Yet another Representation for System Dynamics Models, and Its Advantages

Chen, Yao-Tsung
Institute of Information Science, Academia Sinica
No. 128, Sec. 2, Yanjiouyuan Rd., Taipei City 115, Taiwan
Tel:+886-2-27883799 ext. 2401 Fax:+886-2-27824814
ytchen@mail.iis.sinica.edu.tw

Jeng, Bingchiang
Department of Information Management, National Sun Yat-sen University,
70 Lien-hai Rd., Kaohsiung 804, Taiwan
Tel:+886-7-5252000 ext. 4719 Fax: +886-7-5254799
jeng@mail.nsysu.edu.tw

Yet another Representation for system Dynamics Models, and its Advantages

Abstract

The present paper relates to an artificial neural network (ANN) representation for a system dynamics model (SDM) and its advantages in model construction and policy design. The similarities between SDMs and ANNs have been noted, i.e., both of which store knowledge mainly in the structure (or linkages) of a model, rather than in the units or other components. By a specially designed mapping scheme, it is shown that a given flow diagram (FD) (i.e., traditional representation for a SDM) can be mapped to a corresponding model in the representation of partial recurrent networks (PRNs) that will correctly behave like the one it mimics. Because a (partial recurrent) neural network can be trained with exemplar data, numeric propagation constraints can be identified by extracting rules from a set of multivariate time series of data by induction. This adds an advantage to the study of SD since it is made possible to create a model by learning instead of manual construction, which solely relied on experts' observation and deduction quality. Similarly, it is also beneficial to policy design. By assigning an intended behavior pattern as a set of training examples for a given SDM, it can learn a new system structure that fits the data; the differences between the original and new structures lead to considerations of policy design. In short, the neural representation for SDMs provides a new dimension of studying SD, and some traditionally hard problems in a FD might now be solved easier in the new representation. It is proved in mathematics that the two model representations operate under the same numerical propagation constraints.

Keywords: model representation, model mapping, neural network, propagation of constraints, machine learning

1. Introduction

The emergence of system dynamics (SD) science can be dated back earlier in the publication of book "**Industrial Dynamics**" by Jay W. Forrester at 1961 [Forrester, 1961], in which it is defined as "the study of the information-feedback characteristics of industrial activity to show how organizational structure, amplification (in policies), and time delays (in

decisions and actions) interact to influence the success of the enterprise". Feedback structures and time delays usually play a key role in the behavior of such a system that makes it complicated and hard to understand. Forrester later mentioned, "feedback processes emerged as universal in social systems and seemed to hold the key to structuring and clarifying relationships that had remained baffling and contradictory" [Forrester, 1968a]. Therefore, SD can be viewed as a structuring theory for systems (particularly close-loop systems), in which one can construct and analyze the dynamic feedback model of a social system in a systematic way [Starr, 1980].

To enable the study, a number of representation forms had been proposed for modeling, which have different focuses, strengths, and weaknesses. The most often used are flow diagrams (FD) and causal loop diagrams (CLD) in different levels of abstraction. These diagrams usually consist of a set of graphical icons to denote the semantics of various system components so that a constructor can use them to conceptualize a model and communicate with related people. Although suitable for conceptualization and simulation, these representations are not good for manipulation; it is hard to do transformation and/or reduction in the diagrams themselves. We, however, think this functionality, if exists, will be useful and might provide a new perspective in problem solving.

In the following, we will present a new representation form, i.e., a specially designed partial recurrent network (PRN), and see how to use the FD-PRN model mapping to transform a FD into it. The idea of this new representation comes from the observation of an important similarity between the system dynamics models (SDMs) and artificial neural networks (ANNs); they both store knowledge mainly in the structure of a model, not in the units but in the links between units. By establishing a mapping scheme, it is shown that a given FD can be related to a corresponding PRN that will correctly behave like the one it mimics.

The equivalence between the two types of model representations will be shown both in structure and in mathematics. It will be seen that with the new representation form, one will have an additional choice of using which model representation in problem solving, and some traditionally hard problems in SD research, e.g. model construction, policy design, etc. can now be solved easier in the new representation. As an example, we will illustrate how to use this model representation to learn a SDM by induction from a set of prepared data.

The remainder of this paper is organized as follows. Section 2 briefly reviews the representations that previously used in SD and introduces the concepts of artificial neural networks. Section 3 describes the design concepts of the new representation model and the FD-PRN mapping to show its relationship with a FD. Section 4 illustrates a simple example to show how to use the new representation to assist the construction of a SDM. The comparison of FD and PRN is discussed in Section 5, which concludes the paper.

2. System Dynamics Models and Artificial Neural Networks

2.1. The representation of a SDM

In its purest form, a SDM consists of merely a set of assumptions describing a problematic situation. To simulate the consequences of the assumptions, a model is formulated as a set of equations and coded in a computer program. In order to conceptualize and represent the underlying models, representations for SD typically use a set of graphic icons to denote the semantics of various system components.

Two forms of representation are overwhelmingly used in the SD community. A simpler one is CLD that focus on the representation of variables and loop structures of a model. In contrast, FD is more detailed, discriminating both state and flow variables [Lane, 2000].

In history, FDs is the first representation form appeared in Forrester's first SD book [Forrester, 1961], which consists of a set of symbols for "levels", "rates", "information links" and "conserved flows" diagrammatically to describe a "flow diagram". This diagram is the root of today's FD implementation found in various software packages (e.g., Stella, i-Think, etc.). There is no CLD in that book.

An early form of CLD is first described in Forrester's later publication [Forrester, 1968c], in which it is used as a means of summarizing and explaining the behavior of a specified simulation model. That is, CLDs were used near the end of a study to represent the structure of dominant loops deduced during simulation. In the following years, however, CLDs were frequently suggested as the initial step in model conceptualization to quickly sketch the structure of a target model [Goodman, 1974; Coyle, 1977; Randers, 1980; Richardson and Pugh, 1981; Roberts and et al., 1983].

Lane [2000] compares the two representation forms as follows. As a conceptualization tool, the advantage of CLDs is its simplicity and clarity to those who use them. A very limited library of symbols is involved and the concentration is on loop structures within a model. (A famous usage is the description of archetypes shown in Peter Senge's book [Senge, 1990].) The deficiencies of CLDs are that they do not always explain well how "flows" influence "stocks" and can lead to mislabeling of loops.

In comparison with a CLD, the advantages of a FD are numerous. It carries more information about an underlying model. It graphically displays the relationships between stocks and flows, and distinguishes the important difference between conservative flows and information links. Therefore, it provides a sounder basis for the rigorous deduction of dynamic behavior. The disadvantage of a FD is in its very details and specificities that can obscure the loop structures for a large model. It is also difficult to be introduced in a group discussion with a broad range of individuals.

To combine the advantages of both representation forms, Burns and et al. [Burns, 1977; Burns and et al., 1979; Ramos, 1983] presented an algorithm for converting signed digraphs (i.e., CLDs) into Forrester schematic (i.e., FDs). Richardson [1986] identified the problem of traditional definitions of positive and negative links in CLDs and suggested a modified CLD that distinguishing additive (rate to level) from proportional (information) links with improved definitions of positive and negative links. Some literature called the modified CLDs "influence diagrams" [Coyle, 1977][1] and viewed them lying somewhere between CLDs and FDs. However, this term would be interpreted by the majority of system dynamic scientists as a synonym for CLDs [Richardson, 1991].

Another extension of influence diagrams is Morecroft's notion of "policy structure diagram" with explicit rates and levels [Morecroft, 1982; 1985]. It elaborates some important levels and rates appearing in a portion (or sector) of a complex system with an aggregated view of information connections into those levels and rates. It highlights key policy areas in a sector and shows how information links combined to determine the decisions under these policies. The strengths of a policy structure diagram is: (1) an appropriate level of detail for model users, i.e., explicit representation on stocks and flows, which helps to sensitize users to the importance of accumulations in the dynamics of a system; and (2) a focus on policy, which matches a user's locus of concern. Its main weakness is the tendency to obscure the existence and the character of feedback loops in a system [Richardson, 1991]. Richardson also mentioned that the use of policy structure diagrams in SD literature is limited.

Another kind of representation useful for both conceptualization and communication is "subsystem diagram" [Morecroft, 1982] which is a context diagram for an overview of model and sector boundaries. Such a diagram strives to capture the organization of a simulation model, to show a summary of what is included and excluded, and to show the main interconnections among the identifiable sectors in a system. It is a visual outline of the system as well as the model formulated for policy analysis [Richardson, 1991]. STELLA and i-Think adopt these diagrams on top of FDs [Richmond and et al., 1987].

Along the trend of the above research that strives to find a better representation form for a SDM, we will present another one here. However, in difference with those that emphasize on designing a suitable set of model components and notations for conceptualization and communication, this representation will be emphasized on the model's manipulability for easier problem solving (e.g., model construction, policy design, etc.). We found that artificial neural network is a suitable candidate, and a brief introduction is given in the following.

---

[1] This usage is distinct from that in decision analysis [Oliver and Smith, 1990].

2.2. Artificial neural networks

Artificial neural networks (ANNs) are a kind of knowledge representation form that has been studied for many years by artificial intelligence scientists. Like SDMs, they also store knowledge in system structures rather than units. This is to mimic the structure of a biological brain, in which it consists of a large set of brain cells inter-connected to form a complicated structure and electrical messages propagate within this structure in order to response to outside world's stimuli. An artificial neural network can be simulated by a program easily.

To use an ANN for problem solving, one usually needs first to decide the structure of the network. Different types of problems need different structures; related issues to be considered are: the network types, the number of hidden layers, and the number of units for each layer. A typical structure of such a network is shown in Fig. 1, where numeric data all propagate in one direction toward the output layer and there is no feedback. This type of network (called feed-forward network) is suitable for problems where outputs are only dependent on inputs and nothing else. Once the initial network is created, it enters a learning phase in which one has to decide a training data set, the learning rate parameter, and the convergence of the network. After the training phase, it has to evaluate whether the created network has solved the problem.

We will use a special type of ANNs called partial recurrent networks (PRNs) in this paper. According to Elman's definition [1990], PRN is a kind of ANN with recurrent links that were used to associate a static pattern (a 'Plan') with a serially ordered output pattern (a sequence of 'Actions'). Figs. 2 and 3 are examples of such PRNs, in which there is a new type of units called state units occurring in the input layer. Jordan's network [1986] connects the output units to these state units directly (i.e., recurrent inputs) as shown in Fig. 2. Elman's network [1990] renames the state units as context units and allows the connections of recurrent links to each layer within a network as shown in Fig.3.

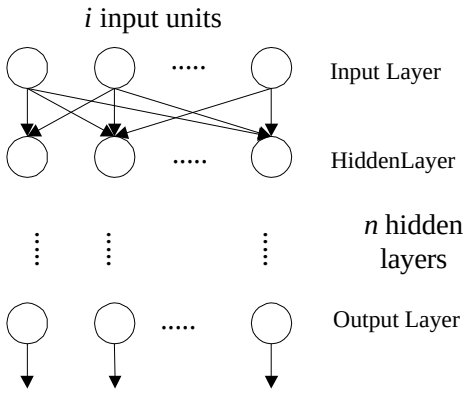


Fig. 1 a typical feed-forward ANN

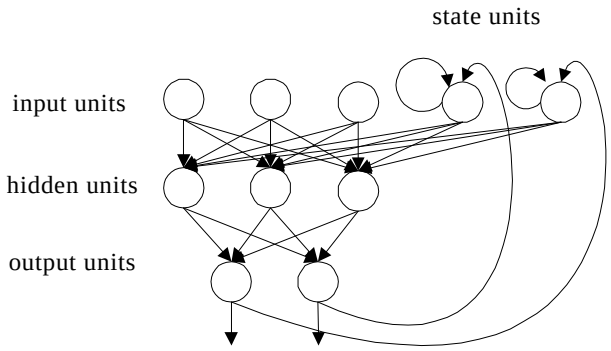


Fig. 2 a Jordan network

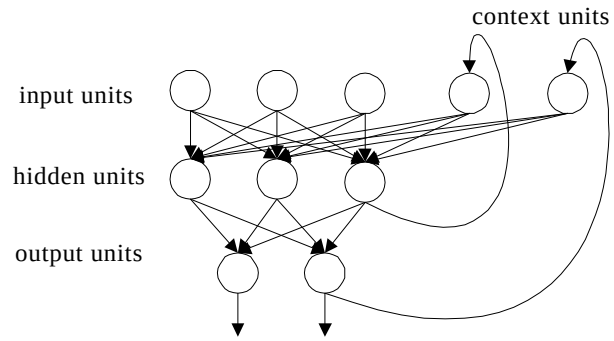


Fig. 3 an Elman network

3. The FD-PRN Mapping

Because of the cause-effect relationships and its graphic representation, system dynamic models are classified as a structural modeling technique. Dolado [1992], from a different aspect, showed that the structure of FD represents a general scheme that propagates numeric constraints. Members belonging to this group also include other types of models such as ANNs that propagate numeric constraints via a network of computing units and links. Therefore, it would be interested to see if there is a way to use the latter representation forms as a modeling tool for system dynamics since they have a unique capability of learning structures from given exemplar data. An approach to verify this is to see if a correspondence mapping exists that can relate a neural network representation to any of a SD representation both in structure and in numeric constraints. If it does, then we have shown that a neural network can also be a model for SD. The following sections will explore this idea and describe the details. In particular, we will establish a mapping scheme that relates a FD to a PRN (and vice versa) and show the advantages of the new representation in some particular applications.

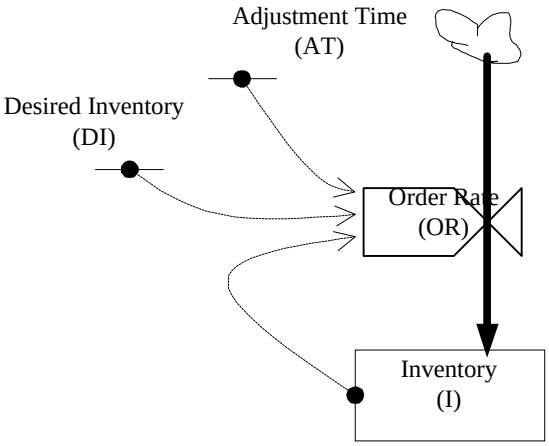


Fig. 4 an Inventory Model

3.1. Mapping in structure

We'll use a FD as the representation for a SDM and a PRN as the candidate representation to be studied. Let us start from a simple FD for an inventory control system shown in Fig. 4. Within this model, there is a decision point (Order **Rate**) that controls the **flow** into a **level** (Inventory). Note that a **flow** is always coupled with a **rate**. There must be exactly one **rate** on each **flow**, and no **flow** can be present without a corresponding **rate**. The model is classified as a first-order system [Forrester, 1968b] since it has only one level variable, which maintains the system's memory. It describes an inventory control system in which there is no delay between the ordering and receiving of goods. The function of the order rate (*OR*) is to bring the actual inventory (*I*) to a desired inventory level (*DI*). If the actual inventory level is below the target, the order rate increases; otherwise, it decreases. The difference between *DI* and *I* should be adjusted within time interval *AT*, in which *DI* and *AT* are all constants and propagated to *OR* through **wires**.

The numeric equations/constraints related to the system in Fig. 4 are the following:

$I(t) = I(t - DT) + (OR) \times DT$

$OR = (1/AT) \times (DI - I)$

$DI = 6000$

$AT = 5$

Now let us take a close look at each part of the FD, and see whether a mapping between this form of representation and a PRN is possible. The most important and obvious components in a FD are "**levels**", whose function is to exchange information with the outside world and keep a memory of the state of a system; that is, accept an initial value before simulation and accumulate the result after each time step. From the previous description for a PRN, there is no single component in a neural network that corresponds to such a level component. However, the functions of a level can be distributed among three types of

components in a PRN: an input unit, an output unit, and a state unit. The pair of an input and an output unit serves as an interface, in which data may be fed in or retrieved out of the network, respectively. The state unit serves the other function of a level, and keeps the previous value of an output unit in a network (i.e., the state of the network). In other words, each input unit receives an initial value in the beginning of a simulation and propagates the value to its corresponding output unit through the network structure. Then the output unit forwards and stores its result to the related state unit and through which to the corresponding hidden unit as another round of input value.

The second important components in a FD are "**rates**", whose function is to control the amount of flow into or out of a level at each time step. Discovering the existence of the rate on a "**flow**" is not always easy. It usually relies on the skill and insight of a model constructor. This is also true for a hidden unit in a neural network, which hides inside and defines a function from a related input stimulus to an output unit. In addition, the number of hidden units required in a network is also dependent on the experience of a network constructor. Therefore, it is natural to map a rate component to a hidden unit and its associated flow as a link between a hidden unit and an output unit in a PRN. (**"Auxiliary"** components will not be discussed here – they are optional in a FD and can always be a subdivided part of a rate equation, which can be treated like a "rate in front of another rate" in the mapping.)

The third type of component in a FD is a "**wire**", which is simply a connection between a rate and some information source like a level or a "**constant**". The mapping is easy, which is a link between a state and a hidden unit in a corresponding PRN.

Depending on its usage in a FD, the mapping of a "**constant**" can be either treated like a (constant) level or viewed as a coefficient in a rate function (explained later in the next section).

The algorithm (FD2PRN) that physically implements the above mapping is described in Fig. 5. The input to the algorithm is a FD while its output is a PRN. Without lost of generality, it assumes that a FD is only composed of levels, rates, flows, wires, constants, and system boundaries. (System boundaries have no physical meaning in a PRN.) Other modeling components not given here are all derivable from these basic components. So, the generated PRN by the algorithm will be expressive enough to cover any kind of FDs.

```
FD2PRN (FD) return PRN
//
//FD: a Flow Diagram
//PRN: a Partial Recurrent Network
//Act_IDENTITY: the identity function as an activation function
//Out_IDENTITY: the identity function as an output function
//
Set default activation function Act_IDENTITY
Set default output function Out_IDENTITY
For each level or constant in FD
    Generate an input unit I
    Generate an output unit O
    Generate a state unit S
    Connect a link LIO from I to O
      Set the weight of LIO 1
    Connect a link LSO from S to O
      Set the weight of LSO 1
    Connect a link LOS from O to S
      Set the weight of LOS 1
For each rate DR in FD
    Generate a hidden unit NR
      If the start point of the flow that DR is upon is a level LV1
        Connect a link LHO1 from NR to the output unit corresponding LV1
          Assign the weight of LHO1 with -DT
      If the end point of the flow that DR is upon is a level LV2
        Connect a link LHO2 from NR to the output unit corresponding LV2
          Assign the weight of LHO2 with DT
      For each information source IS in the rate equation DRE of DR
        Connect a link LSH from the corresponding state unit for IS to NR
          Assign the weight of LSH with the coefficient of IS in DRE
```

Fig. 5 the algorithm of mapping a FD to a PRN (FD2PRN)

Using Fig. 4 as an example, the output of the FD2PRN algorithm will be like the one shown in Fig. 6(c); the relationships of the corresponding components between the two representations described above are listed in Table 1. As shown in Fig. 6(a), **level** (inventory: *I*) and **constant** (desired inventory: *DI*) are mapped to three units: **input** I_I, **output** O_I, **state** S_I and **input** I_{DI}, **output** O_{DI}, **state** $S_{DI,}$ respectively. **Rate** (order rate: *OR*) is mapped to hidden unit H_{OR}, and the **flow** is mapped to the link from H_{OR} to O_I, as shown in Fig. 6(b). The other type of **constants** (e.g., adjust time: *AT*) that appear as a coefficient in a rate equation is mapped to the weights of the links from S_I to H_{OR} and from S_{DI} to $H_{OR,}$ respectively, as shown in Fig. 6(c).

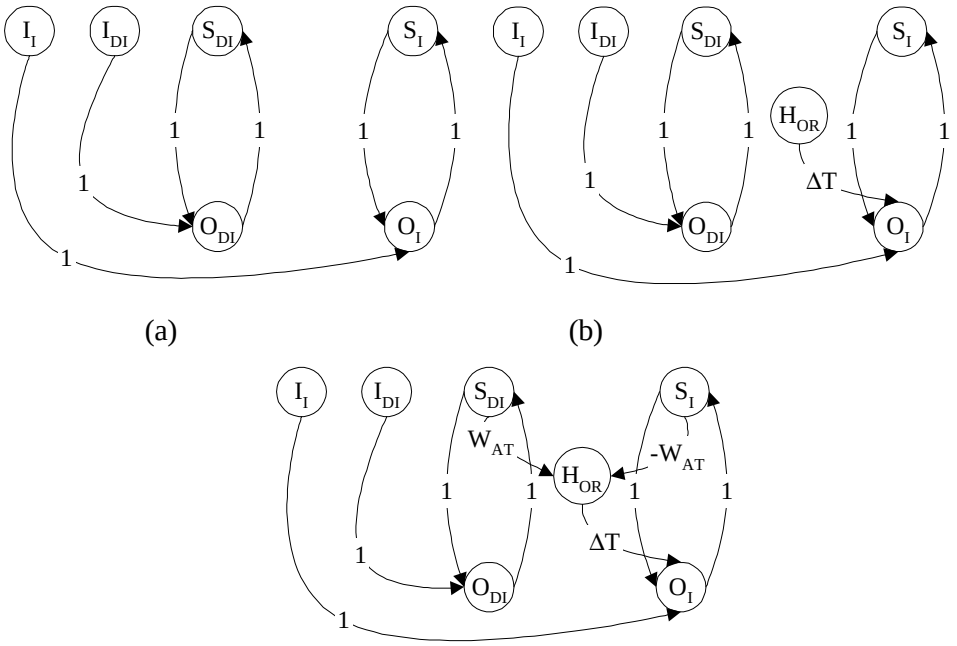


Fig. 6 the PRN corresponding to the SDM of Fig. 4

Table 1 the corresponding components between a SDM and a PRN

Components for SDMs	Components for PRNs
Level variable, constant (not for coefficient)	A triple of input, output, and state units
Rate (or auxiliary) variable	Hidden unit
Wire	Link from a state unit to a hidden unit
Flow	Link from a hidden unit to output unit
Level equation	A weighted sum of the values of hidden and state units connecting to an output unit via links
Rate equation (including constants as coefficients)	A weighted sum of the values of state units connecting to a hidden unit via links
Equation for initial value	Link from an input unit to an output unit
Constant equation	Link from a state unit to an output unit

3.2. Mapping in numerical propagation constraints

Not shown in Fig. 6 are the values that propagate within the network. Are they also same with the corresponding ones in the FD? To verify this, we need to show mathematically that the two representation forms involved in the mapping are equivalent. According to Dolado [1992], a FD represents a set of numeric propagation constraints, in which the intrinsic part is composed of the internal equations of **levels** and **rates** while the extrinsic part is composed of the initial values of **constants** and **levels**. On the other hand, the intrinsic constraints of a PRN are defined by the internal activation function of each unit and the weights on links between two units, while the extrinsic constraints are network inputs. Only if the constraints of the two representations are shown equivalent, one can claim that the two models operate and propagate numeric constraints in the same way with no difference. In the

following, the equivalence of each individual constraint is proved.

3.2.1. Level equations

Forrester [1968b] defines a **level equation** as "a reservoir to accumulate the rates of flow that increase and decrease the content of the reservoir". Thus, the final value of a level is the accumulation of changes within a certain period. A level equation in Forrester's form is as (Eq. 1):

$$L.K = L.J + DT * (RA.JK - RS.JK) \qquad \text{(Eq.1)}$$

where

L is the level,
L.K is the level L's new value,
L.J is the level L's old value,
DT is the period between JK,
RA is the rate on an inbound flow into the level,
RA.JK is the rate value increases between time J and K,
RS is the rate on an outbound flow out of the level, and
RS.JK is the rate value decreases between time J and K.

The equation can be rewritten in a more easily understandable form:

$$L(t) = L(t\text{-}1) + \Delta T \left(\sum_{i=1}^{m} r_i(t-1) - \sum_{j=1}^{n} r_j(t-1)\right), t = 0,1,2,\ldots,n \qquad \text{(Eq. 2)}$$

where

L is the level
$L(t)$ is level *L*'s new value at time *t*,
$L(t-1)$ is level *L*'s old value at time $t-1$,
DT is the time interval of the calculation,
r_i is the rate of an inbound flow into the level,
$r_i(t-1)$ is the rate value between time $t-1$ and *t*,
m is the number of rates of the inbound flow into the level,
r_j is the rate of an outbound flow out of the level,
$r_j(t-1)$ is the rate value between time $t-1$ and *t*, and
n is the number of rates of the outbound flow out of the level.

The corresponding level equation in a PRN is referred to the value of an output unit, which is determined by a weighted sum of output values from the hidden and state units connected to the output unit via links. To show how this part of numerical constraints between the two representations is equivalent, let us start first from the output function of an

output unit, which is

$$a_k(t)=I(net_k(t)) \qquad \text{(Eq. 3)}$$

where

$a_k(t)$ is the output value of the k^{th} output unit at time t,
$net_k(t)$ is the net input to the k^{th} output unit at time t, and
$I(\ )$ is the identity function.

Note that to match the characteristics of a SDM, the PRN model here always uses the identify function as the activation function that passes the net input of a unit to output directly. The net input $net_k(t)$ is calculated as follows:

$$net_k(t) = \dot{w}_k I_k(t) + \ddot{w}_k S_k(t) + \sum_h w_{hk} H_h(t) \qquad \text{(Eq. 4)}$$

where

$net_k(t)$ is the net input for the k^{th} output unit at time t,
k is the index of the k^{th} output unit (also corresponding to the k^{th} level),
$\dot{w}_k$ is the weight of the link from k^{th} input unit to k^{th} output unit,
$I_k(t)$ is the k^{th} input unit's output value at time t,
$\ddot{w}_k$ is the weight of the link from k^{th} state unit to k^{th} output unit,
$S_k(t)$ is the k^{th} state unit's output at time t,
h is the index of the h^{th} hidden unit (also corresponding to the h^{th} rate),
w_{hk} is the weight of the link from the h^{th} hidden unit to the k^{th} output unit, and
$H_h(t)$ is the h^{th} hidden unit's output at time t.

Close examination of Fig. 7 reveals that the incoming links connecting an output unit are divided into two groups: (1) from input or state units and (2) from hidden units. The former set of links is always assigned with a weight equal to one, so that the initial input values are propagated to output units directly and then forwarded to state units. After that, the values of state units are used as a new set of inputs that feed in again to propagate to output units, and this process repeats at each time interval to ensure that the previous system outputs are maintained. This part of mapping corresponds to the levels that accumulate old values in the last step. The weights of links from hidden units to an output unit receive either DT or $-DT$ so that the product values represent the net changes of rate values into output units. Now substitute these weight values into Eq. 4 and simplify the equation:

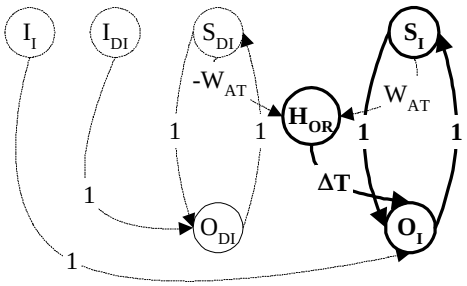


Fig. 7 the corresponding part of a PRN for the level equation of I in Fig. 4

$$net_k(t) = I_k(t) + S_k(t) + \Delta T(\sum_i H_i(t) - \sum_j H_j(t)), i \neq j \wedge t = 0,1,2,...,n \quad \text{(Eq. 5)}$$

where

$net_k(t)$ is the net input for the k^{th} output unit at time t,

k is the index of the k^{th} output unit (also corresponding to the k^{th} level),

$I_k(t)$ is the k^{th} input unit's output value at time t,

$S_k(t)$ is the k^{th} state unit's output at time t,

DT is the weight,

i is the index of the i^{th} hidden unit (also corresponding to the i^{th} rate of the inbound flow into the level),

j is the index of the j^{th} hidden unit (also corresponding to the j^{th} rate of the outbound flow out of the level),

$H_i(t)$ is the i^{th} hidden unit's output at time t, and

$H_j(t)$ is the j^{th} hidden unit's output at time t.

Input $I_k(t)$ is also restricted to carrying values only at step 0 and is reset to zero otherwise. (The reason for this and the method used will be described in the next section.) In contrast, $S_k(t)$, $H_i(t)$, and $H_j(t)$ will receive a zero value at step 0, and any values after that. In addition, $S_k(t)$ represents the current value of the k^{th} state unit at time t as well as the output value of the k^{th} output unit at time t–1. Substituting these values into Eq. 5 results in the following:

$$net_k(0) = I_k(0) \quad \text{(Eq. 6-1)}$$

$$net_k(t) = net_k(t-1) + \Delta T(\sum_i H_i(t) - \sum_j H_j(t)), i \neq j \wedge t = 1,2,...,n \quad \text{(Eq. 6-2)}$$

where

$net_k(t)$ is the net input for the k^{th} output unit at time t,

k is the index of the k^{th} output unit (also corresponding to the k^{th} level),

$I_k(t)$ is the k^{th} input unit's output value at time t,

$net_k(t\text{-}1)$ is the net input for the k^{th} output unit at time $t\text{-}1$,

DT is the weight,

i is the index of the i^{th} hidden unit (also corresponding to the i^{th} rate of the inbound

flow into the level),

j is the index of the j^{th} hidden unit (also corresponding to the j^{th} rate of the outbound flow out of the level),

$H_i(t)$ is the i^{th} hidden unit's output at time t, and

$H_j(t)$ is the j^{th} hidden unit's output at time t.

The above analysis shows that the output functions for a PRN can be expressed as Eqs. 6-1 and 6-2, which can be compared to Forrester's form of level equations (Eq. 2) rewritten from (Eq. 1). On the condition that t = 1, 2, ..., n, Eqs. 2 and 6-2 are identical. This shows that the numeric constraints defined in a level equation can be re-implemented in a PRN designed as above.

3.2.2. Initialization equations

In the previous section, it was shown that each input unit at step 0 propagates its value to the corresponding output unit, which assigns an initial value of a level (Fig. 8). In the subsequent steps, however, a PRN can still allow the input units to feed new values into the network. This is different from the situation in a FD, where each level is set to an initial value by an initialization equation before a simulation starts, and then let the numeric values alone propagate in the simulation process without any interference from the outside world [Forrester, 1968b]. (There are exceptions when a model constructor wants to manipulate some system variables or add noisy data to the system.) To mimic this behavior, one has to restrict the input units of PRNs so that they do not receive more data from the outside world after step 0. This requirement is achieved by a special arrangement of training cases in a data set, in which only the training tuple for step 0 is given initial values, while other tuples in the following steps all receive zero values in the input part.

3.2.3. Constant equations

In a FD, all constants are assigned values by constant equations [Forrester, 1968b]. In the corresponding PRN however, constants can be either appeared as coefficients in a rate equation or treated as levels (as shown in Fig. 9). The latter case is distinguished from a normal level only in that they are connected with no hidden unit, and do not change values in simulation. So, its equivalence proof is the same as that for the level equation.

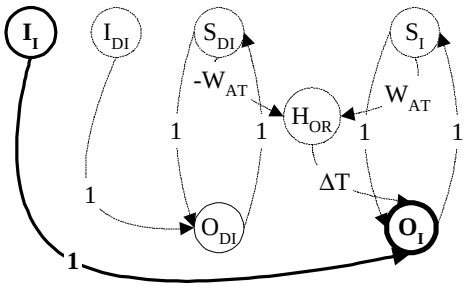


Fig. 8 the corresponding part of a PRN for the initial equation of I in Fig. 4

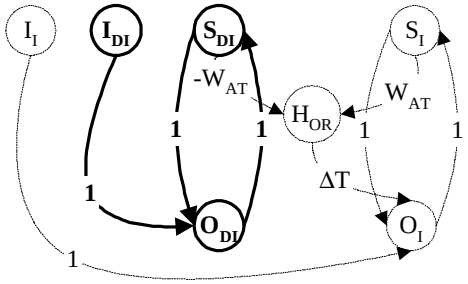


Fig. 9 the corresponding part of a PRN for the constant equation of DI in Fig. 4

3.2.4. Rate equations

A rate equation defines how a flow is controlled. It accepts inputs from levels or constants and generates an output that, in turn, controls a flow into or out of a level. A rate equation in Forrester's format is

$$R.KL = f(\text{all levels and constants}), \qquad \text{(Eq. 9)}$$

where

R.KL is the rate value in time interval *KL*.

The above equation can also be rewritten in a more general form as

$$r(t) = f(L_i(t),\ldots, C_j,\ldots),\ i=1, 2, \ldots, m,\quad j=1, 2, \ldots,n \qquad \text{(Eq.10)}$$

where

$r(t)$ is the rate value between t and $t+1$,

$f()$ is any function,

L_i is level i,

$L_i(t)$ is the level value at time t,

m is the number of levels,

C_j is the constant j, and

n is the number of constants.

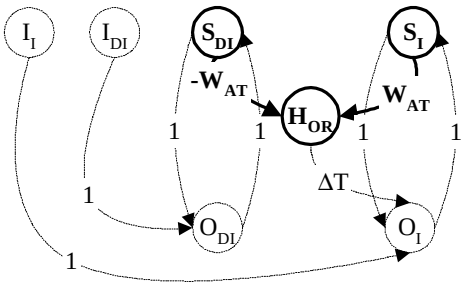


Fig. 10 the corresponding part of a PRN for the rate equation of OR in Fig. 4

A rate equation can be any type with the following restrictions: (1) an equation cannot contain constant *DT*; (2) the right-hand side of an equation should not include other rate variables, but only levels and constants; and (3) the left-hand side of an equation contains the rate variable being defined by the equation. An additional constraint to be noted is that the value of a rate variable is only affected by the outputs of levels in the previous time interval in a FD. Therefore, a level value within function *f* in the above equation is the old value of that level in the previous time step.

Since rates are mapped to hidden units in which the input links come from state units (as shown in Fig. 10), the three restrictions for rate equations are enforced because: (1) DT is only assigned to links from hidden units to output units, which has nothing to do with links from either input or state units to hidden units; (2) there is no connection between any two hidden units, so no part of a rate equation will be represented by another; and (3) the output of a hidden unit itself represents a corresponding rate value. As to the last restriction, the only inputs of a hidden unit are from state units according to the algorithm FD2PRN. If a state unit has come from the mapping of a level, then the state unit keeps the value of a level in the previous time step, which satisfies the constraint; otherwise, it is from a constant (which does not change value in time), and the constraint is satisfied trivially.

Besides the restrictions imposed on the rate equation, there is an additional one in the implementation of a PRN. A rate equation can be any arbitrary function. So it suffers a limitation if it has to be faithfully re-implemented in its original form in a PRN. That is, some functions cannot be trained in a PRN. As a requirement of a neural learning algorithm, an activation function has to be smooth and continuous in order to calculate its derivative value during a training process. Those functions (e.g., a look-up table) that do not satisfy the condition can exist in a PRN, but they will not participate in the learning process. This type of function, however, is usually provided with certain by a human constructor and occurs only in a small portion of a model. The most common rate equations are those that include only levels and constants in a weighted-sum format as illustrated below. An arbitrary weighted-sum equation may take the following form:

$$r(t) = a_1 L_1(t) + b_3 C_3, \qquad \text{(Eq.11)}$$

where

$r(t)$ is the rate value between t and $t+1$,

L_1 is the level *1*,

$L_1(t)$ is the level value at time t,

a_1 is the coefficient of L_1,

C_3 is the constant *3*, and

b_3 is the coefficient of C_3.

The equation for $r(t)$ that corresponds to the output of unit k is

$$net_h(t) = \sum_{i=1}^{m} u_{ih} L_i(t) + \sum_{j=1}^{n} v_{jh} C_j(t), i \neq j \qquad \text{(Eq. 12)}$$

where

$net_h(t)$ is the output for the h^{th} hidden unit at time t,

h is the index of the h^{th} hidden unit (also corresponding to the h^{th} rate),

u_{ih} is the weight of the link from the i^{th} state unit to the h^{th} hidden unit,

$L_i(t)$ is the i^{th} state unit's output at time t (also corresponding to the i^{th} level's value at time t),

v_{jh} is the weight of the link from the j^{th} state unit to the h^{th} hidden unit, and

$C_j(t)$ is the j^{th} state unit's output at time t (also corresponding to the i^{th} constant's value).

Let $u_{ik} = 0$ for $i = 2,3,\ldots,m$, and $v_{jk} = 0$ for $j = 1,2,4,\ldots,n$. Then Eq. 12 becomes:

$$net_h(t) = u_{1h} L_1(t) + v_{3h} C_3(t) \qquad \text{(Eq. 13)}$$

where

$net_h(t)$ is the output for the h^{th} hidden unit at time t,

h is the index of the h^{th} hidden unit (also corresponding to the h^{th} rate),

u_{1h} is the weight of the link from the 1^{st} state unit to the h^{th} hidden unit,

$L_1(t)$ is the 1^{st} state unit's output at time t (also corresponding to the 1^{st} level's value at time t),

v_{3h} is the weight of the link from the 3^{rd} state unit to the h^{th} hidden unit, and

$C_3(t)$ is the 3^{rd} state unit's output at time t (also corresponding to the 3^{rd} constant's value).

One can see that the form of the rate equation in Eq. 11 is identical to that in Eq. 13. For rate equations in a product form, the result will be similar.

As a summary, one may check that the formulae for the PRN shown in Fig. 6(c) are rewritten as follows:

$O_I(t) = O_I(t-1) + H_{OR}(t-1) \times \Delta T$

$H_{OR}(t-1) = (1/5) \times [O_{DI}(t-1) - O_I(t-1)]$

$O_{DI} = 6000$

3.3. Mapping in physical simulation

In the above, it is shown structurally and mathematically that a FD can be mapped to a specially designed PRN. The experiment here will physically examine and evaluate the validity of this claim. First, an arbitrarily selected FD is created using STELLA[2], and then used to produce the output patterns of level variables over a time interval. Meanwhile, the FD2PRN algorithm is used to create a corresponding PRN from this FD with a one-to-one mapping in structures. With this new representation, another set of output patterns is produced which is compared against the original. There is currently no standard comparison method to evaluate the performance of the regenerated patterns, so criteria that are most frequently found in papers are adopted here (e.g., MSE, RMSE, MAE, MAPE, NMSE, and NRMSE). These criteria are commonly used to estimate the correctness of forecasting [Nie, 1997; Zhang and Hu, 1998; Aussem, 1999], and to measure the difference between a real value and an estimate of it. The equations are following:

$$\text{MSE} = \frac{\sum (y_t - \hat{y}_t)^2}{T}$$

$$\text{RMSE} = \sqrt{MSE}$$

$$\text{MAE} = \frac{\sum |y_t - \hat{y}_y|}{T}$$

$$\text{MAPE} = \frac{1}{T} \sum \left| \frac{y_t - \hat{y}_t}{y_t} \right| \times 100$$

$$\text{NMSE} = \frac{MSE}{s^2}$$

$$\text{NRMSE} = \frac{RMSE}{s}$$

where y_t is an output of a FD, $\hat{y}_t$ is the corresponding output of the PRN, t is the number of data points, and s^2 is the variance of the output time series pattern. The first three criteria are a kind of mean values while the last three are normalized with respect to y_t or s, respectively.

Four well-known FDs are adopted in the experiments here, as shown in Fig. 11. The first three (Fig. 11(a)–(c)) are found in [Forrester, 1968b] which were used to illustrate a first-order negative feedback loop, a second-order negative feedback loop, and a positive feedback loop, respectively. (The first model, i.e., Fig. 4, is already seen in this paper.) The fourth one is modified from an example model (named “Business”) given in the library of

[2] This is a standard software package used for the creation and simulation of a SDM since its introduction in 1985 [Richmond, 1987].

STELLA, in which there are many positive and negative feedback loops intermixed together. The coefficient constants within these models are rewritten and incorporated into rate equations; other constants are changed into levels without inbound or outbound flows. These modifications do not affect the behaviors of the models. The four models generate different numbers of data points: 50, 100, 50, and 100, respectively. One point represents an output for one DT time interval in a time series.

Table 2 shows the performance of pattern regeneration of each of the four models. One can see that all of the criteria indices receive a tiny value (of the order of 10^{-6}), which means that the regeneration patterns produced by the corresponding PRN are almost exactly the same as their original patterns, and the new model is merely another representation of the original one with the same structure. Thus, it is proved in simulation that the FD2PRN algorithm for a FD does generate an equivalent PRN. What is interesting from the results is that the regeneration effectiveness of a complicated model (e.g., model 4) is not necessarily worse than a simple one (e.g., model 3). This hints that pattern regeneration has nothing to do with either the number of variables or the number of data points; it only has something to do with the structure.

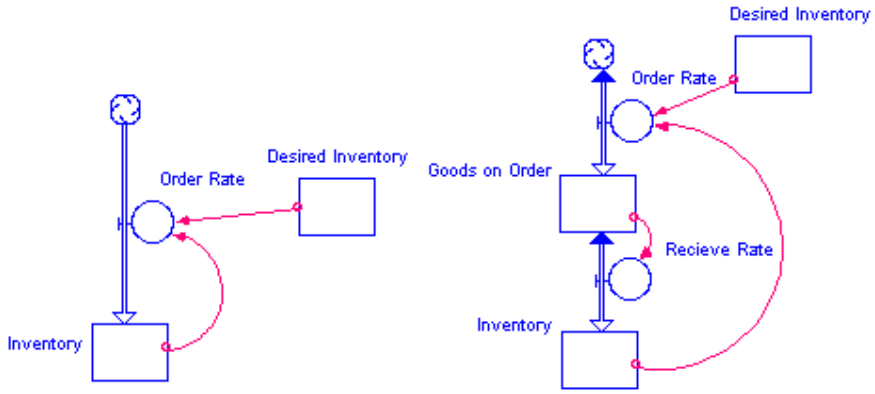
3.4. Reverse Mapping

Basically, the FD-PRN algorithm can be applied in either direction from FD to PRN or vice versa, and each row in Table 1 is a two-way mapping. However, this does not mean that any arbitrary PRN will have a correspondent in SDM. In order to satisfy the requirement, some conditions for a PRN should be fulfilled. First, the numbers of input units, output units and state units must be the same. Second, the same number of recurrent links exists which connects between an output unit and a state unit and the weights on the links must be 1. Third, there is the same number of links connecting between an input unit and an output unit and the weights on the links must be 1. Forth, the activation function for output units must be an identity function. Fifth, absolute values of weights on the links connected from hidden units to output units must be all the same. Note that the first four conditions tie the input, output and state units together to serve the role of levels in a SDM. The fifth condition mimic the simulation that each result is multiplied by the same DT value after one period.

Table 2 the Effectiveness of Pattern Regeneration by PRNs

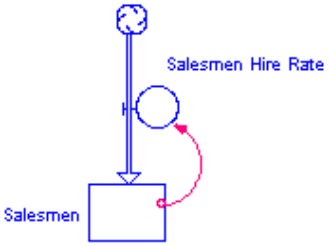
	Levels	MSE	RMSE	MAE	MAPE	NMSE	NRMSE
Model 1	I	0.00000006	0.24494897	0.06000000	0.10020955	0.00000516	2.27220024
Model 2	I	0.00000024	0.48989795	0.24000000	0.43631328	0.00001208	3.47628848
	GO	0.00000030	0.54772256	0.30000000	31.93470574	0.00001026	3.20332483
Model 3	S	0.00000032	0.56568542	0.32000000	3.25769262	0.00037884	19.46393932
Model 4	I	0.00000030	0.54772256	0.30000000	0.36676194	0.00011291	10.62581797
	ES	0.00000025	0.50000000	0.25000000	1.34740232	0.00019029	13.79456641

* All values in this table are already multiplied by 10^6.

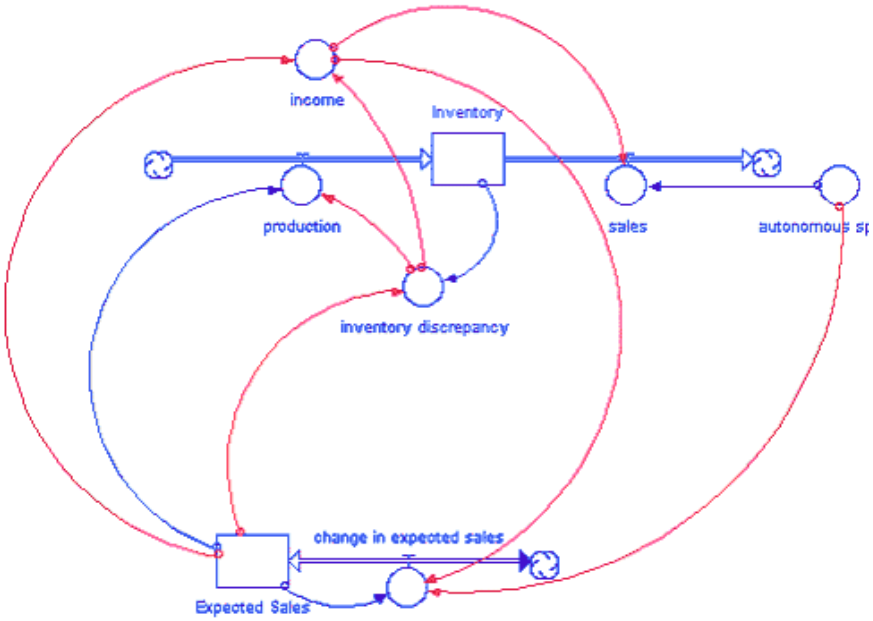


(a)

(b)



(c)



(d)

Fig. 11 System Dynamic Models - (A) First-Order Inventory Model, (B) Second-Order Inventory Model, (C) Salesmen Model, (D) Business Model

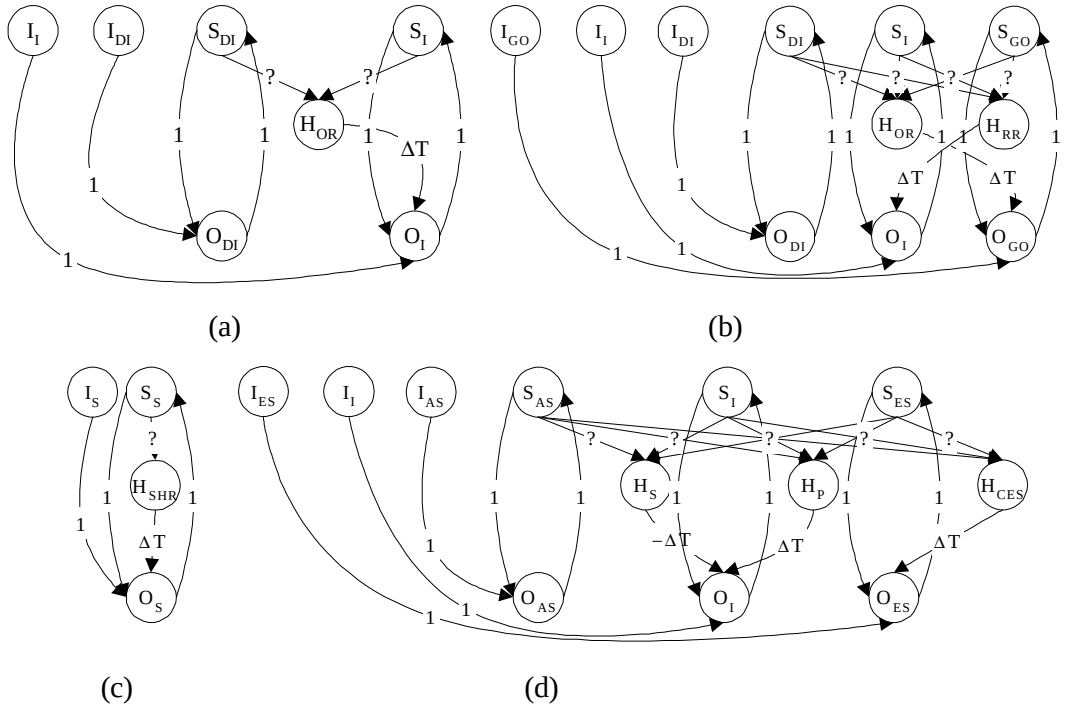


Fig. 12 four models of Fig. 11 in PRN representation

PRNs that do not satisfy the above conditions may or may not be mapped to a FD by a different rule or algorithm. Theory of this part is not clear yet and will be left in future research.

4. A Simple Application of the PRN Representation

We want to investigate the learning capability of the new representation with an attempt to explore the possibility of using it to assist model construction in SD. Therefore additional experiments are conducted. In particular, we are interested to see whether the PRN representation can learn by itself from the given exemplar data. Can a PRN in Section 3.3 re-create its own structure from the given data that generated by STELLA without the knowledge of what the model is? In another word, using the four models in the above section with given exemplar data prepared, will the new representation help us to re-discover the four models in PRNs?

In order to verify this, one needs first to prepare an initial PRN for each model that satisfy the conditions specified in Section 3.4. These are shown in Fig. 12, where there is one layer of hidden units that fully connected to the output units since one has no idea of the cause-effect relationships in the models. The numbers of links to be learned in each model are 2, 6, and 1, which are marked with "?" labels in the diagrams. The training data are collected from the simulation results of each model implemented in STELLA.

Tables 3–5 show the learning process of the first three models with a learning rate of 0.1. One can see that their learning results are almost perfect, with the weights to be adjusted gradually approaching the final target values and the weights on the nonexistent links all being reduced to near zero. The training of the three models completes at around the 150th, 200th, and 10th epoch, respectively. Note that an epoch means that the neural network was trained on the entire set of training examples once and each training example being used once in the learning process.

The data shown in Tables 6 and 7 are for training the network of model 4 with learning rates of 0.1 and 0.05, respectively. The learning effect is a little worse than for the first three models, but it is already very close to the target, and the three link pairs[3] with net effects are also successfully learned in the structure. When the learning rate is 0.1, the training process ends at 10,000 epochs; for a learning rate of 0.05, it ends at 15,000 epochs.

Although the above experiments are simple, it does show that various kind of SDMs can be learned, irrespective of whether a model contains positive (models 3 and 4) or negative (models 1, 2, and 4) feedback loops, is of high (models 2 and 4) or low (models 1 and 3) order, and is complicated (model 4) or simple (models 1, 2, and 3). The experiment also shows that the learning speed is dependent on the complexity of a model, which may be an obstacle when a model is too large. However, with these experiments, it does open the possibility of using the PRN representation to assist in the construction of a SDM, and its evaluation needs more studies.

Table 3. The learning process for model 1 with a learning rate of 0.1

Epochs / Links	1	50	100	150	In FD
DI->OR	0.07201	0.19778	0.19992	0.2	0.2
I->OR	–0.09258	–0.19779	–0.19992	–0.2	–0.2
SSE*	1.302581548690	0.000032366260	0.000000042438	0.000000000063	
*SSE = Sum of Square Error					

Table 4. The learning process for model 2 with a learning rate of 0.1

Epochs / Links	1	50	100	200	In FD
DI->OR	0.11011	0.21374	0.20021	0.2	0.2
GO->OR	–0.07076	–0.00995	0.00013	0	0
I->OR	–0.10091	–0.21682	–0.20033	–0.2	–0.2
DI->RR	0.11052	–0.01960	–0.00025	0	0
GO->RR	0.08630	0.11929	0.09987	0.1	0.1
I->RR	–0.10843	0.02385	0.00043	0	0
SSE	4.030906200408	0.083232857286	0.000025431156	0.000000000038	

[3] A set of links connected from the same state unit through different hidden units to the same output unit.

Table 5. The learning process for model 3 with a learning rate of 0.1

Epochs / Links	1	3	5	10	In FD
S->SHR	0.01248	0.01986	0.01998	0.02	0.02
SSE	0.107980273663	0.000050484439	0.000001129671	0.000000000084	

Table 6. The learning process for model 4 with a learning rate of 0.1

	Epochs / Links	1	1000	5000	10000	Values in FD
1	ES->ESC	–0.10942	–0.14061	–0.09534	–0.09688	–0.09685
2	I->ESC	0.21019	–0.27144	–0.20974	–0.21254	–0.2125
3	AS->ESC	0.28022	1.39048	0.98466	1.00011	1
4	ES->P	0.61841	0.66777	0.62893	0.62969	1.0625
5	I->P	0.43406	0.58526	0.63217	0.63128	–0.25
6	AS->P	0.62896	0.07293	0.25377	0.25072	0
7	ES->S	0.48159	0.43222	0.47107	0.47030	0.903124
8	I->S	0.86594	0.71474	0.66783	0.66873	–0.2125
9	AS->S	0.87104	1.42707	1.24623	1.25072	1
4–7		0.13682	0.23555	0.15786	0.15939	0.159376
5–8		–0.43188	–0.12948	–0.03566	–0.03745	–0.0375
6–9		–0.24208	–1.35414	–0.99246	–1	–1
SSE		2.134996175765	0.019681053236	0.000089574292	0.000000019754	

Table 7. The learning process for model 4 with a learning rate of 0.05

	Epochs / Links	1	5000	10000	15000	Values in FD
1	ES->ESC	–0.09791	–0.09770	–0.09685	–0.09687	–0.09685
2	I->ESC	0.22023	–0.21234	–0.21251	–0.21250	–0.2125
3	AS->ESC	0.28055	1.00563	0.99981	1.00000	1
4	ES->P	0.62255	0.63175	0.62964	0.62971	1.0625
5	I->P	0.43555	0.63359	0.63118	0.63124	–0.25
6	AS->P	0.62887	0.23292	0.25034	0.24961	0
7	ES->S	0.47745	0.46825	0.47037	0.47035	0.903125
8	I->S	0.86445	0.66642	0.66883	0.66877	–0.2125
9	AS->S	0.87113	1.26696	1.24938	1.24945	1
4–7		0.1451	0.1635	0.15927	0.15936	0.159376
5–8		–0.4289	–0.03283	–0.03765	–0.03753	–0.0375
6–9		–0.24226	–1.03404	–0.99904	–0.99984	–1
SSE		3.454487562179	0.000304681831	0.000000247206	0.000000007560	

5. Discussion

Now is the time to have some discussions on the PRN representation and compare it with traditional representations for a SDM. As one has seen, an obvious advantage of the new representation is its learning capability in the usage of assisting model construction. Being learnable, it can also make a created model adaptive to its surround outside environment. In particular, it can be applied to policy design too since a PRN can learn its structures from a set of prepared data. The idea is the same. By assigning an expected output pattern as a new set of training data to a given model, one can obtain a new structure after the training process;

the difference between the original and the new structure is the place for policy design to be considered. Similarly, by re-starting the learning process on and off after some period of using a model, one can fine-tune it to adapt to the environmental changes over time.

The other characteristic of the PRN representation is that it can be treated simply as a directed graph consisting of two symbols, i.e. nodes and arcs. In this aspect, it is more close to a CLD than a FD. The advantage of viewing this way is that many algorithms that apply to a graph can be applied to it. For example, it is easy to enumerate all loops within a model simply by a graph traversal algorithm; when the node equation used in a PRN is given, it is also easy to determine whether a loop is positive or negative from the weights associated with links. If there is an even number of weights in the same sign (positively correlated), it is a positive loop; otherwise, it is negative. This is different from a FD in which a loop is positive or negative depends on both the equation and the diagram.

There are of course weaknesses for the PRN representation. First, it is not suitable for conceptualization and communication because it lacks of meaningful semantic symbols to represent a model. It is better used in companion with a FD and servers as the backend engine for it. In this way, we can combine the benefits of both model representations. With a good tool support that implements the mapping between a FD and a PRN, this can be done transparently. Thus, it enhances the capability of a traditional FD. Second, some special components found in commercial SD modeling tools (e.g. STELLA and i-THINK) do not exist in the current PRN representation, e.g. “conveyor” and “queue”. These components however are optional and do not support in a normal FD, either. Third, the functions in a PRN are restricted to be linear, e.g., a weighted sum form. As describe in Section 3.2.4, a rate equation can be any arbitrary function. So it suffers a limitation if it has to be faithfully re-implemented in its original form in a PRN. Those functions (e.g., a look-up table) whose derivative value cannot be calculated will not participate in the learning process. Lastly and most importantly, to take the advantages of the new representation, there needs to be a set of well-prepared exemplar data. This is usually not available.

In summary, the new representation provides an enhancement to the traditional FD representation. By easily switching between the two model representations, people will have an additional choice of using which model in problem solving, and it is shown some problems, e.g. model construction, may now be solved easier in a new representation. This opens a new dimension for future SD research.

In the last section, we have illustrated the possibility of using the PRN representation to construct a SDM by inductive learning. Although it is a simple example that does not fully show the effectiveness of new model construction approach, it indeed shows the potential of it. A traditional approach of model construction is a *deduction* process performed based on human experts’ observation and intelligence. The difficulty is usually in that the target to be modeled is a dynamically complicated system, which may exist no observable guidelines or

hints for assistance during the construction process. Thus, the constructor's insight and experiences determines the quality of the created model.

The PRN representation, in contrast, suggests a new approach for thinking, i.e., *induction based on evidence*. The approach relies on well-established artificial intelligence algorithms to systematically search a problem space and check out every possibility of cause-effect relationships in order to identify the most appropriate structure for a model. Although it may lack of insight that a human being has initially, the automatic method will eventually generate a satisfactory model through a systematic search, as is shown in the above experiments. Furthermore, the automatic approach is not competitive but complimentary with the traditional ones. An experienced model constructor may use it for review or evaluate the structure generated instead of creating by himself/herself.

The future research will investigate the generality and scalability of the model construction approach in more detail and further study the feasibility of other applications, e.g. policy design and model adaptation.

Reference

Aussem, A. "Dynamical Recurrent Neural Networks towards Prediction and Modeling of Dynamical Systems," *Neurocomputing* **(28)**, 1999, pp. 207-232.

Burns, J. R. "Converting Signed Digraphs to Forrester Schematics and Converting Forrester Schematics to Differential Equations," *IEEE Transactions on Systems, Man, and Cybernetics* **(7:10)**, 1977, pp. 695-707.

Burns, J. R., Ulgen, O. M. and Beights, H. W. "An Algorithm for Converting Signed Digraphs to Forrester Schematics," *IEEE Transactions on System, Man, and Cybernetics* **(9:3)**, 1979, pp. 115-124.

Coyle, R. G. *Management System Dynamics*, Wiley, Chichester, 1977.

Dolado, J. J. "Qualitative Simulation and System Dynamics," *System Dynamics Reviews* **(8:1)**, 1992, pp. 55-81.

Elman, J. L. "Finding Structure in Time," *Cognitive Science* **(14)**, 1990, pp. 179-211.

Forrester, J. W. *Industrial Dynamics*, MIT Press, Cambridge, MA, 1961.

Forrester, J. W. "Industrial Dynamics-After the First Decade," *Management Science* **(14:7)**, 1968a, pp. 393-415.

Forrester, J. W. *Principles of Systems*, MIT Press, Cambridge, MA, 1968b.

Forrester, J. W. "Market Growth as Influenced by Capital Investment," *Sloan Management Review* **(9)**, 1968c, pp. 83-105.

Goodman, M. R. *Study Notes in System Dynamics*, MIT Press, Cambridge, MA, 1974.

Jordan, M. I. "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*,

Hillsdale, NJ, 1986, pp. 531-546.

Lane, D. C. “Diagramming Conventions in System Dynamics,” *Journal of the Operational Research Society* **(51)**, 2000, pp.241-245.

Morecroft, J. D. W. “A Critical Review of Diagramming Tools for Conceptualizing Feedback System Dynamics Models,” *Dynamica* **(8:1)**, 1982, pp. 20-29.

Morecroft, J. D. W. “Rationality in the Analysis of Behavioral Simulation Models,” *Management Science* **(31:7)**, 1985, pp. 900-916.

Nie, J. “Nonlinear Time-series Forecasting: A Fuzzy-neural Approach,” *Neurocomputing* **(16)**, 1997, pp. 63-76.

Ramos, J. M. “Some Modifications to the Burns Algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics* **(13:1)**, 1983, pp.108-110.

Randers, J. “Guidelines for Model Conceptualization,” in *Elements of the system dynamics method* Randers, J. (eds.), MIT Press, Cambridge, MA, 1980, pp. 117-139.

Richardson, G. P. and Pugh, A. L. III *Introduction to System Dynamics Modeling with DYNAMO*, Productivity Press, Cambridge, MA, 1981.

Richardson, G. P. “Problems with Causal-loop Diagrams,” *System Dynamics Review* **(2:2)**, 1986, pp. 158-170.

Richardson, G. P. “System Dynamics: Simulation for Policy Analysis from a Feedback Perspective” in *Qualitative Simulation Modeling and Analysis*, Fishwick P.A. and Luker P. A. (eds.), Springer-Verlag, New York, 1991, pp. 144-169.

Richmond, B., Peterson, S. and Vescuso, P. *An Academic User’s Guide STELLA*, High Performance System, Inc., Lyme, NH, 1987.

Roberts, N. Andersen, D. F., Deal, R., Garet, M. and Shaffer, W. *Introduction to Computer Simulation: The System Dynamics Modeling Approach*, Addison-Wesley, Reading, MA, 1983.

Senge, P. M. The Fifth Discipline: The Art and Practice of the Learning Organization, Doubleday/Currency, New York, 1990.

Starr, P. J. “Modeling Issues and Decisions in System Dynamics,” *TIMS Studies in the Management Science* **(14)**, 1980, pp. 45-59.

Zhang, G. and Hu, M. Y. “Neural Network Forecasting of the British Pound/US Dollar Exchange Rate,” *Omega* **(26:4)**, 1998, pp. 495-506.

Back to the Top