,
        %delta_w(t)=delta_w(t)+eta*network_partial_derivative

        %update w with delta w, wkj=wkj+delta_w
        %update w with delta w, wji=wji+delta_w
        [w_output,delta_w_output,w_hidden,delta_w_hidden,w_output_bias,delta_w_output_bi

        if mod(j,T_report)==0
            fprintf('EPOCH \t%d:\tTRAIN SSE=\t%f;\tTEST SSE=\t%f\n',j,err_train,err_test
        end
        alpha=alpha*1.04;
    end
    fprintf('\n');
    err_avg_train(i)=err_train;
    err_avg_test(i)=err_test;
end
```

neural_net.m

```
%This program constructs the neural network based on the appropriate
%weights and runs the neural network on the given data set to produce
%output vectors

%x has input vectors, y has output vectors, and z has hidden unit values
%This models diagram in lecture slides '14-bp.pdf'
function [y,z,x]=neural_net(x,w_hidden,w_output,w_hidden_bias,w_output_bias,a_min,a_max)
    %no hidden nodess
    if isempty(w_hidden)
        if size(w_output,2)==1
        %implement equations (2) and (3) over all nodes in report.pdf
        x_sz=size(x);
        temp=repmat(w_output_bias,1,x_sz(2));
        net_out_vec=w_output'*x+temp;

            y=g_func_linear(net_out_vec);

        z=[];
        else
            x_sz=size(x);
        temp=repmat(w_output_bias,1,x_sz(2));
        net_out_vec=w_output'*x+temp;

            y=[g_func(0,1,net_out_vec(1:end-1,:));g_func_linear(net_out_vec(end,:))];

        z=[];
        end
    %there are hidden nodes
    else
        if size(w_output,2)==1
        %implement equations (2) and (3) over all nodes in report.pdf
        x_sz=size(x);
        temp=repmat(w_hidden_bias,1,x_sz(2));
        net_hidden_vec=w_hidden'*x+temp;

            z=g_func(a_min,a_max,net_hidden_vec);

        z_sz=size(z);
        temp=repmat(w_output_bias,1,z_sz(2));
        net_out_vec=w_output'*z+temp;

            y=g_func_linear(net_out_vec);
        else
```

```
                x_sz=size(x);
            temp=repmat(w_hidden_bias,1,x_sz(2));
            net_hidden_vec=w_hidden'*x+temp;

                z=g_func(a_min,a_max,net_hidden_vec);

            z_sz=size(z);
            temp=repmat(w_output_bias,1,z_sz(2));
            net_out_vec=w_output'*z+temp;

                y=[g_func(0,1,net_out_vec(1:end-1,:));g_func_linear(net_out_vec(end,:))];
            end
        end
    end
```

backpropogation.m

```
%This implements the backpropogation algorithm along with adding momentum
function [w_output,delta_w_output,w_hidden,delta_w_hidden,w_output_bias,delta_w_output_b

    %no hidden units
    if isempty(w_hidden)

        if size(w_output,2)==1
        %implement equation (22) in report.pdf for all weights
        t_minus_kp_y=t-y;
        siz_x=size(x);
        nets_kp=[w_output' w_output_bias]*[x;ones(1,siz_x(2))];
        g_p_nets_kp=g_func_linear_prime(nets_kp);
        temp=t_minus_kp_y.*g_p_nets_kp*[x' ones(siz_x(2),1)];
        neg_delE_del_wki=temp-gamma*[w_output' w_output_bias];

        else
            %implement equation (22) in report.pdf for all weights
        t_minus_kp_y=t-y;
        siz_x=size(x);
        nets_kp=[w_output' w_output_bias]*[x;ones(1,siz_x(2))];
        g_p_nets_kp=[g_func_prime(0,1,nets_kp(1:end-1,:));g_func_linear_prime(nets_kp(en
        temp=t_minus_kp_y.*g_p_nets_kp*[x' ones(siz_x(2),1)];
        neg_delE_del_wki=temp-gamma*[w_output' w_output_bias];

        end
        w_hidden=[];
        w_hidden_bias=[];
        delta_w_hidden=[];
        delta_w_hidden_bias=[];

        %implement momentum
        delta_w_output=eta*neg_delE_del_wki(:,1:end-1)+alpha*delta_w_output;
        delta_w_output_bias=eta*neg_delE_del_wki(:,end)+alpha*delta_w_output_bias;

        %update weights
        w_output=w_output+delta_w_output';
        w_output_bias=w_output_bias+delta_w_output_bias;
    %there are hidden units
    else
        if size(w_output,2)==1
        %implements equation (14) in report.pdf for all output weights
        t_minus_kp_y=t-y;
        siz_z=size(z);
        nets_kp=[w_output' w_output_bias]*[z;ones(1,siz_z(2))];
        g_p_nets_kp=g_func_linear_prime(nets_kp);%edit here plz
        temp=t_minus_kp_y.*g_p_nets_kp*[z' ones(siz_z(2),1)];
        neg_delE_del_wkj=temp-gamma*[w_output' w_output_bias];

        %implements equation (19) in report.pdf for all hidden weights
        temp=t_minus_kp_y.*g_p_nets_kp;
        siz_x=size(x);
        nets_jp=[w_hidden' w_hidden_bias]*[x; ones(1,siz_x(2))];
        g_p_nets_jp=g_func_prime(a_min,a_max,nets_jp);
        temp=(temp'*w_output')'.*g_p_nets_jp*[x' ones(siz_x(2),1)];
        neg_delE_del_wji=temp-gamma*[w_hidden' w_hidden_bias];
        else
            %implements equation (14) in report.pdf for all output weights
        t_minus_kp_y=t-y;
        siz_z=size(z);
        nets_kp=[w_output' w_output_bias]*[z;ones(1,siz_z(2))];

        g_p_nets_kp=[g_func_prime(0,1,nets_kp(1:end-1,:));g_func_linear_prime(nets_kp(en
        temp=t_minus_kp_y.*g_p_nets_kp*[z' ones(siz_z(2),1)];
        neg_delE_del_wkj=temp-gamma*[w_output' w_output_bias];

        %implements equation (19) in report.pdf for all hidden weights
        temp=t_minus_kp_y.*g_p_nets_kp;
        siz_x=size(x);
        nets_jp=[w_hidden' w_hidden_bias]*[x; ones(1,siz_x(2))];
        g_p_nets_jp=g_func_prime(a_min,a_max,nets_jp);
        temp=(temp'*w_output')'.*g_p_nets_jp*[x' ones(siz_x(2),1)];
        neg_delE_del_wji=temp-gamma*[w_hidden' w_hidden_bias];
        end
        %implement momentum
        delta_w_output=eta*neg_delE_del_wkj(:,1:end-1)+alpha*delta_w_output;
        delta_w_output_bias=eta*neg_delE_del_wkj(:,end)+alpha*delta_w_output_bias;

        delta_w_hidden=eta*neg_delE_del_wji(:,1:end-1)+alpha*delta_w_hidden;
        delta_w_hidden_bias=eta*neg_delE_del_wji(:,end)+alpha*delta_w_hidden_bias;

        %update weights
        w_output=w_output+delta_w_output';
        w_output_bias=w_output_bias+delta_w_output_bias;

        w_hidden=w_hidden+delta_w_hidden';
        w_hidden_bias=w_hidden_bias+delta_w_hidden_bias;
    end
end
```

err.m

```
%This implements the error equation (12) in report.pdf
function [out]=err(y,t,w_output,w_output_bias,w_hidden,w_hidden_bias,gamma)
    out=.5*norm(t-y,'fro')^2+.5*gamma*norm(w_output,'fro')^2+.5*gamma*norm(w_output_bias
end
```

g_func.m

```
%This implements equation (3) from report.pdf and applies to many inputs
function [out]=g_func(a_min,a_max,net)
    siz=size(net);
    out=a_min*ones(siz(1),siz(2))+(a_max-a_min)*(1./(ones(siz(1),siz(2))+exp(-net)));
end
```

g_func_linear.m

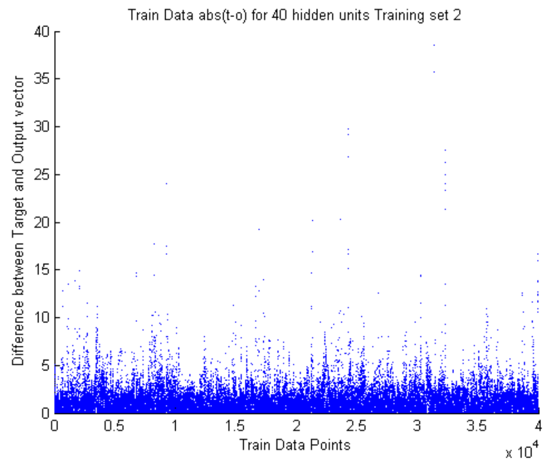
```
function [out]=g_func_linear(net)
    out=net;%return matrix of values
end
```

g_func_prime.m

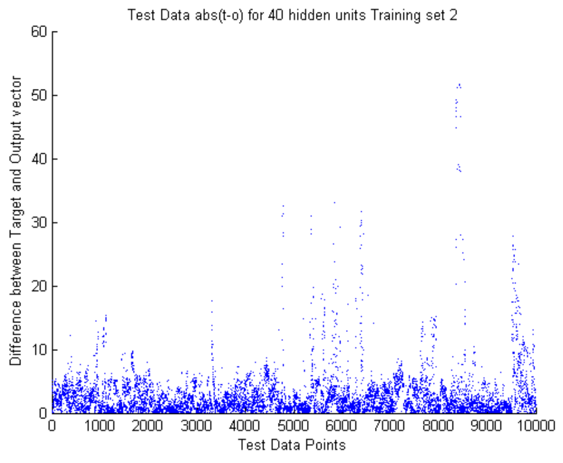
```
%This implements the derivative of g_func, applying to many inputs at once
%(see equation (8) from report.pdf)
function [out]=g_func_prime(a_min,a_max,net)
    siz=size(net);
    out=(g_func(a_min,a_max,net)-a_min*ones(siz(1),siz(2))).*(ones(siz(1),siz(2))-(1/(a_
end
```

g_func_linear_prime.m

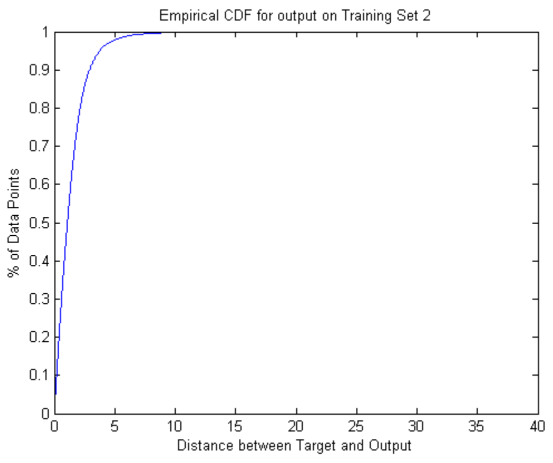
```
function [out]=g_func_linear_prime(net)
    siz=size(net);
    out=ones(siz(1), siz(2));%return matrix of values
end
```



(a) mean 1.41 median 1.09 variance 1.99



(b) mean 2.67 median 1.74 variance 12.7



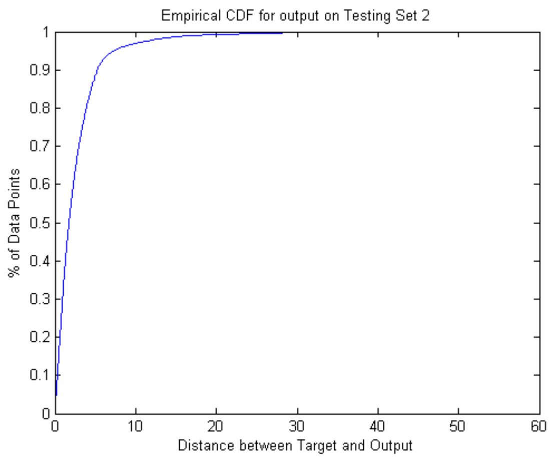
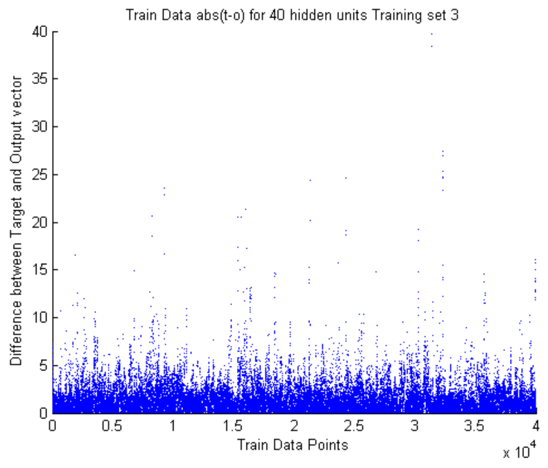
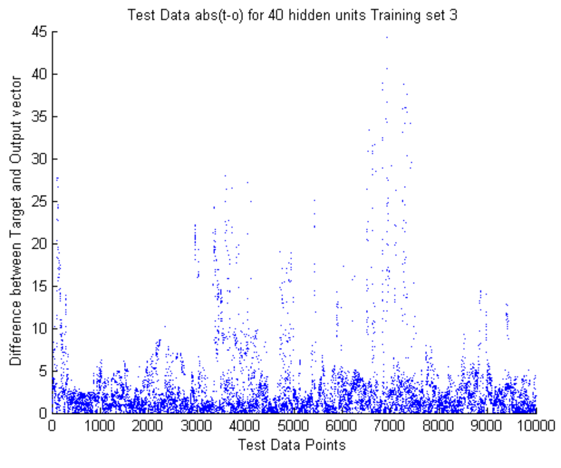


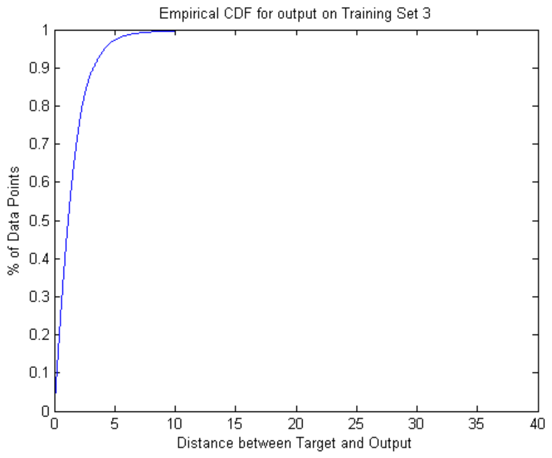
Figure 7: Comparing output and target values on training and testing subset 2 $d_{hid} = 40, d_{out} = 1$



(a) mean 1.52 median 1.15 variance 2.36



(b) mean 2.55 median 1.44 variance 14.4



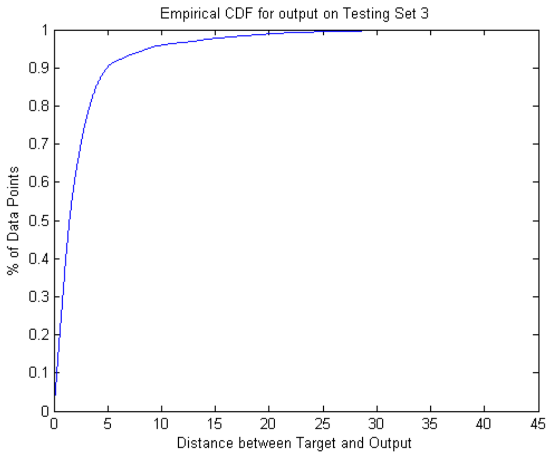
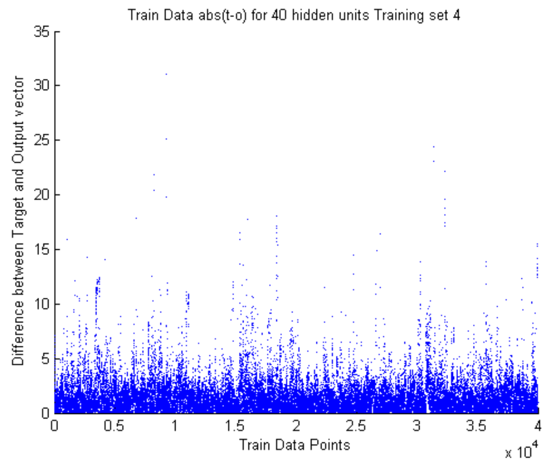
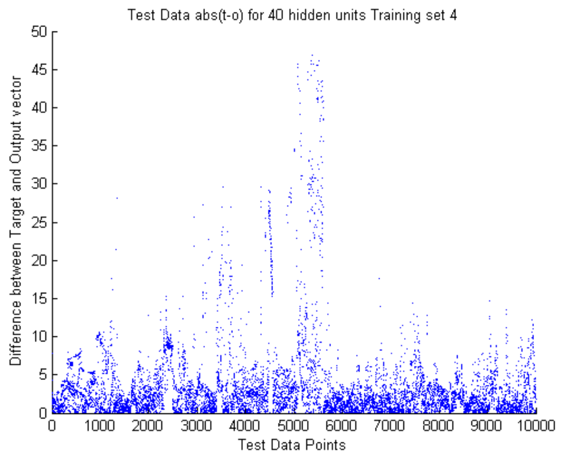


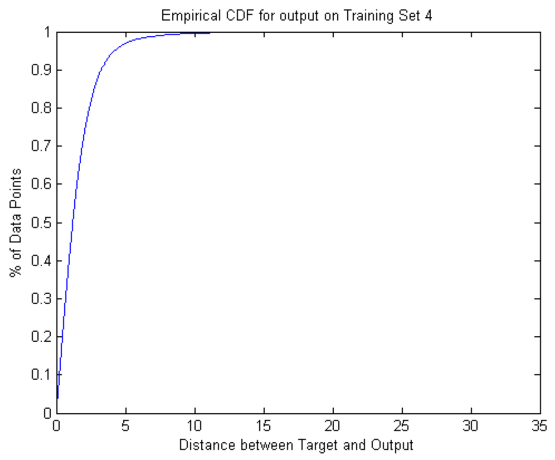
Figure 8: Comparing output and target values on training and testing subset 3 $d_{hid} = 40, d_{out} = 1$



(a) mean 1.57 median 1.19 variance 2.43



(b) mean 3.34 median 1.94 variance 25.7



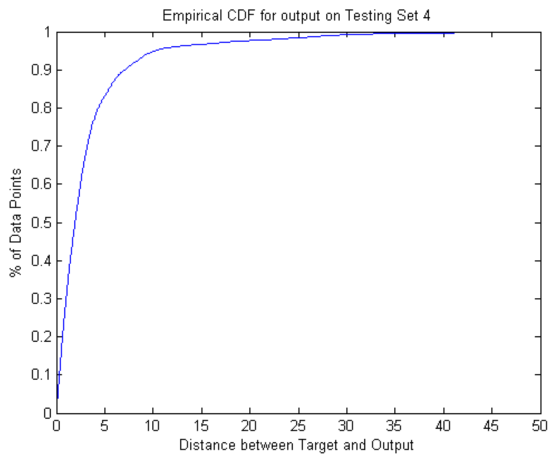
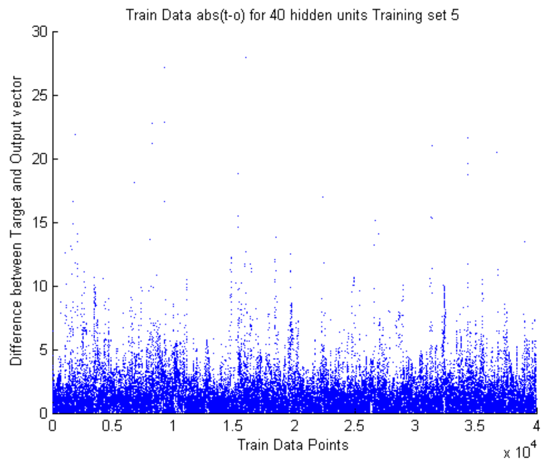
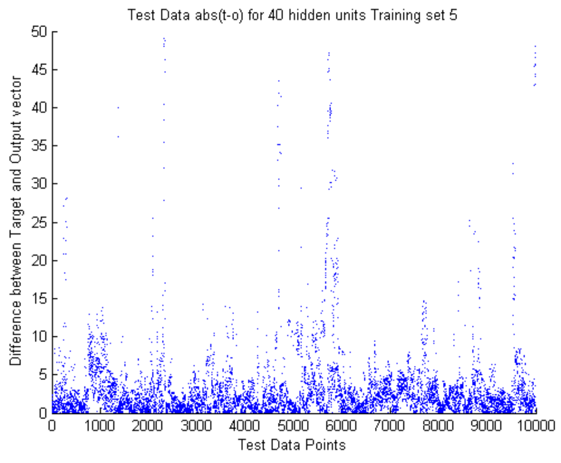


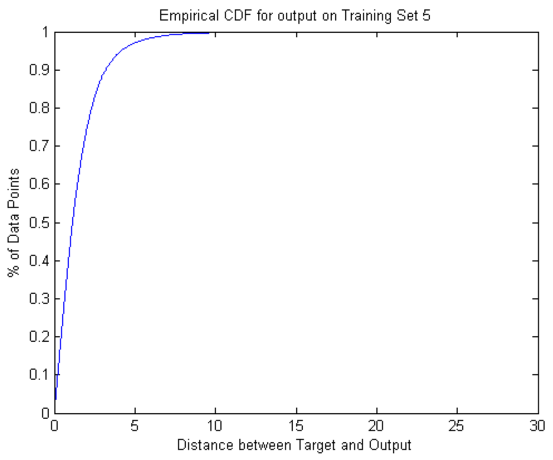
Figure 9: Comparing output and target values on training and testing subset 4 $d_{hid} = 40, d_{out} = 1$



(a) mean 1.51 median 1.14 variance2.18



(b) mean 2.89 median 1.77 variance 20.3



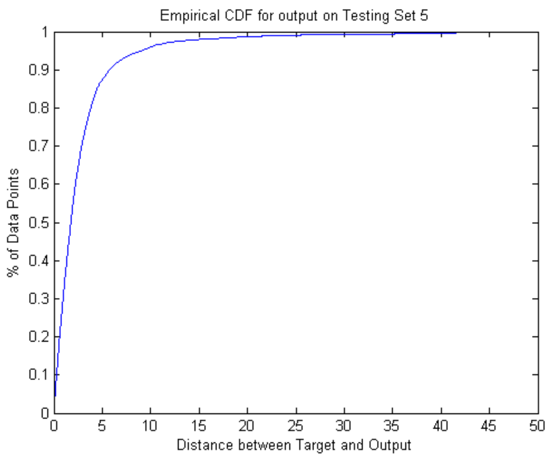
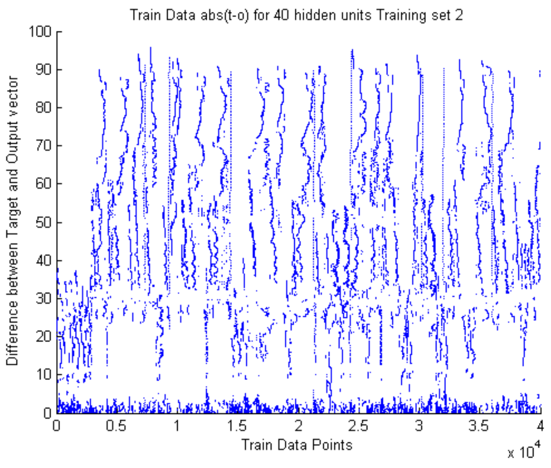
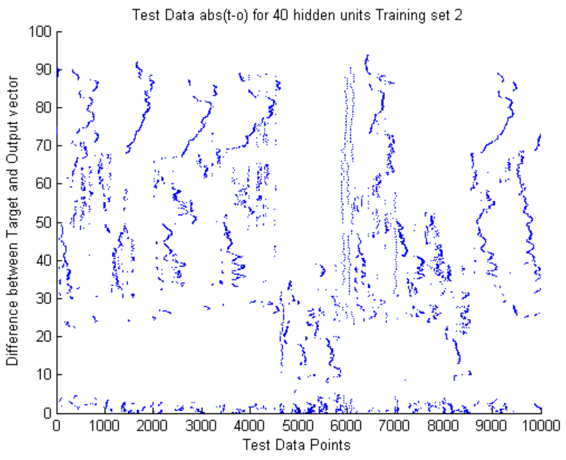


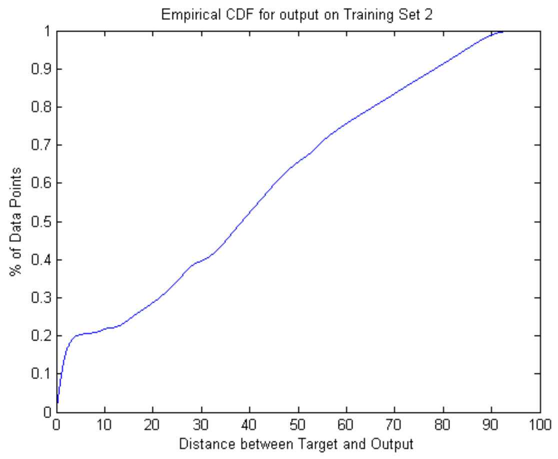
Figure 10: Comparing output and target values on training and testing subset 4 $d_{hid} = 40, d_{out} = 1$



(a) mean 38.6 median 38.5 variance 743.2



(b) mean 45.3 median 45.6 variance 668.2



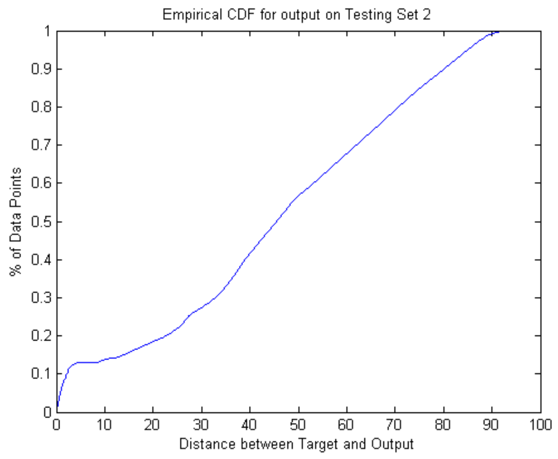
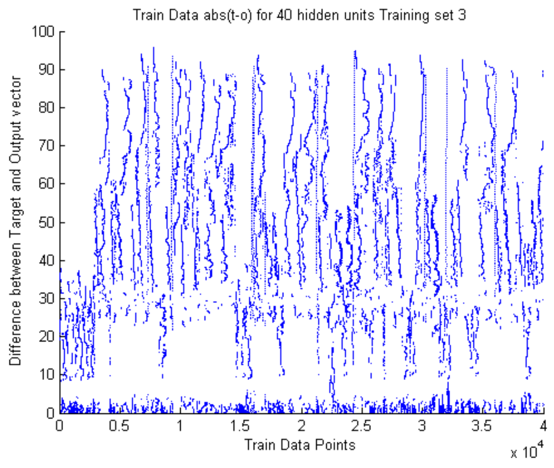
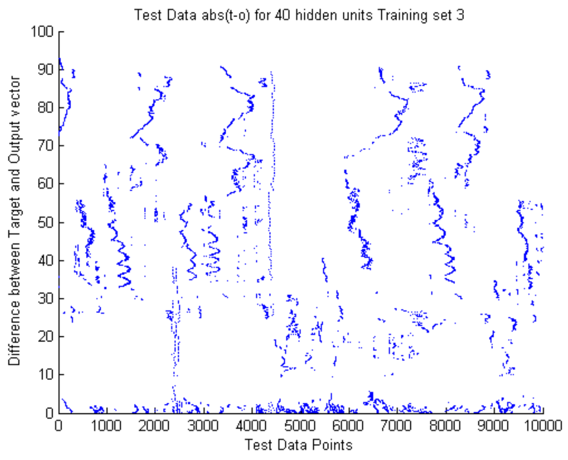


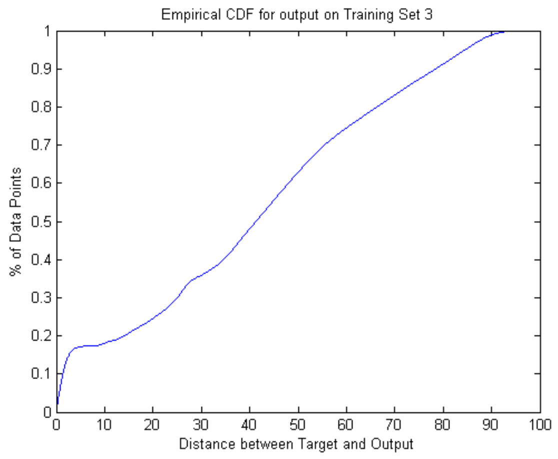
Figure 11: Comparing output and target values on training and testing subset 2 $d_{hid} = 40, d_{out} = 26$



(a) mean 40.7 median 41.3 variance 702.8



(b) mean 40.6 median 41.6 variance 780.3



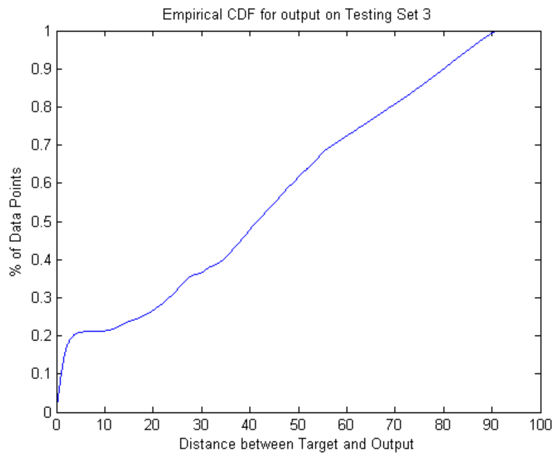
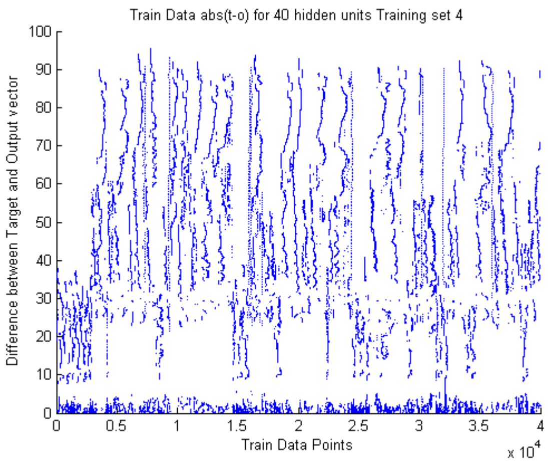
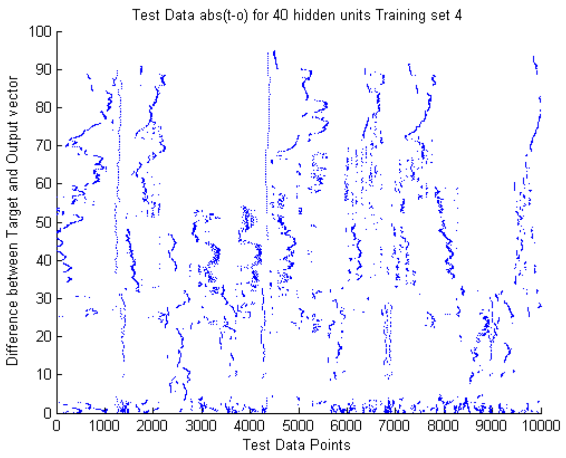


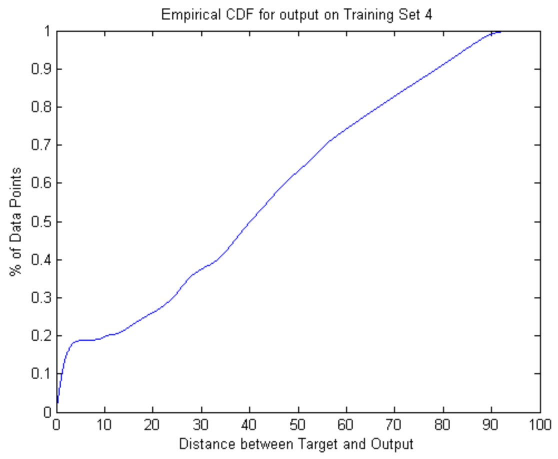
Figure 12: Comparing output and target values on training and testing subset 3 $d_{hid} = 40, d_{out} = 26$



(a) mean 40.1 median 40.2 variance 729.1



(b) mean 40.7 median 40.4 variance 725.2



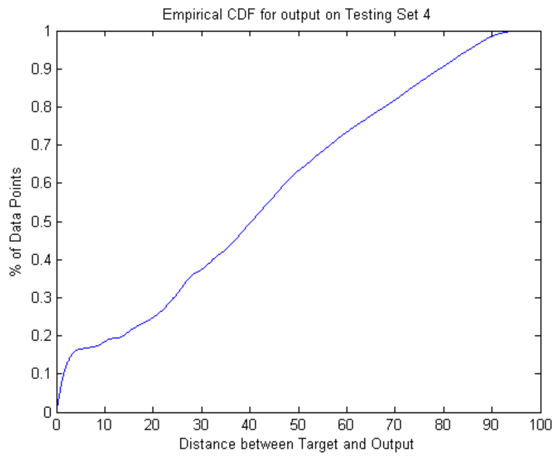
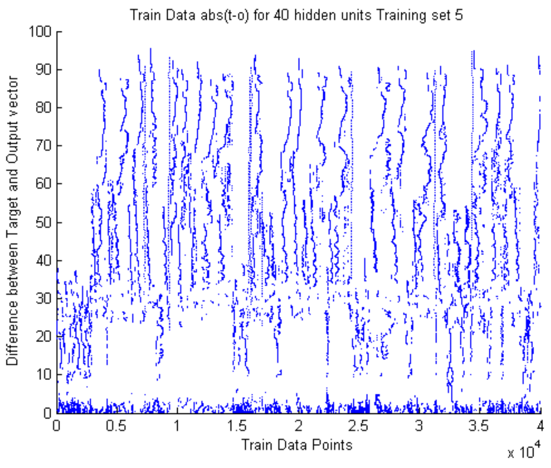
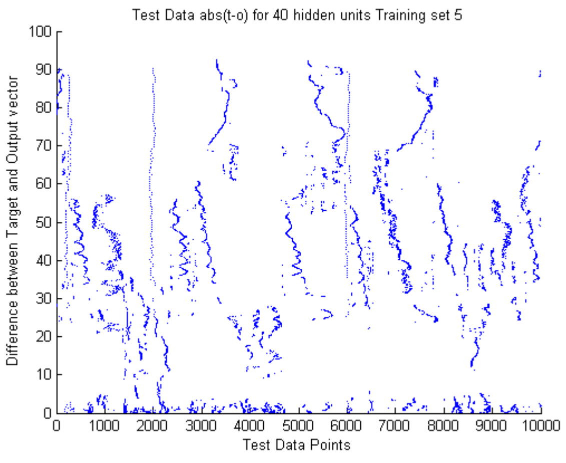


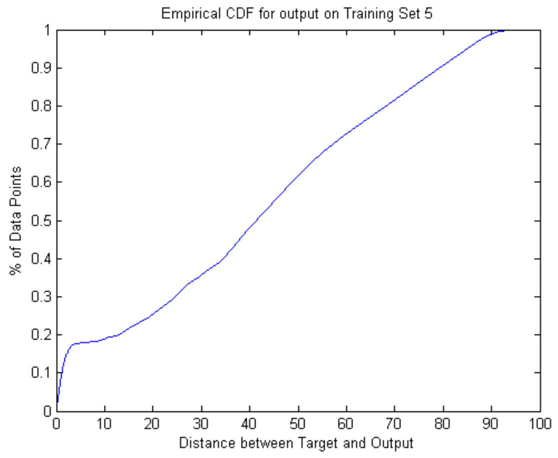
Figure 13: Comparing output and target values on training and testing subset 4 $d_{hid} = 40, d_{out} = 26$



(a) mean 41.1 median 41.5 variance 738



(b) mean 39.1 median 40.1 variance 637



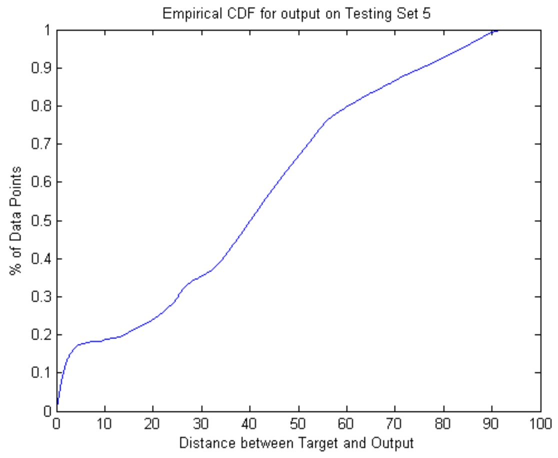


Figure 14: Comparing output and target values on training and testing subset 4 $d_{hid} = 40, d_{out} = 26$

Bibliography

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[2] M. Schubert S. Polesterl A. Cavallaro F. Graf, H.-P. Kriegel. 2d image resgistration in ct images using radial image descriptors. In *In Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Toronto, Canada, 2011.

[3] S. Polsterl M. Schubert A. Cavallaro F. Graf, H.-P. Kriegel. Position prediction in ct volume scans. In *In Proceedings of the 28th International Conference on Machine Learning (ICML) Workshop on Learning for Global Challenges*, Bellevue, Washington, WA, 2011.

[4] Vibhor Jain. report, April 2013. From Programming Assignment 4.

[5] Tom Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[6] David C. Noelle. Backpropogation of error. University Lecture, 2013.