CMPE 545, Artificial Neural Networks

Term Project Report

Çağatay Yıldız - 2014700129

May 25, 2015

1 Time Series Clustering using Self-Organized Maps

In [1], SOM's are used to cluster 49 stocks' prices and gold commodity price for portfolio selection. Their method is based on **component planes**, that they define as *a representation that visualizes relative component values in the weight vectors of the SOM.* This is the same as considering a SOM whose each unit is a vector in N dimension as N different grids(=component planes) in $2D$. Then, clustering these N grids corresponds to clustering components(features) in weight vectors.

1.1 Training Self-Organized Maps

I applied the same procedure as in [1] for clustering outlets so that outlets whose sale numbers behave similarly in time can be grouped. What's done for clustering N time series, each having length T, using a $k \times l$ SOM where each unit is in N dimension is as follows (In my case $N = 340$, $T = 1800$, $k = 20$, $l = 30$)

1. Each value in SOM is initialized to a number between 0 and 1 randomly. Also, each time series is z-normalized. This is because I am interested in how demand to the newspapers in an outlet changes, not the sale numbers themselves.

2. To train the SOM, a vector of dimension N is used. This vector stores the sales of all outlets for a particular day. So, T of such vectors are given to SOM as input in one epoch.

3. At the end of the training, n'th grid of the SOM corresponds to a $k \times l$ representation of n'th time series.

4. After that, the distance between each grid is calculated using **modified** R_v coefficient, which they define as *a similarity coefficient between positive semi-definite matrices.* The choice of modified R_v coefficient is because traditional R_v coefficient deteriorates as maps get larger. Similarly in [2], the article in which modified R_v coefficient is presented, it is noted that R_v coefficient of completely random matrices increases as the size of matrices increases.
   Given two matrices C_1 and C_2, modified R_v coefficient is calculated in two steps:

   - First, $T_i = C_i C_i^T - \mathrm{diag}(C_i C_i^T)$ is calculated for $i = 1, 2$.
   - Then, the similarity measure is

$$\sigma(C_1, C_2) = \frac{\mathrm{trace}\{T_1^T T_2\}}{\sqrt{\mathrm{trace}\{T_1^T T_2\} \times \mathrm{trace}\{T_1^T T_2\}}} \tag{1}$$

1.2 Hierarchical Clustering

Similarity matrix of component planes at the hand, hierarchical clustering is then used to complete the procedure. I implemented single linkage clustering and average linkage clustering. In addition to the real data, I generated very simple synthetic data. You can see performances of both approaches in training data in Fig.1.

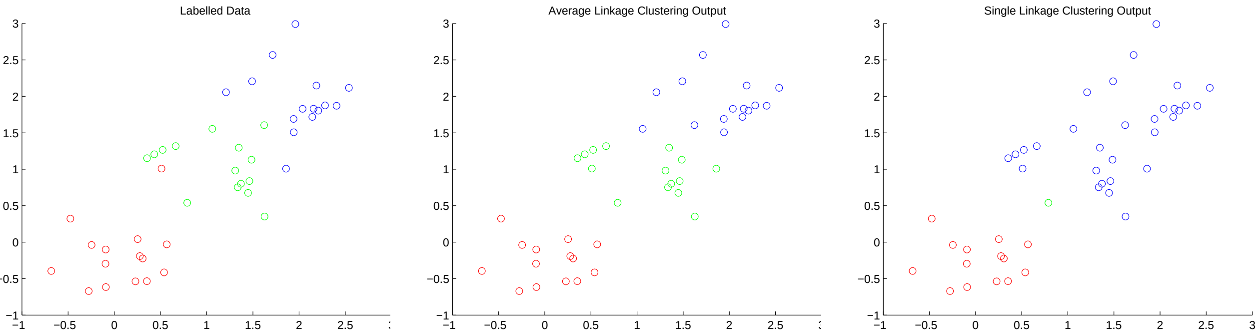

Figure 1: In this simple example, data from 3 different Gaussians centered at (0,0), (1,1) and (2,2) with standard deviation 0.4 is generated. Average linkage approach mislabels some of the data points but this is only because these points are indeed closer to another cluster center than the cluster center from which they are generated. As expected, single linkage approach performs pretty bad in such a data set, which is of course because of the nature of the data.

In Fig.2, you can see the visualization of component planes and sale numbers of randomly chosen 40 outlets. As noted before, we hoped heat maps of two outlets sharing similar trends to be as close as possible, which can roughly be confirmed looking at the figures.

One issue with hierarchical clustering is when to stop merging clusters. Starting with N data points, theoretically it is possible to have any number of clusters from N to 1 at the end of clustering. In this project, I went until merging all clusters to a giant one and recorded cluster labels of outlets at each time. That is, there are N different clustering of the data.
In Fig.3, you can see sale figures of outlets randomly sampled from 4 different clusters. When this figure was drawn, the number of clusters was pulled down to $k = 10$. five of them were single data points, one of them contained 2 data points, three of them had 4-5 data points and the rest belonged to one cluster.

2 Time Series Prediction

The other task is to implement ANN's that learn those series and make predictions. For such purposes, recurrent neural networks are used. Two of the most popular RNN's are Elman network [4] and Jordan network [3]. In this project, I have implemented

- Tapped-delay multilayer perceptron, denoted as **TDMLP** in this report.
- Elman RNN, denoted as **EN** in this report.
- Full lag Elman RNN, denoted as **FLEN** in this report.
- Tapped-delay Jordan RNN, denoted as **TDJN** in this report.

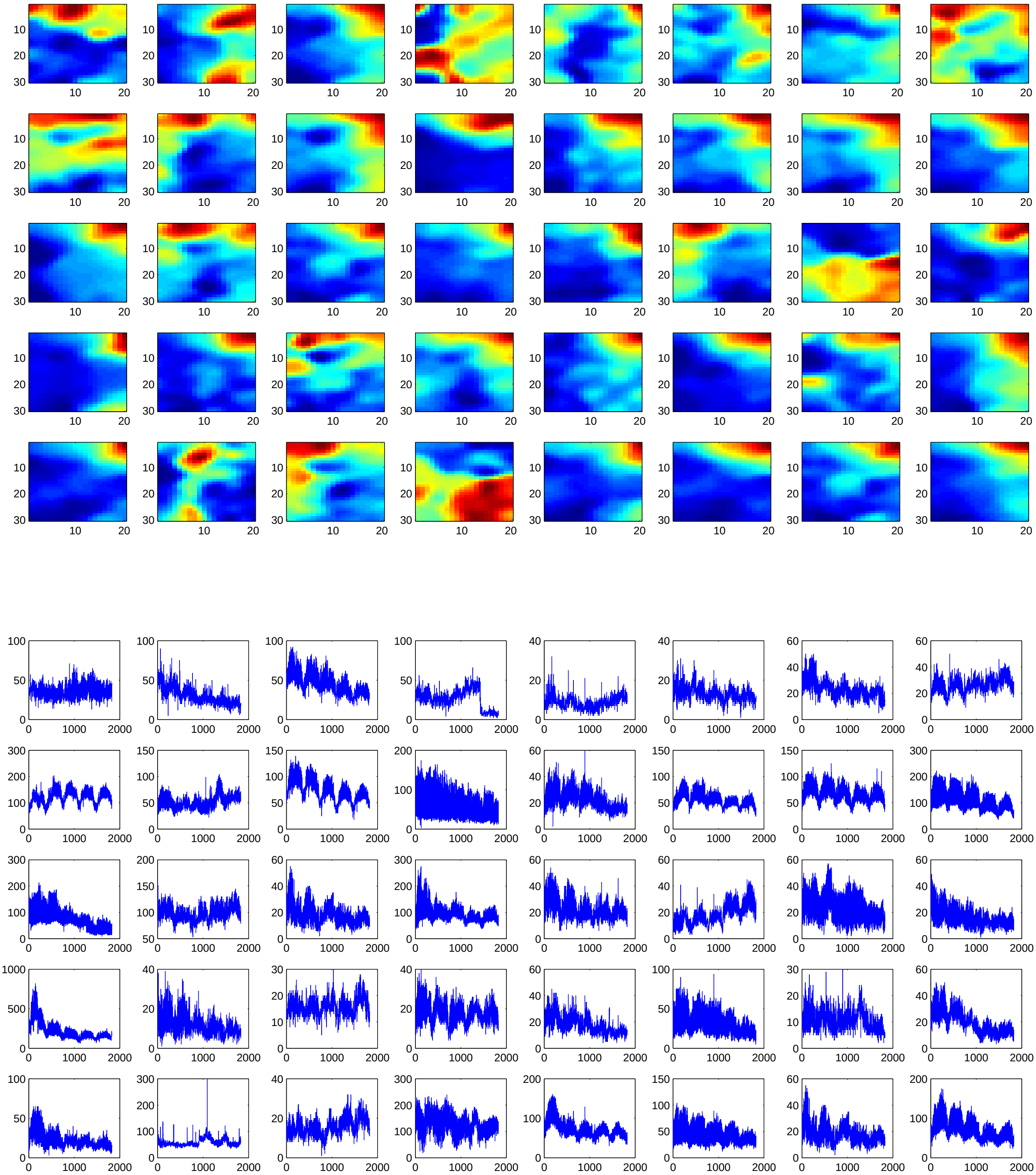

Figure 2: Heat maps and sale figures of randomly chosen 40 outlets. Maps and plots that are at the same coordinates in each subfigure belongs to the same outlet. For instance, sale plots at position (4,1) looks very much like that at (5,5) (all in MATLAB's matrix notation). As expected, corresponding heat maps are very alike, meaning that the distance between these component planes is small.

2.1 Details of Models

In this section, structures of models are explained. You can see the drawings of models in Fig.4
Here is the list of all parameters used in networks:

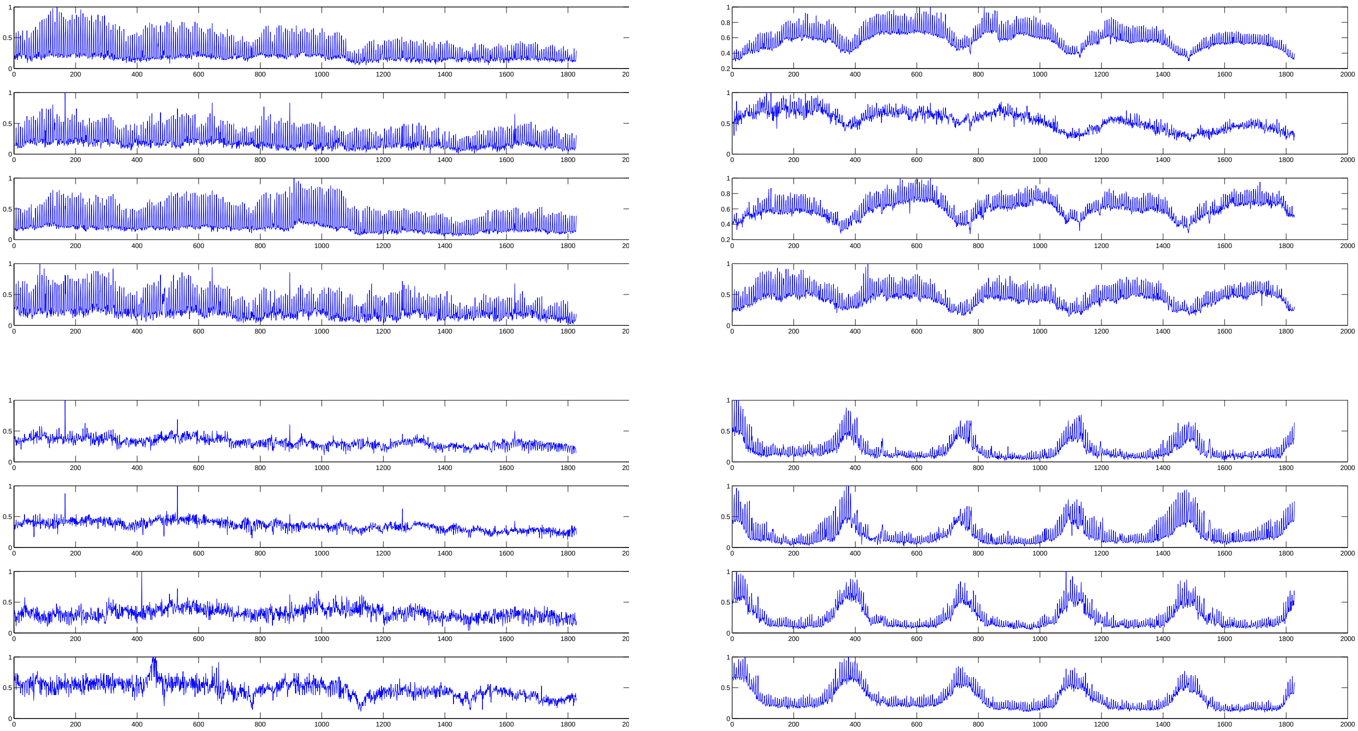

Figure 3: Sale plots of outlets in 4 different clusters (out of $k = 10$ total clusters). Single linkage clustering was used to merge clusters.

- t: Time index
- H: Number of hidden units
- D: Dimensionality of the input
- K: Time lag (for instance $K = 7$ in newspaper sale problem)
- z_i^t: Value of the i'th hidden unit at time t
- c_i^t: Value of the i'th context unit at time t
- s_i^t: Value of the i'th state unit at time t
- W^{oh}: Weights between output and hidden unit
- W^{hi}: Weights between hidden unit and input unit
- W^{hc}: Weights between hidden unit and context unit
- W^{hs}: Weights between hidden unit and state unit

2.1.1 Tapped-Delay Multilayer Perceptron

In tapped-delay MLP, for each time t, data in a window of length K is given as the input. At time $t+1$, window is slid one unit on the data. This way, input is hoped to contain all the information that are

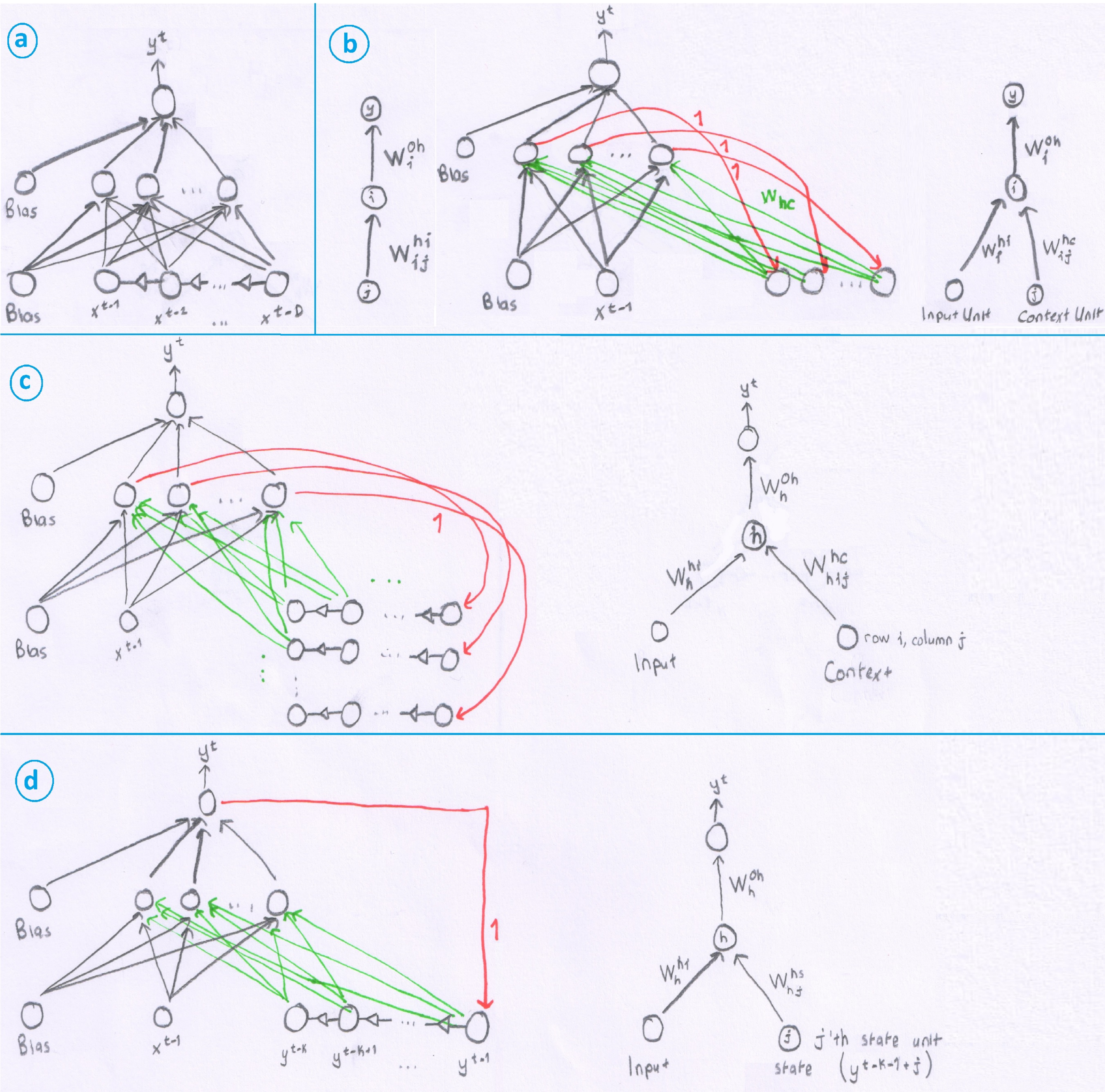

Figure 4: Structures of all network types that are implemented. (a)Tapped-delay multilayer perceptron, (b)Elman neural network, (c) Full lag Elman RNN, (d)Tapped-delay Jordan RNN

enough to predict current value in time series.

Update Equations:

$$
\begin{aligned}
z_i^t &= sigmoid\Big(\sum_{j=1}^{D} W_{ij}^{hi} x^{t-j} + W_{i0}^{hi}\Big) \\
y^t &= \sum_{h=1}^{H} W_h^{oh} z_h^t + W_0^{oh} \\
\Delta W_{ij}^{hi} &= \eta(x^t - y^t) W_i^{oh} z_i^t (1 - z_i^t) x^{t-j} \\
\Delta W_{i0}^{hi} &= \eta(x^t - y^t) W_i^{oh} z_i^t (1 - z_i^t) \\
\Delta W_h^{oh} &= \eta(x^t - y^t) z_h^t \\
\Delta W_0^{oh} &= \eta(x^t - y^t)
\end{aligned}
$$

2.1.2 Elman Recurrent Neural Network

In Elman network, there is so-called *context layer* in addition to input, hidden and output layers. Recurrency in this network occurs thanks to the cycles between hidden layer and context layer: Each context unit output is inputted to all hidden units (with different weights) and the output of hidden layer is 1:1 connected to the context layer ($c_i^{t+1} \leftarrow h_i^t$, $\forall i \in \{1, H\}$ where c and h represents context and hidden units and t is the time index and H is the number of hidden units.).
Update Equations:

$$
\begin{aligned}
z_i^t &= sigmoid\Big(W_i^{hi} x^{t-1} + W_0^{hi} + \sum_{j=1}^{H} W_{ij}^{hc} c_j^t\Big) \\
y^t &= \sum_{h=1}^{H} W_h^{oh} z_h^t + W_0^{oh} \\
\Delta W_i^{hi} &= \eta(x^t - y^t) W_i^{oh} z_i^t (1 - z_i^t) x^{t-1} \\
\Delta W_0^{hi} &= \eta(x^t - y^t) W_i^{oh} z_i^t (1 - z_i^t) \\
\Delta W_{ij}^{hc} &= \eta(x^t - y^t) W_i^{oh} z_i^t (1 - z_i^t) c_j^t \\
\Delta W_h^{oh} &= \eta(x^t - y^t) z_h^t \\
\Delta W_0^{oh} &= \eta(x^t - y^t)
\end{aligned}
$$

2.1.3 Full Lag Elman Recurrent Neural Network

FLEN is a variant of Elman networks. In this architecture, context layer stores not just h_i^{t-1}, $\forall i \in \{1, H\}$ but h_i^{t-k}, $\forall i \in \{1, H\}$ and $\forall k \in \{1, K\}$. As time ticks to $t+1$, values in context units are shifted in such a way that h_i^{t-K}, $\forall i \in \{1, H\}$ leaves the context layer and h_i^t, $\forall i \in \{1, H\}$ is included.
Consider c as a $H \times K$ matrix whose j'th column stores hidden layer values at time $t - K - 1 + j$. For example, for $K = 4$, the first column stores hidden layer values at time $t - 4$, second column stores those at time $t - 3$, etc. In this representation, the following assignments are done:

$$
\begin{aligned}
&c_{i,j}^{t+1} \leftarrow c_{i,j+1}^t \ \forall i \in \{1, H\} \ \forall j \in \{1, H-1\} \\
&c_{i,K}^{t+1} \leftarrow h_i^t \ \forall i \in \{1, H\}
\end{aligned}
$$

Update Equations:

$$
\begin{aligned}
z_i^t &= sigmoid\Big(W_i^{hi}x^{t-1} + W_0^{hi} + \sum_{k=1}^{K}\sum_{j=1}^{H} W_{ijk}^{hc}c_{jk}^t\Big) \\
y^t &= \sum_{h=1}^{H} W_h^{oh}z_h^t + W_0^{oh} \\
\Delta W_i^{hi} &= \eta(x^t - y^t)W_i^{oh}z_i^t(1-z_i^t)x^{t-1} \\
\Delta W_0^{hi} &= \eta(x^t - y^t)W_i^{oh}z_i^t(1-z_i^t) \\
\Delta W_{ijk}^{hc} &= \eta(x^t - y^t)W_i^{oh}z_i^t(1-z_i^t)c_{jk}^t \\
\Delta W_h^{oh} &= \eta(x^t - y^t)z_h^t \\
\Delta W_0^{oh} &= \eta(x^t - y^t)
\end{aligned}
$$

2.1.4 Tapped-Delay Jordan Recurrent Neural Network

In Jordan RNN, the additional layer is called *state layer*. Just like in Elman network, recurrency is done by adding cycles between hidden layer and state layer. Each state unit output is inputted to all hidden units. But in contrast to Elman network, state unit values at time $t+1$ are equal to the outputs at time t. Therefore, there are as many units in the state layer as the number of outputs.
Since there is only one output in my problem, Jordan network seems unpromising. To overcome this issue, I set the size of state unit as K. At time t, state layer stores $y^{t-1:t-K}$. As time ticks, window on outputs is slid one unit forward and values in state layer becomes $y_{t:t-K+1}$.
Update Equations:

$$
\begin{aligned}
z_i^t &= sigmoid\Big(W_i^{hi}x^{t-1} + W_0^{hi} + \sum_{j=1}^{H} W_{ij}^{hs}s_j^t\Big) \\
y^t &= \sum_{h=1}^{H} W_h^{oh}z_h^t + W_0^{oh} \\
\Delta W_i^{hi} &= \eta(x^t - y^t)W_i^{oh}z_i^t(1-z_i^t)x^{t-1} \\
\Delta W_0^{hi} &= \eta(x^t - y^t)W_i^{oh}z_i^t(1-z_i^t) \\
\Delta W_{ij}^{hs} &= \eta(x^t - y^t)W_i^{oh}z_i^t(1-z_i^t)s_j^t \\
\Delta W_h^{oh} &= \eta(x^t - y^t)z_h^t \\
\Delta W_0^{oh} &= \eta(x^t - y^t)
\end{aligned}
$$

2.2 Data Sets Used

I divided the data set (that is mentioned in previous section) into two parts(about 1200 data points in the first set and 600 in the second one), used the first one for training and the second for testing.
In addition, I have generated a number of time series and compared performances of the models. While generating data, I defined various functions where x^t is a nonlinear function of $x^{t-1:t-L}$ for a fixed lag L and a Gaussian noise added.

3 Results

For both synthetic and real data, in order to get the highest accuracy, I trained each model with different hyperparameters and then picked the best hyperparameters while comparing models. One interesting finding is that the change in the number of hidden units does not alter the convergence of error in the long run very much. You can see the plots in Fig.6.

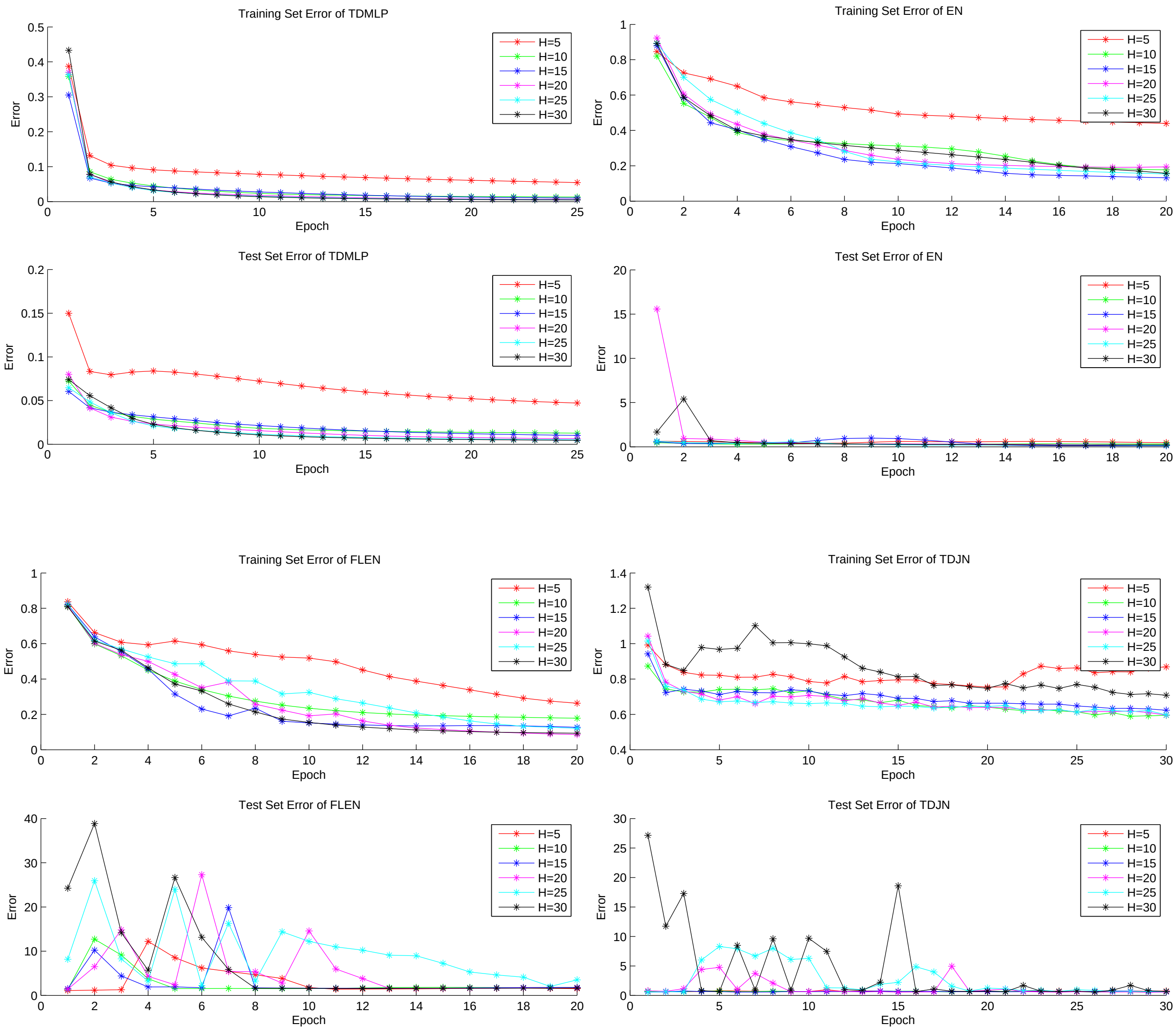

Figure 6: Error figures for each method and different number of hidden units. Data set for this plots was generated by the function f = @(x)(x(1)*x(2)*0.3 - abs(x(4)) - sin(pi*x(3)) + 1) but all the functions I have tried yielded similar plots.

3.1 Synthetic Data Sets

- I did have hard time to generate time series. Most of my attempts yielded series that converge to the same number. Here, I have incorporated three pairs of error plots, each shows the results gathered from different time series. You can see those in Figures 7, 8 and 9.
- Figure 8 indicates that error rate of TDJN fluctuates so much. So, this learner seems to be problematic.

- Validation set errors converge to the same value in figures 8 and 9. That is somewhat a surprising finding and my best guess is it may be because of the nature of the data set.
- Among all approaches, TDMLP has the lowest overall error rate.
- In contrast to what we had encountered in homework, training error may make sudden increases during the application of a method to a time series. This is something that I cannot explain.

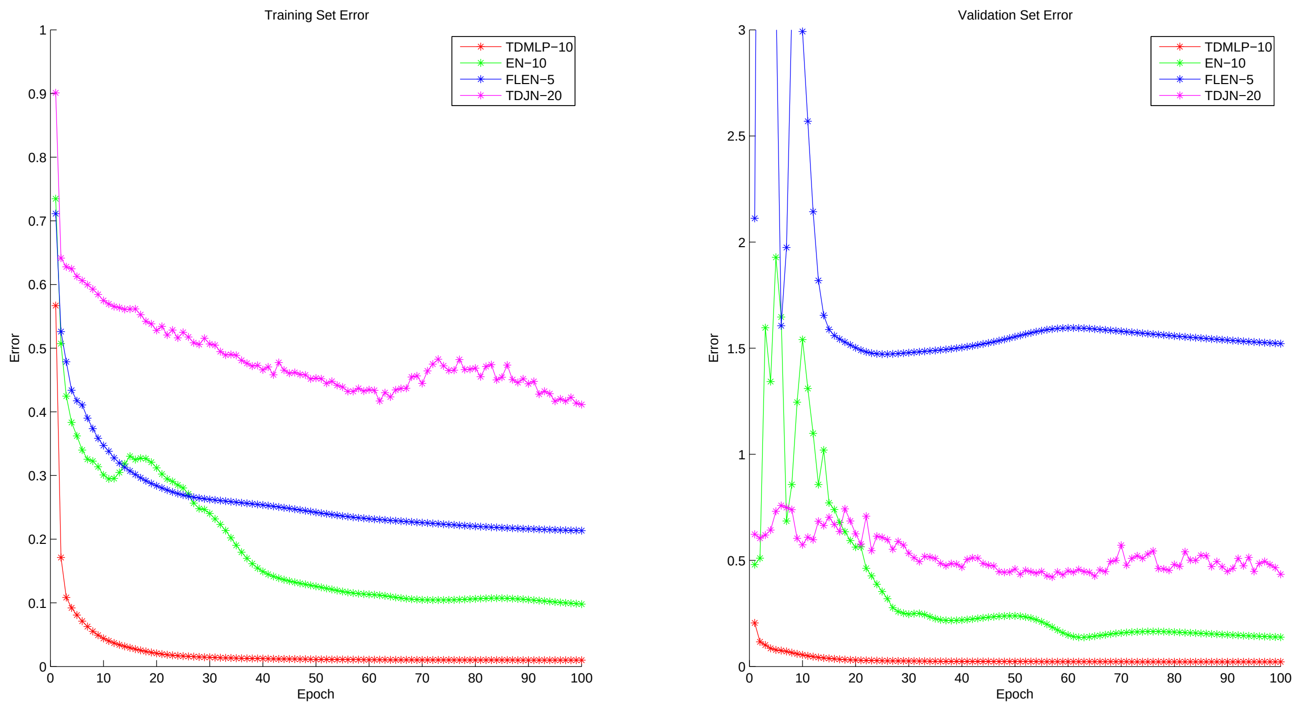

Figure 7: Error plots with function f = @(x)(x(1)*x(2)*0.3 - abs(x(4)) - sin(pi*x(3)) + 1). Number of hidden units are given in the legend.

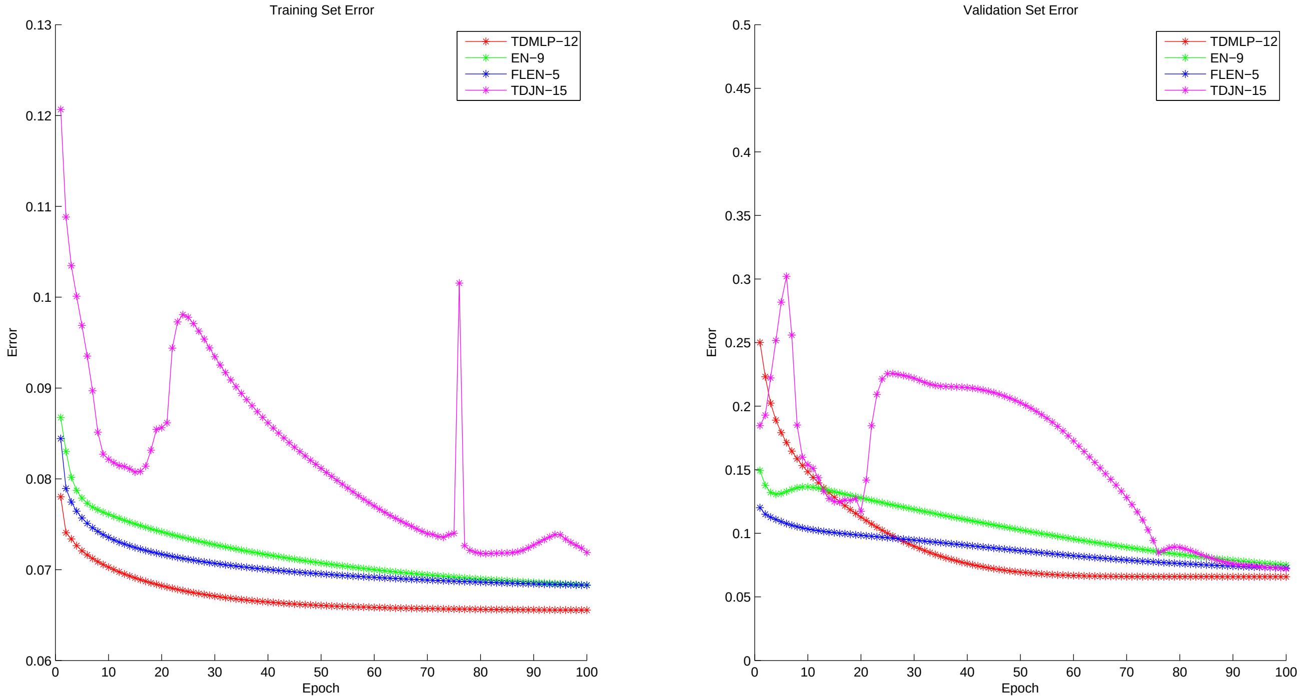

Figure 8: Error plots with function f = @(x) (mod(x(1)*x(2) + 358,547)/211 - sigmoid(x(3)-x(4))*(rand $\geq$ 0.5)). Number of hidden units are given in the legend.

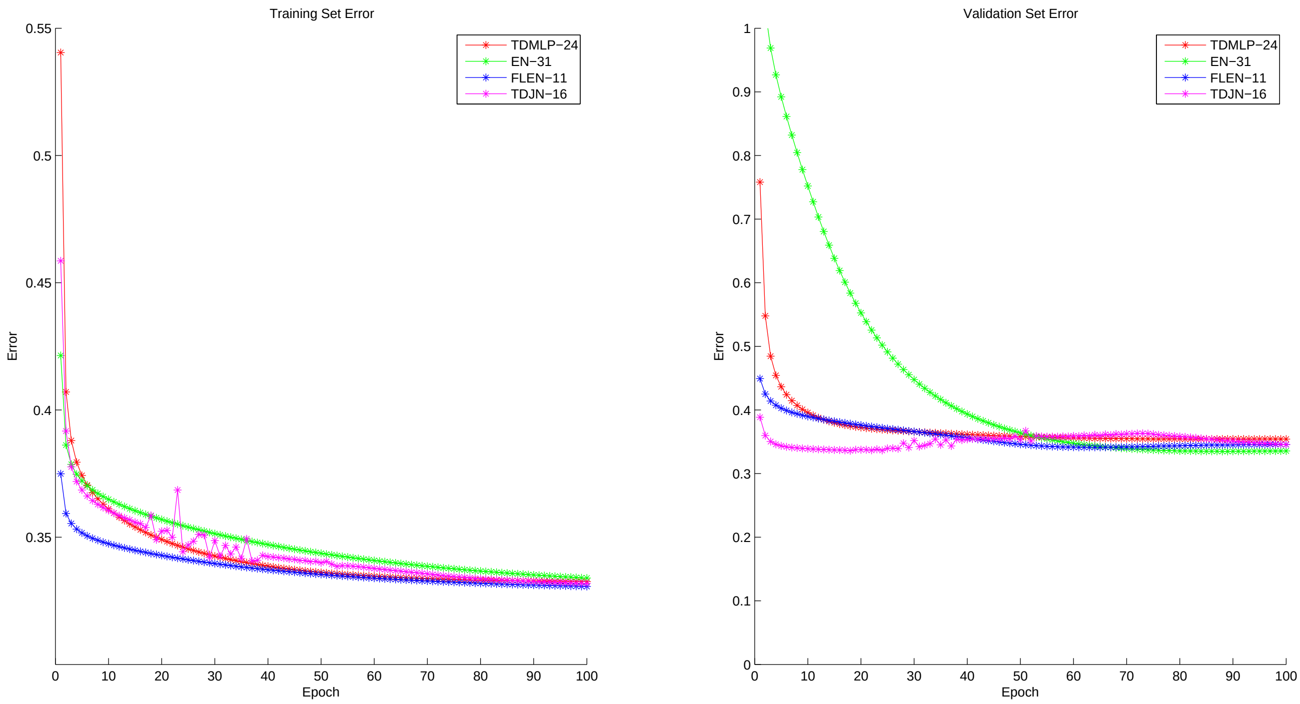

Figure 9: Error plots with function f = @(x)(sigmoid(x(1)*2-x(2)*5) + sin(pi*x(3)) - abs((x(1))/(x(1)+x(4))) - sin(pi*x(2)*4)). Number of hidden units are given in the legend.

3.2 Original Data Sets

- Here, I have incorporated three pairs of plots, each for one outlet. You can see those in Figures 10, 11 and 12.

- Since the raw data are made of integers that may be as big as 500, I needed to somehow scale down the data. Two approaches that I used were to divide each time series to the greatest number in the series (to scale down to 0-1 interval) and taking the derivative. These plots are generated from the data that are preprocessed with the first method described.
- As you see in all figures, FLEN performs really poor. When I first see the plots, I was suspicious of overfitting but decreasing number of hidden units did not change plots much.
- The main goal of a newspaper distributor is to predict demand and deliver newspapers accordingly. But none of these methods was able to do so. In fact, the returns increase substantially in many of the cases that I tested.
- In some of cases, the error rate in validation set does not decrease. So basically learner learns nothing.
- In addition to those, for each outlet I consider sales in a day-of-week as one time series, which corresponds to 7 different time series for each outlet. The second way of processing data usually results in better predictions because standard deviation is smaller in the latter case, compared to having one long series for one outlet. Standard deviation is higher in the first case because newspaper sales tend to make jumps in weekends whereas numbers are relatively more stable during weekdays. Results are not included in this work because I ran out of time while experimenting.

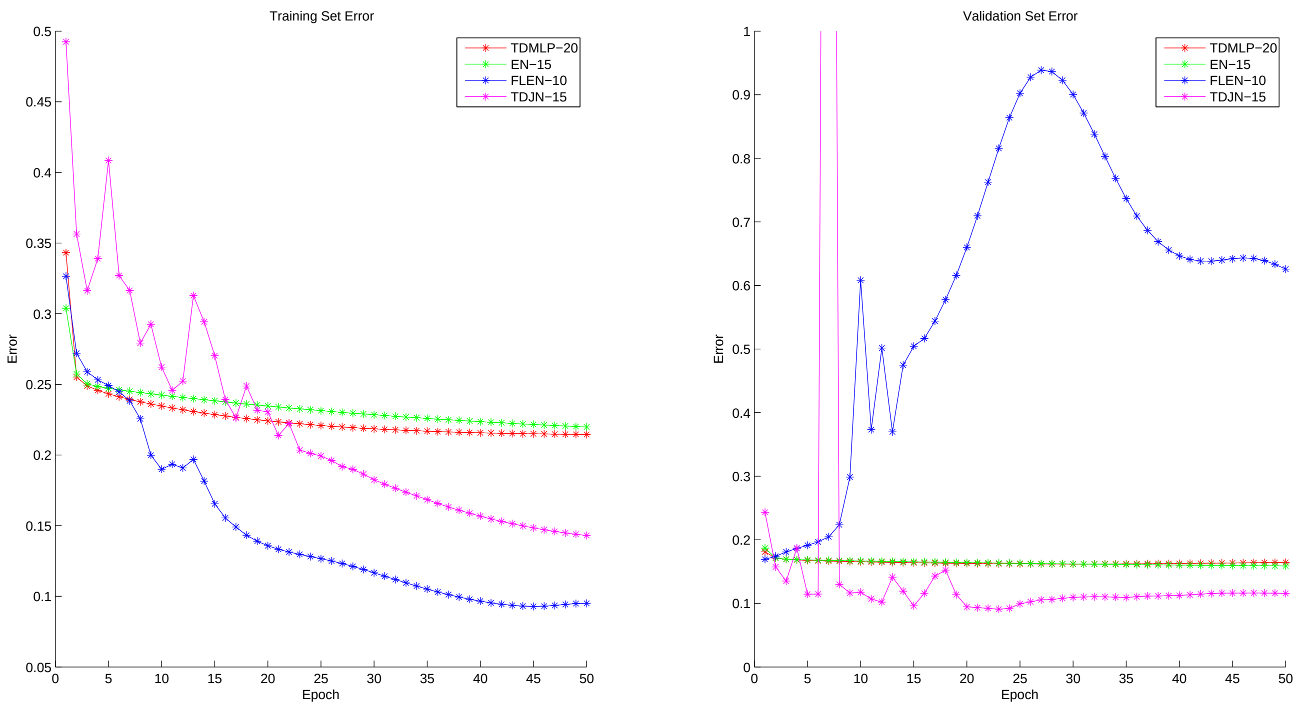

Figure 10: Result for one of the outlets

4 Further Work

- More advanced ways of generating synthetic time series could be discovered.
- Real data can be preprocessed much better.
- It would be nice to compare performances of each method in each (time series) cluster.

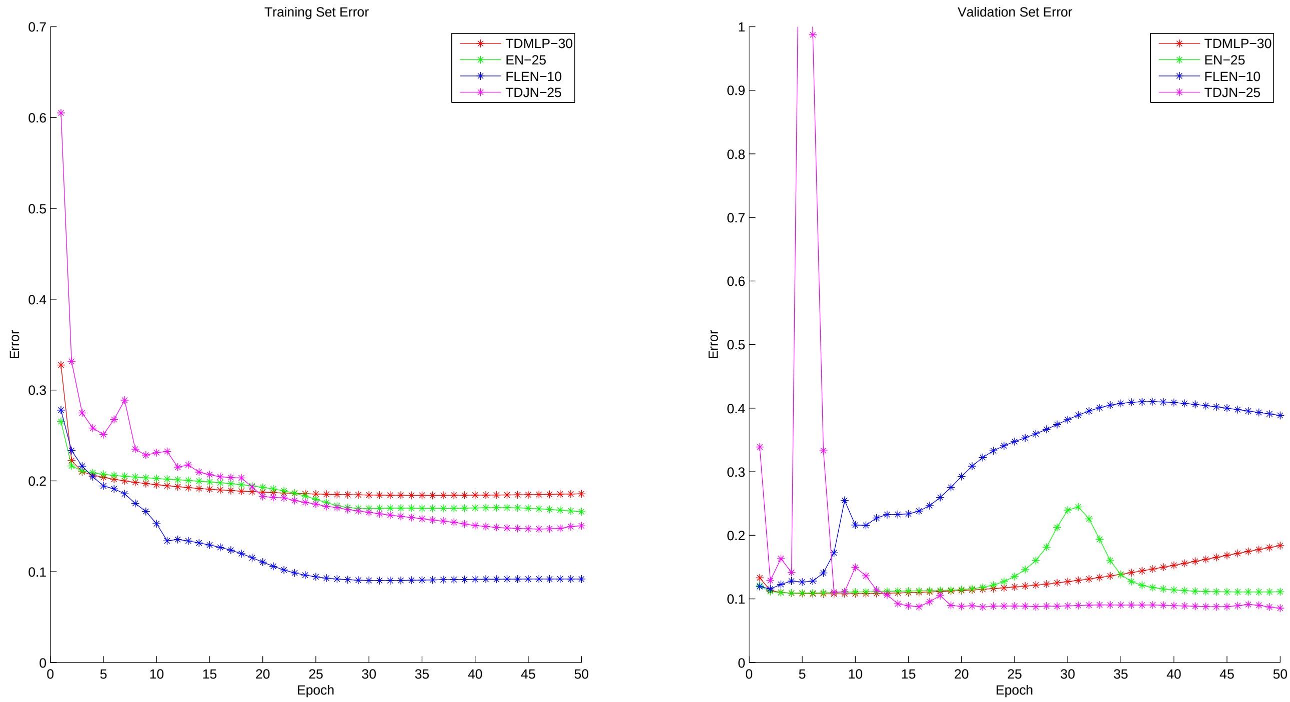

Figure 11: Result for one of the outlets

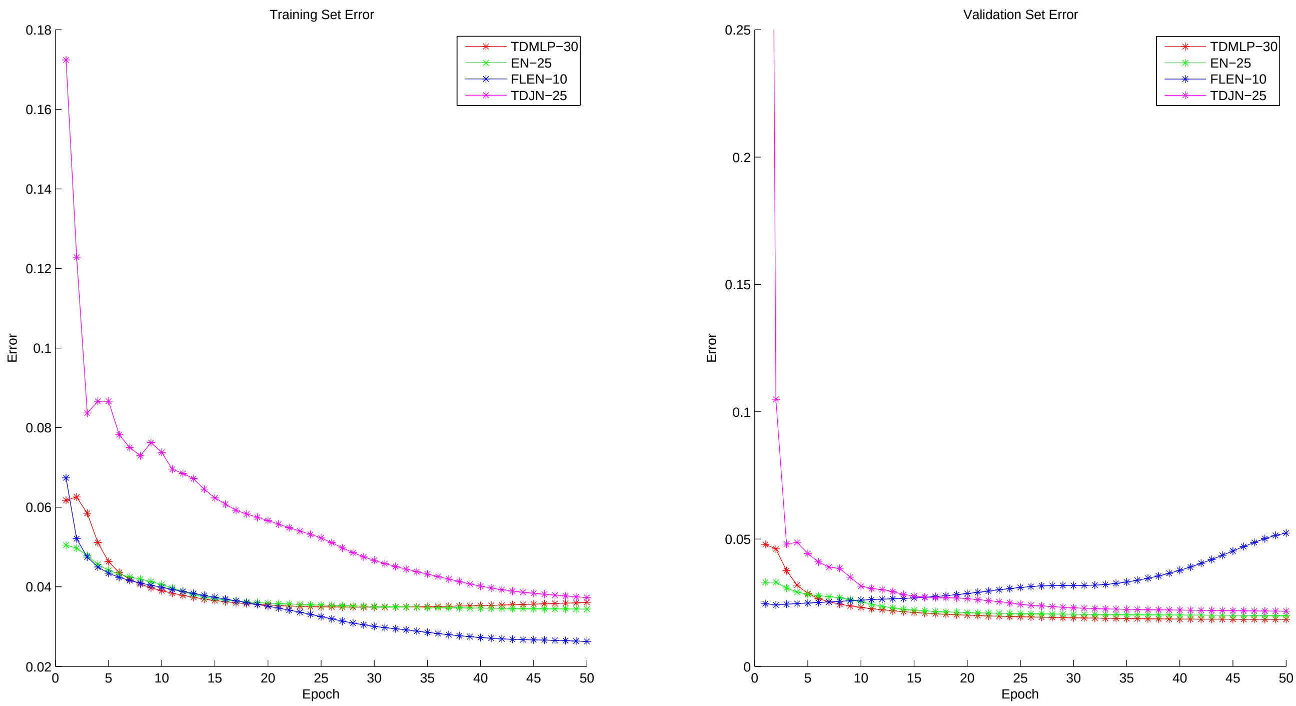

Figure 12: Result for one of the outlets

5 Source Code

5.1 Manual for Source Code

Below is a brief explanation of what each script does. Please note that there are a lot of repetition in the source code this is because I preferred to keep things easy to see and debug.

5.1.1 SOM-related Code

- *som_main_script.m*: Just a wrapper script. Prepares data and model parameters, makes calls to other scripts and visualizes the result.
- *feature_clustering.m*: Implements SOM.
- *hierarchical_clustering.m*: Generates synthetic data and make visualization to test implementation of average and single linkage clustering.
- *average_linkage_clustering.m*: Implements average linkage clustering.
- *single_linkage_clustering.m*: Implements single linkage clustering.

5.1.2 RNN-related Code

- *generative_model.m*: Creates linear/nonlinear sequences of numbers with some Gaussian noise
- *script_synthetic_data.m*: Runs networks with synthetic data and plots figures
- *script_real_data.m*: Runs networks with one of the outlets and plots figures
- *mlp_updates.m*: Implements tapped-delay multilayer perceptron updates.
- *mlp_real_data_day_by_day.m*: Runs the TDMLP on real data. Here, sales of a particular day for an outlet is assumed to form one time series.
- *elman_updates.m*: Implements updates for Elman network
- *elman_real_data_day_by_day.m*: Just like in *mlp_real_data_day_by_day.m*, assumes sales of a particular day for an outlet as one time series and runs EN updates.
- *full_lag_elman_updates.m*: Runs full lag Elman network updates on synthetic data
- *tapped_jordan_updates.m*: Runs tapped-delay Jordan network updates on synthetic data

5.2 SOM-related Code

5.2.1 som_main_script.m

```
clc;
clear all;
close all;

%% initialization
T = 261*7; % length of time series = number of objects
N = 340; % number of time series = number of features

sRow = 30; % number of rows in SOM
sCol = 20; % number of columns in SOM

eta = 0.1; % learning rate
eta_update = 1 - 1e-3; % update parameter of eta
epC = 10; % number of epochs

```

```
sigmaI = 5; % initial width
t1 = epC / log(sigmaI); % time constant

%% data preprocessing
load full_outlets_data; % only outlets with no missing info
Sat = reshape(Sat,T,N); % each column contains 261 week info of one outlet
zSat = bsxfun(@rdivide, Sat, max(Sat))'; % z-normalization

%% running feature clustering algorithm
feature_clustering

%% visualization of 40 random outlets info
nums = randi(N,1,40);
figure(1);
for i=1:40
    subplot(5,8,i);
    plot(Sat(:,nums(i)));
    %ylim([-5,5]);
end
figure(2);
%som = permute(som,[2 3 1]);
for i=1:40
    subplot(5,8,i);
    imagesc(som(:,:,nums(i)));
end

%% extracting similarity matrix
similarity_matrix

%% hierarchical clustering
average_linkage_clustering
% single_linkage_clustering

%% end
beep;
pause(1);
beep;
```

5.2.2 feature_clustering.m

```
% each day is another sample
% each sample in #_of_outlets=N dimension
som = rand(N, sRow, sCol); % som object
dis = zeros(sRow, sCol); % euclidian distance of current time series to all
    maps

for ep=1:epC
     ep
     %% updates
     % sigma update
     %{
```

```
    sigma = sigmaI * exp(-ep/t1);
    variance = sigma^2;
    %}

    %another sigma update
    sigma = sigmaI / (1 + 2*ep/epC);
    var = sigma^2;

    % learning rate update
    eta = eta*eta_update;

    % stores neighbourhood functions of neurons in the map
    nhood = zeros(sRow, sCol, T);

    %%
    for t=randperm(T)

        %% distance calculation bw current data point and all som
        % objects
        for r=1:sRow
            for c=1:sCol
                tmp = som(:,r,c) - zSat(:,t);
                %dis(r,c) = tmp' * tmp;
                dis(r,c) = sqrt(tmp' * tmp);
            end
        end

        %% winner som object
        [minRow, minCol] = find(dis==min(min(dis)));

        %% calculating neighborhood
        for r=1:sRow
            for c=1:sCol
                tmp = (minRow - r)^2 + (minCol - c)^2;
                nhood(r, c, t) = exp(-tmp/(2*var));
            end
        end

        %% som update
        for r=1:sRow
            for c=1:sCol
                delta =  eta*nhood(r, c, t)*(zSat(:,t)-som(:,r,c));
                som(:,r,c) = delta + som(:,r,c);
            end
        end
    end
end
```

5.2.3 hierarchical_clustering.m

```
%% synthetic data generation
```

```
clear all;
clc;
close all;
N = 45;
cc = [0,0; 1,1; 2,2]';
sigma = [0.4, 0.4, 0.4]';
nums = zeros(2,N);

for c=1:3
    for i=1:N/3
        nums(:,(c-1)*N/3+i) = cc(:,c) + sigma(c)*randn(2,1);
    end
end

figure;
colors = {'ro','go','bo'};
hold on;
for c=1:3
    for i=1:N/3
        plot(nums(1,(c-1)*N/3+i),nums(2,(c-1)*N/3+i), colors{c});
    end
end
title('Real Case');
hold off;

sigma = zeros(N); % distance matrix
for i=1:N
    tmp = repmat(nums(:,i),1,N)-nums;
    sigma(i,:) = -1*sum(tmp.^2); % inverting distances
    sigma(i,1:i) = -Inf;
end

average_linkage_clustering;
single_linkage_clustering
```

5.2.4 average_linkage_clustering.m

```
%% hierarchical clustering (average linkage clustering)

% load sigma.mat;
c = num2cell(1:N); % cell i stores ids of outlets in cluster i
for i=1:N-1
    c_count = length(c); % number of clusters
    dis = ones(c_count)*-Inf; % distances between clusters

    % calculation of inter-cluster distances
    for c1 = 1:c_count-1
        for c2 = c1+1:c_count
            cluster1 = c{c1};
            cluster2 = c{c2};
            tmp = combvec(cluster1, cluster2);
```

```
            d = 0;
            for t=1:size(tmp,2)
                %lower triangle of sigma (similarity) matrix is set to -Inf
                %so that max fnc. can be used with one line.
                d = d + max(sigma(tmp(1,t),tmp(2,t)),sigma(tmp(2,t),tmp(1,t)));
            end
            d = d/size(tmp,2); % average similarity bw two clusters
            dis(c1,c2) = d;
        end
    end
    [maxRow, maxCol] = find(dis==max(max(dis))); % most similar clusters

    c{maxCol} = [c{maxCol}, c{maxRow}]; % merge clusters
    c(maxRow) = []; % remove cluster
    snapshots{i} = c; % save current clusters
end

if 0
    %% visualization of clustering with real data
    k = 20; % number of clusters at the time we stopped
    m = 4; % only clusters with m+ members are plotted
    for i=1:length(clusters)
        if length(clusters{i})>=m
            figure;
            for j=1:m
                subplot(m,1,j);
                plot(zSat(clusters{i}(j),:));
            end
        end
    end
else
    %% visualization of synthetic data
    figure;
    colors = {'ro','go','bo'};
    hold on;
    for c=1:3
        for i=1:length(snapshots{N-3}{c})
            plot(nums(1,snapshots{N-3}{c}(i)),nums(2,snapshots{N-3}{c}(i)),colors{c});
        end
    end
    title('Average Linkage Clustering Output');
    hold off
end
```

5.2.5 single_linkage_clustering.m

```
%% hierarchical clustering (single linkage clustering)
% load sigma.mat;
snapshots = zeros(N,N);
```

```
c_id = (1:N)';
for c=1:N-1
    snapshots(:,c) = c_id;
    while 1
        [maxRow, maxCol] = find(sigma==max(max(sigma)));
        sigma(maxRow, maxCol) = -Inf;
        sigma(maxCol, maxRow) = -Inf;
        if c_id(maxCol) ~= c_id(maxRow)
            id1 = c_id(maxCol);
            id2 = c_id(maxRow);
            c_id(find(c_id==id2)) = id1;
            break;
        end
    end
end

if 0
    %% visualization of real data
    k = 5; % number of clusters
    m = 4; % only clusters with m+ members are plotted
    ss = snapshots(:,N-k+1); % current snapshot = cluster id's
    unqs = unique(ss); % number of clusters
    counts = zeros(1,k);
    for i=1:k
        counts(i) = length(find(ss==unqs(i)));
    end
    clusters = unqs(counts>=m); % clusters with m+ members
    for i=1:length(clusters)
        figure;
        outlet_ids = find(ss==clusters(i));
        for j=1:4
            subplot(4,1,j);
            plot(zSat(outlet_ids(j),:));
        end
    end
else
    %% visualization of synthetic data
    k = 3; % number of clusters
    ss = snapshots(:,N-k+1); % current snapshot = cluster id's
    unqs = unique(ss); % number of clusters
    figure;
    colors = {'ro','go','bo'};
    hold on;
    for c=1:3
        ids = find(ss==unqs(c));
        for j=1:length(ids)
            plot(nums(1,ids(j)),nums(2,ids(j)), colors{c});
        end
    end
    title('Single Linkage Clustering Output');
```

```
    hold off;

end
```

5.2.6 RNN-related Code

5.2.7 generative_model.m

```
D = 4; % time lag
N = 3000; % length of the data
mn = 0; % mean of the noise
var = 0.05; % variance of the noise
dataOriginal = zeros(N,1); % original data
dataOriginal(1:D) = rand(D,1);
data = dataOriginal; % data with gaussian noise
sigmoid = @(x)(1./(1+exp(-x))); % sigmoid function

%f = @(x)(x(1)*x(2)*0.3 - abs(x(4)) - sin(pi*x(3)) + 1);
% f = @(x)(min(sqrt(abs(x(1)*x(2))),0.5) - 0.25  - sin(pi*x(4)));
%f = @(x) (mod(x(1)*x(2) + 358,547)/211 - sigmoid(x(3)-x(4))*(rand>0.5) );
% f = @(x)( sigmoid(x(1)^2-x(2)) + sigmoid(-x(3)+x(4)/3) - (x(2)<(x(1)+x(3)+
    x(4))/3)*3 - (x(1)<(x(1)+x(3)+x(4))/3)*2  );
%f = @(x)( sigmoid(x(1)*2-x(2)*5) + sin(pi*x(3)) - abs((x(1))/(x(1)+x(4))) -
     sin(pi*x(2)*4) );

for i = D+1:N
    dataOriginal(i) = f(dataOriginal(i-D:i-1));
end
data(D+1:end) = dataOriginal(D+1:end) + mn*ones(N-D,1) + var*randn(N-D,1);

fprintf('max in data %d\n',max(data))
fprintf('min in data %d\n',min(data))

figure;
subplot(211);
plot(dataOriginal);
title('Original Data');
subplot(212);
plot(data);
title('Perturbated Data');
```

5.2.8 script_synthetic_data.m

```
addpath MLP; % 1
addpath ELMAN; % 2
addpath FULL_LAG_ELMAN; % 3
addpath TAPPED_JORDAN; % 4

methods{1}='TDMLP';
methods{2}='EN';
methods{3}='FLEN';
methods{4}='TDJN';

%% data generation
generative_model;

```

```
%% preparing  data
N = 2000; % number of data points in the training set
Nval = 999; % number of data points in the validation set
% dimensionality of the input
if config==1, D=4; else D=1; end

x = data(1:N);
r = data(D+1:N+1);
xval = data(N-D+2:end-1);
rval = data(N+2:end);

%% learning
Hs = [24, 31, 11, 16];
H = Hs(1);
mlp_updates
c{1} = error;
cv{1} = errorval;
H = Hs(2);
elman_updates
c{2} = error;
cv{2} = errorval;
H = Hs(3);
full_lag_elman_updates
c{3} = error;
cv{3} = errorval;
H = Hs(4);
tapped_jordan_updates
c{4} = error;
cv{4} = errorval;

%%
figure;
subplot(121);
hold on;
plot(1:length(c{1}), c{1},'-r*');
plot(1:length(c{2}), c{2},'-g*');
plot(1:length(c{3}), c{3},'-b*');
plot(1:length(c{4}), c{4},'-m*');
legend(sprintf('TDMLP-%d',Hs(1)),...
sprintf('EN-%d',Hs(2)),...
sprintf('FLEN-%d',Hs(3)),...
sprintf('TDJN-%d',Hs(4)));
hold off;
title('Training Set Error');
xlabel('Epoch');
ylabel('Error');
subplot(122);
hold on;
plot(1:length(cv{1}), cv{1},'-r*');
plot(1:length(cv{2}), cv{2},'-g*');
```

```
plot(1:length(cv{3}), cv{3},'-b*');
plot(1:length(cv{4}), cv{4},'-m*');
legend(sprintf('TDMLP-%d',Hs(1)),...
sprintf('EN-%d',Hs(2)),...
sprintf('FLEN-%d',Hs(3)),...
sprintf('TDJN-%d',Hs(4)));
hold off;
title('Validation Set Error');
xlabel('Epoch');
ylabel('Error');
ylim([0,1000]);
```

5.2.9 script_real_data.m

```
close all;
load full_outlets_data.mat; % only outlets with no missing info

methods{1}='TDMLP';
methods{2}='EN';
methods{3}='FLEN';
methods{4}='TDJN';

T = 261*7;
N = 340;
Sat = reshape(Sat,T,N); % each column contains 261 week info of one outlet
Iade = reshape(Iade,T,N); % each column contains 261 week info of one outlet
Sat = log(Sat);
%Ncons = max(Sat);
%Sat = bsxfun(@rdivide, Sat/5, Ncons); % division by max/5

%% reading data
N = 1200; % number of data points in the training set
Nval = 626; % number of data points in the validation set
index = randi(340);
x = Sat(1:N,index);
r = Sat(D+1:N+1,index);
xval = Sat(N-D+2:end-1,index);
rval = Sat(N+2:end,index);

Hs = [30, 25, 10, 25];

%% learning
H = Hs(1);
mlp_updates
c{1} = error;
cv{1} = errorval;
H = Hs(2);
elman_updates
c{2} = error;
cv{2} = errorval;
H = Hs(3);
```

```
Lag = 7;
full_lag_elman_updates
c{3} = error;
cv{3} = errorval;
H = Hs(4);
Lag = 7;
tapped_jordan_updates
c{4} = error;
cv{4} = errorval;

%% plotting error figures
if 1
    figure;
    subplot(121);
    hold on;
    plot(1:length(c{1}), c{1},'-r*');
    plot(1:length(c{2}), c{2},'-g*');
    plot(1:length(c{3}), c{3},'-b*');
    plot(1:length(c{4}), c{4},'-m*');
    legend(sprintf('TDMLP-%d',Hs(1)),...
        sprintf('EN-%d',Hs(2)),...
        sprintf('FLEN-%d',Hs(3)),...
        sprintf('TDJN-%d',Hs(4)));
    hold off;
    title('Training Set Error');
    xlabel('Epoch');
    ylabel('Error');
    subplot(122);
    hold on;
    plot(1:length(cv{1}), cv{1},'-r*');
    plot(1:length(cv{2}), cv{2},'-g*');
    plot(1:length(cv{3}), cv{3},'-b*');
    plot(1:length(cv{4}), cv{4},'-m*');
    legend(sprintf('TDMLP-%d',Hs(1)),...
        sprintf('EN-%d',Hs(2)),...
        sprintf('FLEN-%d',Hs(3)),...
        sprintf('TDJN-%d',Hs(4)));
    hold off;
    title('Validation Set Error');
    xlabel('Epoch');
    ylabel('Error');
    ylim([0,1]);
end

%% plotting sale figures
if 1
    figure;
    plot(1:N,y*Ncons(index),'-r*',1:N,r*Ncons(index),'-b*')
    title('Training plots')
    legend('prediction','real sale')
```

```
    figure;
    plot(1:Nval,yval*Ncons(index),'-r*',1:Nval,rval*Ncons(index),'-b*')
    title('Test plots')
    legend('prediction','real sale')
end

%% plotting error figures
if 0
    figure;
    subplot(1,3,1);
    plot(1:N,r*Ncons(indexes),'-b')
    title('Training sale plots')
    subplot(1,3,2);
    plot((y-r)*Ncons(indexes))
    title('Training diff plots')
    subplot(1,3,3);
    plot((yval-rval)*Ncons(indexes))
    sum(abs((yval-rval)*Ncons(indexes)))
    sum(rval*Ncons(indexes))
    title('Test diff plots')
end

%% errors
diff = yval - rval ;
returns = sum(ceil(diff(diff>0)*Ncons(index)));
misses = sum(ceil(diff(diff<0)*Ncons(index)));
[sum(Sat(N+2:end,index)*Ncons(index)),sum(Iade(N+2:end,index)),sum(returns),
    sum(misses)]
```

5.2.10 mlp_updates.m

```
%% constants
%H = 10; % number of hidden units
K = 1; % number of outputs
Whi = rand(H,D+1)-0.5; % weights bw real input and hidden unit, weights in
    rows
Woh = rand(K,H+1)-0.5; % weights bw hidden unit and output unit
epoch_c = 50; % number of epochs
sigmoid = @(x)(1./(1+exp(-x))); % sigmoid function
eta = 0.2; % eta
eta_update = 0.95; % in each epoch, eta is multiplied with this number
error = zeros(epoch_c,1); % training error
errorval = zeros(epoch_c,1); % validation error

%% perceptron updates
for ep = 1:epoch_c  % for each epoch
    ep
    dwoh = zeros(K,H+1);
    dwhi = zeros(H,D+1);
    eta = eta*eta_update;
    y = zeros(N-D+1,1);
```

```
    for t = 1:N-D+1
        tmp = sigmoid ( Whi*[1; x(t:t+D-1)] );
        Z = [1; tmp];
        y(t) = Woh * Z;
        dwoh = eta*(r(t)-y(t))*Z';
        dwhi = zeros(H,D+1);
        for h=1:H
            dwhi(h,:) = eta*(r(t)-y(t))*Woh(h+1)*Z(h+1)*(1-Z(h+1))*[1;x(t:t+
               D-1)];
        end
        Woh = Woh + dwoh;
        Whi = Whi + dwhi;
    end

    %error calculation
    yval = zeros(Nval,1);
    for t=1:Nval
        tmp = sigmoid ( Whi*[1; xval(t:t+D-1)] );
        Z = [1; tmp];
        yval(t) = Woh * Z;
    end

    error(ep) = ((y-r)'*(y-r))/N;
    errorval(ep) = ((yval-rval)'*(yval-rval))/Nval;
end
```

5.2.11 mlp_real_data.m

```
load 'full_outlets_data.mat'; % only outlets with no missing info
T = 261*7;
N = 340;

X = 1; % number of outlets to process
indexes = randi(340,1,X);

for i=1:X

    index = indexes(i);
    index = 10;
    Ncons = zeros(1,7);
    returns = zeros(1,7);
    misses = zeros(1,7);

    for d=1:7
        Satd = Sat(d,:,index); % each column contains 261 week info of one
           outlet
        Ncons(d) = max(Satd);
        Satd = Satd'/Ncons(d);

        D = 7; % dimensionality of the input
        N = 200; % number of data points in the training set
```

```
        Nval = 60; % number of data points in the validation set

        %% reading data
        x = Satd(1:N);
        r = Satd(D+1:N+1);
        xval = Satd(N-D+2:end-1);
        rval = Satd(N+2:end);

        %mlp_day_by_day_updates;
        mlp_updates;

        diff = yval - rval ;
        returns(d) = sum(ceil(diff(diff>0)*Ncons(d)));
        misses(d) = sum(ceil(diff(diff<0)*Ncons(d)));


        %% plotting error figures
        if 0
            figure;
            subplot(211);
            plot(1:length(error), error,'-r*');
            title('Training Set Error');
            subplot(212);
            plot(1:length(errorval), errorval,'-r*');
            title('Validation Set Error');
        end
    end
    [sum(sum(Sat(:,end-Nval:end,index))),sum(sum(Iade(:,end-Nval:end,index))
       ),sum(returns),sum(misses)]


    %% plotting error figures
    if 0
        figure;
        subplot(211);
        plot(1:length(error), error,'-r*');
        title('Training Set Error');
        subplot(212);
        plot(1:length(errorval), errorval,'-r*');
        title('Validation Set Error');
    end

    %% plotting sale figures
    if 0
        figure;
        plot(1:N,y*Ncons(indexes),'-r',1:N,r*Ncons(indexes),'-b')
        title('Training plots')
        legend('prediction','real sale')
        figure;
```

```
        plot(1:Nval,yval*Ncons(indexes),'-r',1:Nval,rval*Ncons(indexes),'-b')
        title('Test plots')
        legend('prediction','real sale')
    end

    %% plotting error figures
    if 0
        figure;
        subplot(3,1,1);
        plot(r*Ncons(index))
        title('Training sale plots')
        subplot(3,1,2);
        plot((y-r)*Ncons(index))
        title('Training diff plots')
        subplot(3,1,3);
        plot((yval-rval)*Ncons(index))
        sum(abs((yval-rval)*Ncons(index)))
        sum(rval*Ncons(index))
        title('Test diff plots')
    end
end
```

5.2.12 elman_updates.m

```
%% constants
D = 1; % dimensionality of the input
H = 15; % number of hidden units
K = 1; % number of outputs
C = ones(H,1)/2; % context unit
Whc = rand(H,H)-0.5; % weights bw hidden unit and context unit. first row = weights from context units to the first hidden unit
Whi = rand(H,D+1)-0.5; % weights bw real input and hidden unit, weights in rows
Woh = rand(K,H+1)-0.5; % weights bw hidden unit and output unit
epoch_c = 50; % number of epochs
sigmoid = @(x)(1./(1+exp(-x))); % sigmoid function
eta = 0.2; % eta
mom = 0; % momentum term
eta_update = 0.98; % in each epoch, eta is multiplied with this number
error = zeros(epoch_c,1); % training error
errorval = zeros(epoch_c,1); % validation error

%% perceptron updates
for ep = 1:epoch_c  % for each epoch
    ep
    dwhc = zeros(H,H);
    dwhi = zeros(H,D+1);
    dwoh = zeros(K,H+1);
    eta = eta*eta_update;
    y = zeros(N-D+1,1);
```

```
    yval = zeros(Nval,1);
    for t = 1:N-D+1
        C_old = C;
        C = sigmoid(Whi*[1; x(t:t+D-1)] + Whc*C_old); % stores hidden value
            for this iteration
        Z = [1; C];
        y(t) = Woh * Z;

        dwoh_old = dwoh;
        dwhi_old = dwhi;
        dwhc_old = dwhc;

        dwoh = eta*(r(t)-y(t))*Z';
        dwhi = zeros(H,D+1);
        dwhc = zeros(H,H);
        for h=1:H
            dwhi(h,:) = eta*(r(t)-y(t))*Woh(h+1)*Z(h+1)*(1-Z(h+1))*[1;x(t:t+
                D-1)]';
            dwhc(h,:) = eta*(r(t)-y(t))*Woh(h+1)*Z(h+1)*(1-Z(h+1))*C_old';
        end
        Woh = Woh + (1-mom)*dwoh + mom*dwoh_old;
        Whi = Whi + (1-mom)*dwhi + mom*dwhi_old;
        Whc = Whc + (1-mom)*dwhc + mom*dwhc_old;
    end

    %error calculation
    for t=1:Nval
        C = sigmoid (Whi*[1; xval(t:t+D-1)] + Whc*C);
        Z = [1; C];
        yval(t) = Woh * Z;
    end
    error(ep) = ((y-r)'*(y-r))/N;
    errorval(ep) = ((yval-rval)'*(yval-rval))/Nval;

end
```

5.2.13 elman_real_data_day_by_day.m

```
close all;
%clc;
clear all;

load 'full_outlets_data.mat'; % only outlets with no missing info
T = 261*7;
N = 340;

X = 1; % number of outlets to process
indices = randi(340,1,1);
indices = 20;

results = zeros(4,X);
```

```
for i=1:X

    index = indices(i);
    Ncons = zeros(1,7);
    returns = zeros(1,7);
    misses = zeros(1,7);

    for d=1:7

        %% preparing data
        tmp = Sat(d,:,index); % each column contains 261 week info of one outlet
        Ncons(d) = max(tmp);
        tmp = tmp'/Ncons(d);

        N = 200; % number of data points in the training set
        Nval = 60; % number of data points in the validation set
        x = tmp(1:N);
        r = tmp(2:N+1);
        xval = tmp(N-1:end-1);
        rval = tmp(N+2:end);

        elman_updates;

        diff = yval - rval ;
        returns(d) = sum(ceil(diff(diff>0)*Ncons(d)));
        misses(d) = sum(ceil(diff(diff<0)*Ncons(d)));

        %% plotting error figures
        if 0
            figure;
            subplot(211);
            plot(1:length(error), error,'-r*');
            title('Training Set Error');
            subplot(212);
            plot(1:length(errorval), errorval,'-r*');
            title('Validation Set Error');
        end

        %% plotting sale figures
        if 1
            figure;
            plot(1:length(y),ceil(y*Ncons(d)),'-r*',1:length(r),ceil(r*Ncons(d)),'-b*')
            title('Training plots')
            legend('prediction','real sale')
            figure;
            plot(1:length(yval),ceil(yval*Ncons(d)),'-r*',1:length(rval),ceil(rval*Ncons(d)),'-b*')
```

```
            title('Test plots')
            legend('prediction','real sale')
        end

        %% plotting miss-return figures
        if 0
            figure;
            subplot(3,1,1);
            plot(r*Ncons(d))
            title('Training sale plots')
            subplot(3,1,2);
            plot((y-r)*Ncons(d))
            title('Training diff plots')
            subplot(3,1,3);
            plot((yval-rval)*Ncons(d))
            sum(abs((yval-rval)*Ncons(d)))
            sum(rval*Ncons(d))
            title('Test diff plots')
        end

    end
    results(:,i) = [sum(sum(Sat(:,end-Nval:end,index))),sum(sum(Iade(:,end-
        Nval:end,index))),sum(returns),sum(misses)]';

end

results
```

5.2.14 full_lag_elman_update.m

```
%% constants
D = 1; % dimensionality of the input
%H = 10; % number of hidden units
K = 1; % number of outputs
Lag = 4; % number of hidden unit "groups" stored
C = ones(H,Lag)/2; % context unit
Whc = rand(H,H,Lag)-0.5; % weights bw hidden unit and context unit. first
    row = weights from context units to the first hidden unit
Whi = rand(H,D+1)-0.5; % weights bw real input and hidden unit, weights in
    rows
Woh = rand(K,H+1)-0.5; % weights bw hidden unit and output unit
epoch_c = 50; % number of epochs
sigmoid = @(x)(1./(1+exp(-x))); % sigmoid function
eta = 0.1; % eta
mom = 0.5; % momentum term
eta_update = 0.98; % in each epoch, eta is multiplied with this number
error = zeros(epoch_c,1); % training error
errorval = zeros(epoch_c,1); % validation error

%% perceptron updates
for ep = 1:epoch_c  % for each epoch
```

```
    ep
    dwhc = zeros(H,H,Lag);
    dwhi = zeros(H,D+1);
    dwoh = zeros(K,H+1);

    eta = eta*eta_update;
    y = zeros(N-D+1,1);
    yval = zeros(Nval,1);
    hdns = zeros(H,Lag+1); % stores hidden unit values

    for t = 1:N-D+1
        C_old = C;
        hdns(:,1) = Whi*[1; x(t:t+D-1)]; % input to hidden units
        for lag=1:Lag
            hdns(:,lag+1) = Whc(:,:,lag)*C(:,lag); % contribution of each lag
        end
        hdns = sigmoid(sum(hdns,2)); % stores hidden values for this iteration
        Z = [1; hdns];
        y(t) = Woh * Z;

        dwoh_old = dwoh;
        dwhi_old = dwhi;
        dwhc_old = dwhc;

        dwoh = eta*(r(t)-y(t))*Z';
        dwhi = zeros(H,D+1);
        dwhc = zeros(H,H,Lag);
        for h=1:H
            dwhi(h,:) = eta*(r(t)-y(t))*Woh(h+1)*Z(h+1)*(1-Z(h+1))*[1;x(t:t+D-1)]';
        end
        for lag=1:Lag
            for h=1:H
                dwhc(h,:,lag) = eta*(r(t)-y(t))*Woh(h+1)*Z(h+1)*(1-Z(h+1))*C_old(:,lag)';
            end
        end
        Woh = Woh + (1-mom)*dwoh + mom*dwoh_old;
        Whi = Whi + (1-mom)*dwhi + mom*dwhi_old;
        Whc = Whc + (1-mom)*dwhc + mom*dwhc_old;
        C = [C(:,2:end), hdns];
    end

    hdns = zeros(H,Lag+1); % stores hidden unit values
    %error calculation
    for t=1:Nval
        C_old = C;
        hdns(:,1) = Whi*[1; x(t:t+D-1)]; % input to hidden units
```

```
        for lag=1:Lag
            hdns(:,lag+1) = Whc(:,:,lag)*C(:,lag); % contribution of each lag
        end
        hdns = sigmoid(sum(hdns,2)); % stores hidden values for this iteration

        Z = [1; hdns];
        yval(t) = Woh * Z;
        C = [C(:,2:end), hdns];
    end
    error(ep) = ((y-r)'*(y-r))/N;
    errorval(ep) = ((yval-rval)'*(yval-rval))/Nval;

end
```

5.2.15 tapped_jordan.m

```
%% constants
D = 1; % dimensionality of the input
%H = 20; % number of hidden units
K = 1; % number of outputs
Lag = 4; % number of outputs stored and given as input to hidden layer
S = rand(Lag,1)-0.5; % state layer
Whs = rand(H,Lag)-0.5; % weights bw hidden unit and state unit. first row = weights from context units to the first hidden unit
Whi = rand(H,D+1)-0.5; % weights bw real input and hidden unit, weights in rows
Woh = rand(K,H+1)-0.5; % weights bw hidden unit and output unit
epoch_c = 50; % number of epochs
sigmoid = @(x)(1./(1+exp(-x))); % sigmoid function
eta = 0.2; % eta
mom = 0.4; % momentum term
eta_update = 0.98; % in each epoch, eta is multiplied with this number
error = zeros(epoch_c,1); % training error
errorval = zeros(epoch_c,1); % validation error
% changes in weights
dwhs = zeros(H,Lag);
dwhi = zeros(H,D+1);
dwoh = zeros(K,H+1);

%% perceptron updates
for ep = 1:epoch_c  % for each epoch
    ep
    eta = eta*eta_update;
    y = zeros(N-D+1,1);
    yval = zeros(Nval,1);
    S = ones(Lag,1)/2;

    for t = 1:N-D+1
        Hunits = sigmoid (Whi*[1; x(t:t+D-1)] + Whs*S); % hidden unit values
```

```
        Z = [1; Hunits];
        y(t) = Woh * Z;

        % storing last updates
        dwoh_old = dwoh;
        dwhi_old = dwhi;
        dwhs_old = dwhs;

        % updates
        dwoh = eta*(r(t)-y(t))*Z';
        for h=1:H
            %dwhi(h,:) = dwhi(h,:) +  eta*(r(t)-y(t))*Woh(h+1)*Z(h+1)*(1-Z(h
               +1))*[1;x(t:t+D-1)]';
            %dwhc(h,:) = dwhc(h,:) +  eta*(r(t)-y(t))*Whc(h+1)*Z(h+1)*(1-Z(h
               +1))*C';

            dwhi(h,:) = eta*(r(t)-y(t))*Woh(h+1)*Z(h+1)*(1-Z(h+1))*[1;x(t:t+
               D-1)]';
            dwhs(h,:) = eta*(r(t)-y(t))*Whs(h+1)*Z(h+1)*(1-Z(h+1))*S';
        end

        % update with momentum
        Woh = Woh + (1-mom)*dwoh + mom*dwoh_old;
        Whi = Whi + (1-mom)*dwhi + mom*dwhi_old;
        Whs = Whs + (1-mom)*dwhs + mom*dwhs_old;

        % update state layer
        S = [S(2:end); y(t)];
        %pause;

    end

    %error calculation
    for t=1:Nval
        tmp = sigmoid (Whi*[1; xval(t:t+D-1)] + Whs*S);
        Z = [1; tmp];
        yval(t) = Woh * Z;
        S = [S(2:end); yval(t)];
    end
    error(ep) = ((y-r)'*(y-r))/N;
    errorval(ep) = ((yval-rval)'*(yval-rval))/Nval;

end

%{
%error calculation
C = ones(H,1)/2; % context unit
for t=1:Nval

```

```
    C = sigmoid (Whi*[1; xval(t:t+D-1)] + Whc*C);
    Z = [1; C];
    yval(t) = Woh * Z;

    if 0
        dwoh = eta*(rval(t)-yval(t))*Z';
        dwhi = zeros(H,D+1);
        dwhc = zeros(H,H);
        for h=1:H
            %dwhi(h,:) = dwhi(h,:) + eta*(r(t)-y(t))*Woh(h+1)*Z(h+1)*(1-Z(h+1))*[1;x(t:t+D-1)]';
            %dwhc(h,:) = dwhc(h,:) + eta*(r(t)-y(t))*Whc(h+1)*Z(h+1)*(1-Z(h+1))*C';
            dwhi(h,:) = eta*(rval(t)-yval(t))*Woh(h+1)*Z(h+1)^2*exp(-1*Whc(h,:)*C)*[1;xval(t:t+D-1)]';
            dwhc(h,:) = eta*(rval(t)-yval(t))*Whc(h+1)*Z(h+1)^2*exp(-1*Whi(h,:)*[1; xval(t:t+D-1)])*C';
        end
        Woh = Woh + (1-mom)*dwoh + mom*dwoh_old;
        Whi = Whi + (1-mom)*dwhi + mom*dwhi_old;
        Whc = Whc + (1-mom)*dwhc + mom*dwhc_old;
    end
end
error(ep) = ((y-r)'*(y-r))/(N*std(r));
errorval(ep) = ((yval-rval)'*(yval-rval))/(Nval*std(rval));
%}
```

References

[1] B. Silva and N. C. Marques, “Feature clustering with self-organizing maps and an application to financial time-series for portfolio selection.” in *IJCCI (ICFC-ICNC)*, 2010, pp. 301–309.

[2] A. K. Smilde, H. A. L. Kiers, S. Bijlsma, C. M. Rubingh, and M. J. van Erk, “Matrix correlations for high-dimensional data: the modified rv-coefficient.” *Bioinformatics*, vol. 25, no. 3, pp. 401–405, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/bioinformatics/bioinformatics25.html

[3] M. I. Jordan, “Serial order: A parallel, distributed processing approach,” Institute for Cognitive Science, University of California, San Diego, Tech. Rep. 8604, 1986.

[4] J. L. Elman, “Finding structure in time,” *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.