Dynamic and Recurrent Neural networks

Keyvan Golestan, Yanxin Wang

***Abstract*— In this paper, Dynamic neural networks are introduced. Typically, architectures and training methods of recurrent neural networks (RNN) are illustrated. This paper presents a recurrent neural network integrated with grey models and simulates it in MATLAB.**

I. INTRODUCTION

Artificial Neural Networks have been researched for decades and are proved to be powerful tools in classification, data processing, function approximation and robotics. Most of the neural network structures used for earlier engineering applications are static neural networks. These neural networks respond instantaneously to the inputs. This mapping of static input-output is well suited for classification applications where both the input and output are independent of time. The absence of feedback from output to input in static neural networks ensures that networks are conditionally stable[1]. However, these networks suffer from the following limitations[1]: (i) In feedforward networks, the information flows from input layer to hidden layer, to output layer and never come back to input layer. (ii) The static neural network does not take into account the time delays that affect the reaction of the whole system. The appliance of time delays to neural networks is biologically motivated, since it is well believed that signal delays are omnipresent in the brain and play a critical role in neurobiological information processing. This concept prompted the development of dynamic neural networks.

Unlike a static neural network, a dynamic neural network deploys extra feedback between the neurons of a layer, and/or between the neurons of different layers in the network. The node equations in dynamic networks are then described by differential or difference equations. This kind of feedback implies that the network has its local memory. The response of dynamic neural networks is now recurrent because the feedback transmits their output to the inputs. Recurrent means that, every epoch a network is trained, the weights are adjusted and the output is then recalculated related to the previous output. For a stable network, successive iterations should produce smaller and smaller output changes until eventually the outputs become constant.

In the 1960's, Minsky and Papert pointed out hidden units to be a potential remedy for some of connection problems. Recurrent connections have lately attracted the same kind of interest much like hidden units extend the computational power of feedforward networks[3].

Keyvan Golestan is with the University of Waterloo, Waterloo, ON, Canada.
Yanxin Wang is with the University of Waterloo, Waterloo, ON, Canada

Dynamic networks are particularly appropriate for system modeling, control and prediction applications. These networks are important because many of the systems that we wish to model in the real world are non-linear dynamic systems. It is necessary to model both the forward and backward dynamics of systems such as airplanes, rockets, spacecraft and robots.

This paper introduces the Dynamic Neural Networks and gives an application example on utilizing one of these networks. Section 2 contains a discussion of the structures developed through these years. The concepts underlying the dynamic neural networks are demonstrated for a simple structure. The training algorithms for these feedback networks are described in Section 3. An online training algorithm, namely real-time recurrent learning, is illustrated in this section, followed by applications and an example in the last section.

II. STRUCTURES OF DYNAMIC NEURAL NETWORKS

A. Non-recurrent networks

If there is no closed interconnection path, the network can still perform as a dynamic network. This type of network is called non-recurrent network. Architecture like this is often referred to as a Time-Delay Neural Network (TDNN)[4]. This network is dynamic because they can deal with time delay. They cache input for a short period of time so that not only current input but also previous input is used at the same time. It should be noted, however, that the TDNN is a feedforward network. It attains dynamic behavior by unfolding the input sequence over time. Non-recurrent networks share the same structures as static neural networks except the cache. To illustrate non-recurrent networks, consider Fig.1, which is a simple dynamic network. It appears as a no-hidden layer static neural network with an extra input, which is connected to the original input through a single time delay. In this figure the vector $P(t)$ represents the input of the network at time interval t. Now suppose that $P(t)$ is a pulse input sequence in discrete time space:

$$p = \left\{0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\right\}$$

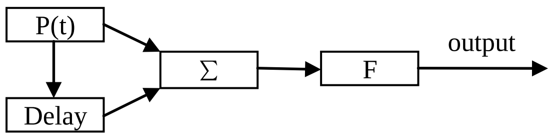

Figure.1 The structure of a simple non-recurrent network

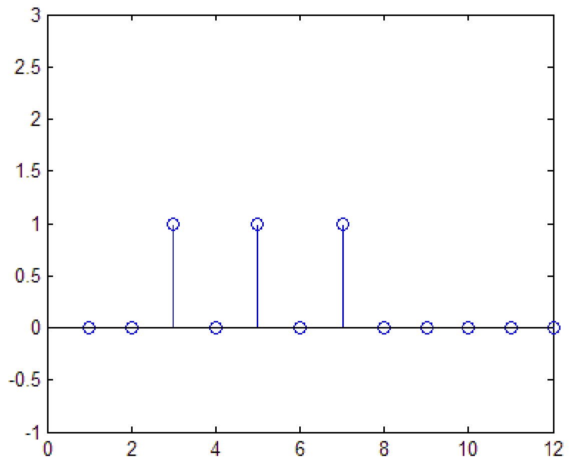

Figure.2 Input time series *p(t)*

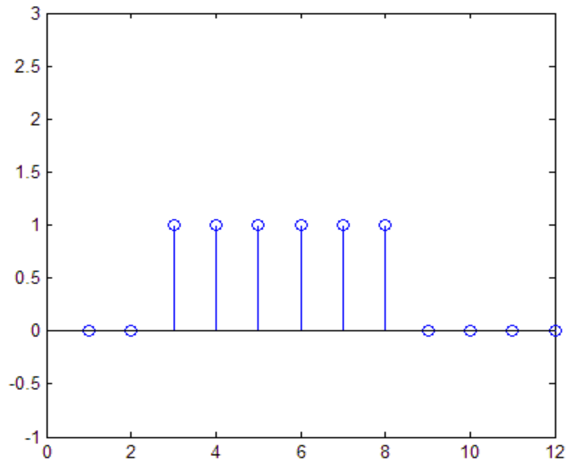

Figure.3 Output time series

p(t) is showed in Fig.2. If all weights are set to 1, the output is showed in Fig.3. The network performs dynamic characteristics because the output is related to previous input data.

The TDNN structure has been used in many applications. For example, Sejnowski and Rosenberg used TDNN for text-to-speech conversion[5], while Waibei et al employed this structure for phoneme recognition[6]. Narendra and Parthasarthy have applied this type of neural structure for system identification and control of nonlinear dynamical systems[7].

B. Recurrent networks

If the feedback from output result in closed circles within network, this kind of network is known as recurrent networks. The neural network model introduced by Hopfield is one of the first dynamic neural network models. This model consists of a single layer network, included in a feedback construction with a time delay as shown in Fig.4. In this figure, *y(k)* and *y(k+1)* represent the output of the neural network at time interval k and k+1, *x* represents the input value, *W(k)* denotes

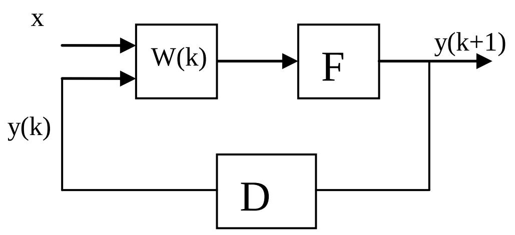

Figure.4 The state-space model of the Hopfield neural structure

the vector of the neural weights, *F* is the nonlinear activation function and , *D* represents the unit delay operator. This feedback network represents a discrete-time dynamical system and can be described by the following equation[7]

$$y(k+1) = F\left[w_{y(k)}\right]$$

Given an initial value *x*, the dynamical system evolves to an equilibrium state if activation function is suitably chosen.

Hopfield neural model has been widely used in many applications such as associative memories, system control, and optimization problems. For mathematical tractability, early model contain only single layer of neurons. Nowadays, however, multi-layered recurrent neural networks have been developed and used for many applications. In spite of the dynamic usages, the basic architecture of the neuron is static. In other words, the neuron simply provides a weighted integration of synaptic inputs over a period of time. That is, there are no dynamical elements within the structure of the neural networks..

On the other side, there are networks having dynamical elements within their structures. In these networks, any neuron may have feedback connections with other neurons. If all neurons have feedback connections with the others, this is called a totally recurrent network. If only part of the neurons have feedback connections with some each other, this is called a partially recurrent network. Elman networks, also known as the Simple Recurrent Networks, shown in Fig.5, are the simplest partially recurrent networks.

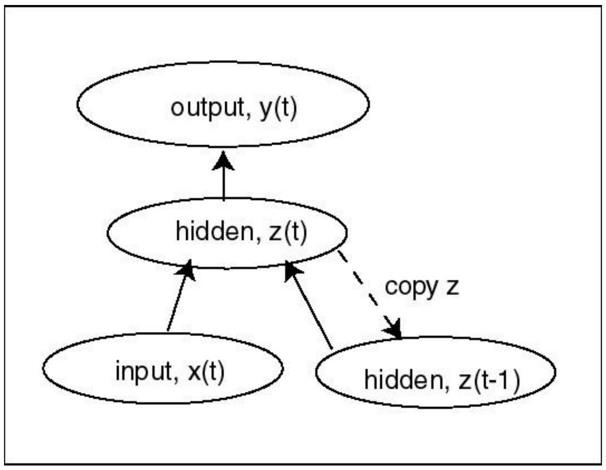

Figure.5 Structure of Elman Network

III. TRAINING ALGORITHMS

How to train dynamic neural networks especially recurrent networks was a problem in 1980s. Since non-recurrent networks have the same feedforward mechanism with static networks, traditional training algorithms could be utilized on them without any trouble. However, recurrent neural networks' unique backward structure caused a lot of problems. Many algorithms were tried and those algorithms evolved as time went by.

The process of training a neural network is, in view of mathematic, to generate the weight matrix from training data. This could be seen as an optimization problem whose target function is to minimize the total error of output. Actually many popular algorithms for static neural networks training take their origins from the gradient descent methods in numeric optimization. Could these methods be applied in dynamic situations was a question bothering researchers when they first came up with idea of recurrent neural networks. Much effort was put on the mathematic proving in 1980s. In 1990, Barak A. Pearlmutter proved some algorithms available in continuous time scenario[8]. He guaranteed that a recurrent network converges to stable fixpoints. One algorithm that is capable of learning fixpoints is back-propagation through time (BPTT). Like back-propagation in static networks, BPTT functions well on recurrent neural networks. Although the main idea of BPTT is still back-propagation learning, BPTT advances in learning through a certain period of time. BPTT has become the most used learning algorithm but could hardly been used as an online training algorithm. Though Elman investigated a version of discrete BPTT in which the temporal history is cut off to make BPTT an online algorithm. However, accuracy is traded off against storage and computational expense. A real online training algorithm namely real-time recurrent learning (RTRL) helps solving this problem.

In this section, BPTT and RTRL are reviewed to show how recurrent networks can be trained.

A. Back-Propagation Through Time (BPTT)

As mentioned above, the BPTT training method is an extension of the standard back-propagation algorithm which is an off-line training method. The basic thought is to transform from a spatial-temporal representation into standard representation in static networks. Extra artificial layers are created to simulate the network structure frozen in time for each sequence of the data obtained. In other words, the recurrent network is transformed into an equivalent feed forward network by unfolding it into artificial layers. The unfolded network can then be trained using the conventional backpropagation algorithm. The process of unfolding a sample network is shown in Fig.6.

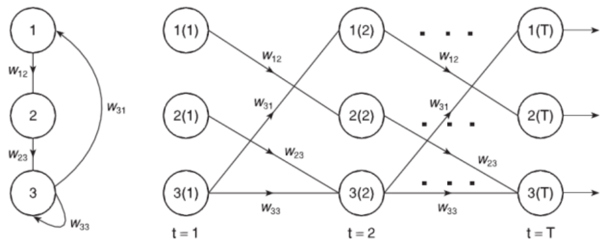

Figure.6 Three-unit recurrent network (left) and its feedforward equivalent network (right)

B. Real-Time Recurrent Learning (RTRL)

In contrast to the off-line weight adaptation of BPTT, the real-time recurrent learning (RTRL) algorithm deals with an online weight adaptation scenario. Assume we are dealing with a fully recurrent neural network where each node of the network has an output signal given by

$$o_i(t) = f(tot_i(t-1)) = f\left(\sum_j w_{ij} o_j(t-1) + x_i(t-1)\right) \quad (1)$$

and where the learning occurs during presentation of sequences of signals. If we denote E_{cum} as the cumulative error over the total time scale "*T*" of all output node errors $E_l(t)$:

$$E_{cum} = \sum_t \sum_l \frac{1}{2} E_l^2(t) \quad (2)$$

where $E_l(t)$ being the error at time *t* between target $d_l(t)$ and output $o_l(t)$ at the output layer *l* with $E_l(t) = d_l(t) - o_l(t)$.

As in the back-propagation learning, we wish to minimize the cumulative error E_{cum} in the weight space:

$$\frac{\partial E_{cum}}{\partial w_{ij}(t)} = \sum_t \frac{\partial E_{cum}}{\partial tot_i(t)} \frac{\partial tot_i(t)}{\partial w_{ij}(t)}, t = 0,1,\cdots,T \quad (3)$$

Using the gradient descent technique, we define the update of the weight as:

$$\begin{aligned}\Delta w_{ij}(t) &= -\eta \frac{\partial E_{cum}}{\partial w_{ij}} \\ &= \eta \sum_l \frac{\partial E_{cum}}{\partial o_l(t)} \frac{\partial o_l(t)}{\partial w_{ij}} \\ &= \eta \sum_l E_l \frac{\partial o_l(t)}{\partial w_{ij}}\end{aligned} \quad (4)$$

where η is the learning rate parameter and

$$\frac{\partial o_l(t)}{\partial w_{ij}} = f'(tot_l(t-1))\left(\delta_{li} o_j(t-1) + \sum_p w_{lp} \frac{\partial o_p(t-1)}{\partial w_{ij}}\right) \quad (5)$$

where $\delta_{ij} = \begin{cases} 1 & if \quad i = j \\ 0 & if \quad i \neq j \end{cases}$

If we denote $\frac{\partial o_p(t)}{\partial w_{ij}} = v_{ij}^{(p)}(t)$, then we have

$$v_{ij}^{(l)}(t) = f'(tot_l(t-1))\left(\delta_{li} o_j(t-1) + \sum_p w_{lp} v_{ij}^{(p)}(t-1)\right) \quad (6)$$

and

$$\Delta w_{ij}(t) = \eta \sum_l E_l v_{ij}^{(l)}(t)$$

The total weight correction over the whole set of sample times $t = 0, 1, \cdots, T$ is given as

$$\Delta w_{ij,total} = \eta \sum_t \sum_l E_l v_{ij}^{(l)}(t), t = 0, 1, \cdots, T \qquad (7)$$

While this method allows for online adaptation, the main drawback is its computational complexity. In asymptotic notation, $O\left(n^4\right)$ computations are performed for each time point, where *n* indicates the number of neurons in the network.

In the consideration of the computational complexity, several modifications could be utilized to RTRL. Teacher forcing technique replaces the actual output $o_l(t)$ with desired output $d_l(t)$ which performs like a teacher. This technique may accelerate the learning process.

The gradient descent approaches, with all their limitations, have been very often used as standard training methods because of the solid mathematical rational for using such approaches. However, the problems of high computational complexity and slow convergence remain the main drawbacks of such approaches. New approaches using genetic algorithms have been proposed and verified. As another important tool in optimization, genetic algorithm can perfectly locate global minimum in relative short time. Research about it is still ongoing, since there are increasing demands for quick online adaptation.

IV. APPLICATIONS AND AN EXAMPLE

The applications using recurrent neural networks in numerous fields have been documented in the literature. The amount of applications using RNNs is large. Various architectures and training methods have been used in different applications ranging from fault monitoring to modeling linguistic behaviors and from control of large chemical plants to the design of micro motors[9]. For instance, Wang et al. used a simple three-layer RNN and the BPTT algorithm to realize speech-to-text conversion. Over 70% of character recognition accuracy was achieved[10]. Kermanshahi applied a combination of an RNN and an FFBP for long-term power system load forecasting[11]. The application of RNNs in finance like stock market prediction has been often considered with skepticism by researchers and experts in the field, but there is ongoing research in this forecasting area as well. In the area of computer science, RNNs have been used to process data in arbitrarily shaped structures to encode complex relationships. Other examples include structural and syntactic pattern recognition, computational biology, and theorem proving. But the major applications of recurrent neural networks still remain in the area of control of nonlinear systems and chaotic system prediction.

In the remaining portion of this section an example by Yuh-horng Wen et al.[12] is introduced to illustrate how a recurrent neural network works in prediction. They developed a travel time estimation process by integrating a missing data treatment and data-fusion-based approaches. They presented a recurrent neural network integrated with grey models for real-time travel time estimation and further compared with a speed-based link travel time extrapolation model for analytical travel time estimation.

A. Problem and Background

Research into travel time estimation keeps to attract the attention of Intelligent Transportation Systems (ITS) academicians and engineers, since accurate travel time estimation is an essential part of an Advanced Traveler Information Systems (ATIS). Many ITS researchers and transportation agencies use the traffic data from dual-loop detectors, which are sensors now available in many locales of highways and urban roads. A dual-loop detector consists of two consecutive single inductance loops spaced a few feet apart. In the dual-loop system, a timer is started when the first single inductance loop detects a vehicle. The timer stops when the same vehicle is detected at the second single inductance loop. The vehicle speed is calculated through dividing distance between the two single-loop detectors by the elapsed time and finally converts to miles per hour. The speed, length, and vehicle data are aggregated into 20-second intervals for outputs. Dual-loop detector systems are capable of gathering traffic count (i.e., the number of vehicles that pass over the detector in that period of time), velocity and occupancy (i.e., the length of time that vehicles are detected).

A number of methods to estimate travel time have previously been proposed and developed, such as, Dailey[13], Cremer and Schutt[14], Nam and Drew[15]. Most of these models used extrapolation to get the link travel time by utilizing the flow and occupancy data measured by the loop detectors to calculate the speed at that point.

Recently, real-time forecasting of travel time with reasonable speed and accuracy has become an emerging issue towards travel time estimation. More and more studies for travel time estimation focused on real-time data fusion approaches. Data fusion is concerned with the problem of combining data sets coming from different sources into a single data set when entries are absent or missing in some data sets. Neural networks easily satisfy the demands of data fusion due to their nature ability to deal with complex information. Advanced Driver and Vehicle Advisory Navigation Concept, one of the ITS projects in Chicago, proposed a series of studies using neural networks and fuzzy neural networks for travel time prediction[16]. Recently, Dharia and Adeli used back propagation neural network for travel time prediction[17]. Lint et al. developed the state-space neural networks for travel time prediction. They also discussed the accuracy, robustness, and uncertainty issues in their networks.

However, raw real-time traffic data from loop detectors are not suitable to be processed by data fusion tools for travel time estimation because they may contain a lot of misleading data items. Raw Real-time traffic data from roadside loop detectors are inevitably corrupted by detector faults or transmission distortion, resulting in unexpected missing values, abnormal data and other errors. Most missing values in traffic data sets occurred at some time-spots of the traffic data time-series due to temporary power or communication failures in the traffic monitor system. Some failed detectors will result in a whole series of detector data missing for some time. Missing data treatment is an important step before data processing in travel time estimation, since raw real-time traffic data may cause large errors and lower the quality of travel time estimation. Therefore, raw real-time traffic data should be pre-processed such that data sets become complete and reasonable in order to allow the whole or partial data set to be processed by a data fusion tool. Specifically, in aspect of accurate travel time estimation, an understanding of the quantitative effects of missing data on travel time estimating performance is needed.

Missing data problems have been studied in many areas, such as image processing, system identification, speed recognition, and to a limited extent, traffic forecasting. Lint et al. followed the procedure of imputing missing data and estimating travel times using data fusion approaches. What was different to previous studies, they attempted to integrate grey theory models in data imputation and data fusion models based on neural network. Grey system theory is an interdisciplinary scientific area that was first introduced in early 1980s by Deng (1982). Since then, the theory has become quite popular with its ability to deal with the systems that have partially unknown parameters. As superiority to conventional statistical models, grey models require only a limited amount of data to estimate the behavior of unknown systems. Fields covered by grey theory include system analysis, data processing, prediction and control. It has been widely and successfully applied to various systems such as social, economic, financial, scientific and technological, agricultural, industrial, transportation, mechanical, meteorological, ecological, hydrological, geological, medical, military, etc., systems. In the missing data treatment section, this paper develops a grey-theory-based approach to impute missing values for spatial and temporal absences. Furthermore, two data fusion models for travel time estimation are discussed, one is off-line model and another is real-time model. Temporal patterns together with spatial patterns of traffic data are considered in both data fusion models. They use the speed-based link travel time extrapolation approach for off-line estimation and use grey-based recurrent neural networks for real-time travel time prediction. The grey-based recurrent neural networks combine grey models with recurrent neural networks such that they are capable of dealing with both randomness and implicitly in spatial-temporal traffic data.

B. *Missing Data Treatment*

Missing values from a single detector may occur at some time spots of the time series and/or occur in the whole series because of detector failure. Their study uses a grey time series model to impute temporal missing values occurring at some time spots. Spatial missing data is recovered by proposing a grey-relational-based nearest neighbor method in addition.

The grey accumulated generating operation (AGO) is an important feature of grey models, which focuses largely on reducing the randomness of data.

Let $x_d^0 = \left[x_d^0(1), x_d^0(2), \ldots, x_d^0(n)\right]$ be an original time-series traffic data record from detector d with n time-spots. The AGO formation of x_d^0 is $x_d^1 = \left[x_d^1(1), x_d^1(2), \ldots, x_d^1(n)\right]$, where $x_d^1(k) = \sum_{i=1}^{k} x_d^0(i), k = 1, 2, \ldots n$. The inverse operation of AGO (IAGO) is to recover x_d^0 from x_d^1 as $x_d^0(k) = x_d^1(k) - x_d^1(k-1)$. The grey time series model can be formulated as $\frac{dx_d^1}{dk} + ax_d^1 = u$. The forecasting function of $x_d^0(k)$ can be obtained as follow:

$$\hat{x}_d^0(k) = \left(x_d^0(1) - \frac{\hat{u}}{\hat{a}}\right)\left(1 - e^{\hat{a}}\right)e^{-\hat{a}(k-1)} \tag{8}$$

Now missing values $NA_d(t+1)$ from detector d at time point $t+1$, can be replaced by a forecast $\hat{x}_d^0(k+1)$.

Let $x = \left[x(1), x(1), \ldots, x(n)\right]$ be a traffic data set. we use $x_k = \left[x(1), x(2), \ldots, x(k), NA(k+1), \ldots, NA(n)\right], k < n$ to represent a data vector with $(n-k)$ missing values. Consider we have x_k and x_l, they can be expressed as

$x_k = \left[x_k(1), x_k(2), \ldots, x_k(d), S(d+1), \ldots S(k), NA(k+1), \ldots, NA(n)\right]$
and

$x_l = \left[x_l(1), x_l(2), \ldots, x_l(d), S(d+1), \ldots S(k), NA(k+1), \ldots, NA(n)\right]$

where $d \le \min(k, l)$, and $S(i)$ represents the value that is missing in one of the vectors x_k and x_l but not in both. The grey relational coefficient, $\xi_{kl}(i)$, between data vectors x_k and x_l at a certain time point $i = t$ can be expressed by the following equation.

$$\xi_{kl}(t) = \frac{\min_l \min_t |x_k(t) - x_l(t)| + \rho \max_l \max_t |x_k(t) - x_l(t)|}{|x_k(t) - x_l(t)| + \rho \max_l \max_t |x_k(t) - x_l(t)|} \quad (9)$$

where $|x_k(t) - x_l(t)|$ represents the absolute difference between the two vectors. $\rho(0 \le \rho \le 1)$ is a distinguishing coefficient used to adjust the range of the comparison environment, and to control level of difference of the relational coefficients. In cases where data variation is large, ρ usually ranges from 0.1 to 0.5 for reducing the influence of extremely values. For all $i \le d$, the grey relational grade, γ_{kl} , can be expressed as

$$\gamma_{kl} = \frac{1}{d} \sum_{i=1}^{d} \xi_{kl}(i) \quad (10)$$

The aim of grey relational grade is to recognize the geometric relationship between two data vectors in relational space. In this case, we use grey relational grade to measure the pseudo-similarity between vectors. The grey-relational-based pseudo-nearest neighbor method is presented as follows:

x_l has the presence of value $x_l(k+1)$; x_l has the maximum grey relational grade; the present value $x_l(k+1)$ of x_l is used to replace the $NA(k+1)$ of x_k .

C. Real-time Travel Time Estimation

Neural network models had been used in many studies to predict travel times, e.g., Lint et al. developed the state-space neural networks for travel time forecasting[18]. However, in travel time data fusion, both spatial and temporal patterns of traffic data should be considered as the dynamic inherence of travel times. The recurrent neural network (RNN) model is a dynamic network with internal feedbacks as mentioned in previous section. The simple recurrent neural network, or Elman neural network (ENN), is used in this paper. Elman emphasizes the capability of the ENN model to learn complex spatial -temporal patterns. ENNs are backpropagation networks of one hidden layer, with the additional feedback connection from the output of the hidden layer to its input layer. This feedback connection allows ENNs to learn to handle both temporal patterns and spatial patterns. The ENN learns to integrate current inputs together with its previous internal states. However, inherent uncertainty of the traffic data distribution complicates real-time travel time estimation. Stochastic fluctuations in traffic data and unexpected abnormal fluctuations (i.e., if congestion and incidence occurring) significantly influence the traffic state in future time. Ma et al. noticed that the uncertainty and randomness in training data degrades the forecasting performance of neural network[19]. If the randomness in the training data can be reduced, better forecasting performance can be achieved. Their study proposes a novel grey-based recurrent neural network, which combine grey models with the recurrent neural network, for dynamic travel time estimation.

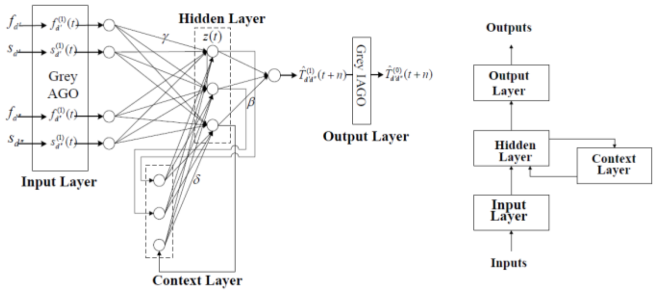

Figure.7 Architecture of Grey-based Recurrent Neural Network

Fig.7 shows the structure of the grey-based ENN model, which consists of two trainable layers of neurons (hidden and output layers), with feedback from the first-layer output to the first layer input. Context layer is simple duplication of hidden layer. In Fig.7, $f_{d'}$ and $s_{d'}$ respectively represent the volume and speed data from the upstream station; while $f_{d''}$ and $s_{d''}$ represent those from downstream station. $f_{d'}^{(1)}(t)$ and $s_{d'}^{(1)}(t)$ are the AGO formations of $f_{d'}$ and $s_{d'}$ at time interval t . $f_{d''}^{(1)}(t)$ and $s_{d''}^{(1)}(t)$ are the AGO formations of $f_{d''}$ and $s_{d''}$. AGO formations is used to reduce the randomness in data. Then the current AGOs for both volume and speed parameters from upstream and downstream stations are used as inputs to the grey-based RNN model. The output of grey-based RNN model represents the link travel time within the same section at some future time interval $(t+n)$. The hidden neuron function for travel time estimation can be described as:

$$Z_j(t+n) = f_H^{'}\left(\begin{array}{l} \gamma_{1j} f_{d'}^{(1)}(t) + \gamma_{2j} s_{d'}^{(1)}(t) + \gamma_{3j} f_{d''}^{(1)}(t) + \gamma_{4j} s_{d''}^{(1)}(t) + \\ \sum_j \delta_j z_j(t+n-1) \end{array} \right) \quad (11)$$

where $f_H^{'}(\bullet)$ is the hidden neuron function for travel time estimation. $\gamma_{1j}, \gamma_{2j}, \gamma_{3j}, \gamma_{4j}$ are weights connecting the input layer units to hidden unit j at time interval t . δ_j is the weight connecting context layer units to hidden unit j . $z_j(t+n)$ is a output of hidden node j at time interval $t+n$, and context layer input at time interval $t+n-1$ then is given as $z_j(t+n-1)$.

The output neuron of grey-based RNN can then be presented as

$$\hat{T}_{d'd''}^{(1)}(t+n) = f_o^{'}\left(\beta_1 + \sum_j \beta_j z_j(t+n) \right) \quad (12)$$

where $f_o^{'}(\bullet)$ is the output neuron function for travel time estimation. Final, the travel time estimates at time interval $t+n$, $\hat{T}_{d'd''}^{(0)}(t+n)$, are obtained through IAGO.

The learning algorithm in the ENN model is the same as that in the backpropagation networks, using the gradient descent rule, which adjusts the weights based on the derivatives of the error with respect to the weights. The entire input sequence is presented to the ENN network, and its outputs are calculated and compare with the target sequence to generate an error sequence. Mean squared error (MSE) is used as a performance function in training the ENN model. For each time step (epoch), the error is backpropagated to find gradients of errors for each weight and bias. The gradient is actually an approximation since the contributions of weights and biases to errors via the delayed recurrent connection are ignored. The gradient is then used to update the weights with the backpropagation training function. The grey-based ENN model is trained offline for target sequences. The training process continues until the prediction errors (MSE) become very small and the network parameters converge to values that allow it to perform the desire mapping for each input-output correlation. Once the grey-based ENN is trained and its performance is shown to be satisfactory, it can then be used on-line to provide link travel time estimation based on new real-time data.

D. *Simulations*

We simulate ENNS by using the neural-network-toolbox in MATLAB. The function 'elmannet' in network-toolbox of MATLAB allows users to create an Elman network.

Utilizing the neural-network-toolbox is MATLAB is easy, since the structure of well-known networks is already implemented, and we only need to customize this structure and finally feed the network with the prepared data. Briefly speaking, we first collect the data, then we create the structure of the network (Elman network in our simulations), then we train the network with 60% of the collected data, test it with 20%, and finally perform the validation step with the remaining 20%. Our results contain the average Mean Squared Error (MSE) and number of epochs over 100 numbers of trials.

Since we do not access the real data. provided by the authors in[15], we have synthesized the data for all the six detectors using speed and volume features, and modeled travel time for them. Then, we have trained an ENN with the data gained from three of the detectors, and tested the network with the remaining ones. Table 1 shows the parameters we used for synthesizing the data. Using this parameters and including a normally distributed pseudorandom number generator, we used equations (13) and (14) for gaining the speed and volume features.

TABLE. 1 Parameters used for synthesizing the traffic data

Parameters	Values
Speed Mean	50 (km/h)
Speed Standard Deviation	15 (km/h)
Volume Mean	70 (Cars)
Volume Standard Deviation	30 (Cars)
Speed Effect Constant	3600 (Seconds)
Volume Effect Constant	1800 (Seconds)
Minimum Travel Time	216 (Seconds)

$$s_t^{[i]} = |\, N(\mu_s, \sigma_s)\,| \tag{13}$$

$$v_t^{[i]} = |\, N(\mu_v, \sigma_v)\,| \tag{14}$$

where μ_s and μ_v are speed and volume means respectively, and σ_s and σ_v are their standard deviations. Defining the impact of speed and volume to the travel time, we used equation (15) to calculate the travel time.

$$r_t^{[i]} = \frac{(c_1 + c_2) v_t^{[i]} \max(s^{[i]})}{s_t^{[i]} \max(v^{[i]})} \tag{15}$$

in which $r_t^{[i]}$ is the calculated travel time at time step t by ith detector, and $\max(s^{[i]})$ and $\max(v^{[i]})$ are the measured maximum of speed and volume by that detector, respectively.

After creating the data, we then modeled the proposed ENN structure in the paper using MATLAB. The structure of the network is based on what is presented in[15], and is shown in Fig.7.

We used 'newelm' and 'elmnet' function sof MATLAB to model the RNN. This model is shown in Fig.8.

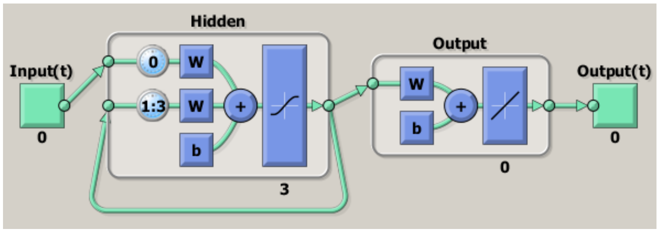

Figure.8 Structure of our modeled RNN in MATLAB

Using the synthesized data obtained in previous steps and after training the network with them, we tested its performance by calculating the Mean Squared Error (MSE), number of epochs needed for training the network, and that is averaged over 100 numbers of trials. Table 2 briefly shows this information.

TABLE. 2 Results attributes of the trained RNN and its performance

Parameters	Values
Averaged MSE	0.001753
Best MSE	0.001532
Average Number of Epochs	64
Best Number of Epochs	52

Although the results provided in Table 2 seems promising, but it took a rather long time to obtain them. Basically, high computational burden is one of the major problems in Recurrent Neural Networks that has kept them far from being fully utilized by researchers. In this simple example, we were working with a 2 dimensional feature space, and for 100 trials it took us some time around an hour to train the network on a Core i7 3.4 GHz processor platform with 16GB of internal memory. However, lowering the computational complexity of RNNs is still an open problem that can be assumed as one of the future works in this research area.

V. Conclusion

In this paper analysis report, we briefly introduced Recurrent Dynamic Neural Networks (RNNs), and presented its history. Moreover, we discussed about the Real Time Recurrent Learning (RTRL), and the methods RNNs are trained. Furthermore, we showed some applications and examples of neural networks, and finally we simulated one of those applications which uses ENN to predict the travel time in a particular highway. Computational complexity is the main bottleneck of Recurrent Neural Networks which can be considered as one of the future works in this area.

References

[1] Sinha, N.K., "Dynamic neural networks: an overview," in *Industrial Tech 2000 Proceedings of IEEE International Conf on*, vol.2, pp. 491-496.

[2] J. J. Hopfield, "Artificial Neural Networks are Coming", *IEEE Expert* , An Intewiew by W. Myers, April 1990, pp. 3-6.

[3] Jason M. Eisner, "Dynamical-Systems Behavior in Recurrent and Non-Recurren Connectionist Nets," B.A. thesis, Dept. Psychology, Harvard Univ., Cambridge, MA, 1990.

[4] J.A Anderson, J.W. Silverstein, S.A. Ritz and R.S. Jones, "Distinctive Features, Categorical Perception, and Probability Learning: Some Applications of a Neural Model," *Psychological Review*, 1977, vol.84, pp. 413-451

[5] T. Sejnowski and C.R. Rosenberg, "NETtalk: A Neural Network That Learns to Read Aloud," *Tech. Report*, JHU/EECS-86/01, John Hopkins Univ., 1986.

[6] A. Waibel, T. Hanazawa, G. Hinton, K.Shinkano and K.J. Lang, "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Trans. on Acoustics, Speech and Signal Processing*, Vol.37, No.2, pp.328-339, March 1989

[7] K.S.Narendra and K.Parthasarthy, "Identification and Control of Dynamicl Systems Using Neural Networks," *IEEE Trans. on Neural Network*, March 1900, Vol.1, No.1, pp.4-27.

[8] B.A. Pearlmutter, "Dynamic Recurrent Neural Networks," CMU-CD-90-196, School of Comp. Sci. CMU, PA,1990.

[9] Kolen, J.F., and Kremer, A Field Guide to Dynamical Recurrent Networks, IEEE Press, 2001

[10] W.J. Wang, Y.F. Liao and S.H. Chen, "RNN-based prosodic modeling for mandarin speech and its application to speech-to text conversion, " *Speech Communication*, Vol.36, pp. 247-265

[11] Kermanshashi, "Recurrent neural network for forecasting next 10 years loads of nine Japanese utilities" *Neurocomputing*, Vol.23, pp. 125-133

[12] Y.-horng Wen and T.-tian Lee, "Missing Data Treatment And Data Fusion Toward Travel Time Estimation For ATIS," *Journal of the Eastern Asia Society for Transportation Studies*, vol. 6, pp. 2546-2560, 2005.

[13] Dailey, D.J., "Travel time estimation using cross-correlation techniques," *Transportation Research-B*, Vol.27, pp. 97-107

[14] Cremer, M. and Schutt, H., "A comprehensive concept for simultaneous state observation, parameter estimation and incident detection," in 12th international Symposium on Transportation and Traffic Theory, 1900

[15] Nam, D.H. and Drew, D.R., "Traffic dynamics: method for estimating freeway travel times in real time from flow measurements," Journal of Transportation Engineering, Vol.122, pp.185-191

[16] P. Nelson and P. Palacharla, "A neural network model for data fusion in ADVANCE," 1993 Transtech Pacific Rim Conference, Seattle, July 1993.

[17] A.Dharia and H.Adeli, "Neural network model for rapid forecasting of freeway link travel time," Engineering Applications of Artificial Intelligence, Vol.16, pp. 607-613

[18] J.W.C. Lint, S.P. Hoogendorn, H.J. Zuylen, "Freeway travel time prediction with state-space neural networks," Transportation Research Records, No.1811, pp.30-39.

[19] H. Ma, A.A. El-Keib and X. Ma, "Training data sensitivity for artificial neural network-based power load forecasting," Proc.26th Southern Synposium on System Theory, pp. 650-652