System Neural Network: Evolution and Change Based Structure Learning

Article *in* IEEE Transactions on Artificial Intelligence · June 2022

DOI: 10.1109/TAI.2022.3143778

CITATIONS
7

READS
83

4 authors, including:

Animesh Chaturvedi
Indian Institute of Information Technology Dharwad
25 PUBLICATIONS **148** CITATIONS
SEE PROFILE

Shubhangi Chaturvedi
Saffrony Institute of Technology
5 PUBLICATIONS **29** CITATIONS
SEE PROFILE

System Neural Network: Evolution and Change based Structure Learning

Animesh Chaturvedi, Aruna Tiwari, Shubhangi Chaturvedi, and Pietro Liò

***Abstract*— System evolution analytics with artificial neural networks is a challenging and path-breaking direction, which could ease intelligent processes for systems that evolve over time. In this paper, we contribute an approach to do *Evolution and Change Learning* (ECL), which uses an *evolution representor* and forms a *System Neural Network* (SysNN). We proposed an algorithm *System Structure Learning* (SSL), which is divided in two steps. First step uses the evolution representor as an evolving matrix *Evolving Design Structure Matrix* (EDSM) for intelligent design learning. Second step uses a *Deep Evolution Learner* (DEL) that learns from evolution and changes patterns of an EDSM to generate Deep SysNN. The result demonstrates application of the proposed approach to analyze four real-world system domains: software, natural-language, retail market, and movie genre. We achieved significant learning over highly imbalanced datasets. The learning from previous states formed SysNN as a feed-forward neural network, and then memorized information as an output matrix that has recommendations for entity-connections.**

Impact Statement — It is useful and challenging to apply machine learning to analyze the multiple states of an evolving system. To do this, we represented existing states of an evolving system as EDSM that is used to generate SysNN. The SysNN generates an output matrix that helps to analyze and make recommendations about that evolving system. We present experiments to study imbalanced data of evolving systems.

***Index Terms*— Systems Engineering and Theory, Artificial Neural Networks, Machine Learning, and Graph theory.**

I. Introduction

SYSTEM has several entities (or component) that are inter-connected to each other. Some systems evolve with time contains evolving entity-connections; such a system is referred as an evolving system. Such an *evolving system* generates time-variant (or non-stationary) data. Bertalanfy [1] "growth of components within a system… applies to many growth phenomena in biology (evolution)". Growth of evolving systems both in numbers of components (or entities) and in the size of an entity – makes it challenging to learn a system.

We are considering a system with the following two properties. First, the system contains entities (or features) interacting with each other to make connections between entities. These entities and connections can be represented as a matrix. Second, the system evolves over time, which makes states that can be represented as a series of matrices.

System network structure example 1, in Fig. 1, assumes software contains procedures (or functions) as entities (or nodes) that make procedure-calls as connections (or edges), which makes a call-graph G_1. The call-graph G_1 evolves to G_2 then to G_3 for three software version series. System network structure example 2, in Fig. 1, assumes three graphs (G_1, G_2, G_3) represent three phrases of a natural-language paragraph such that vertices represent keywords and edges represent keyword-connections. This makes keyword-network means in a sentence source-keyword A appears before target-keyword B.

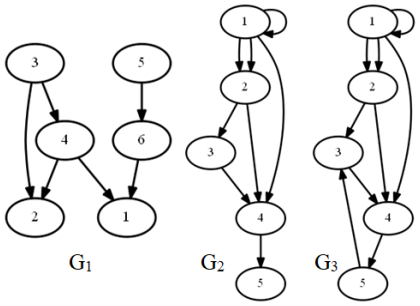


Fig 1. Evolution of a system graph from G_1 to G_2 to G_3 when a system evolves from state S_1 to S_2 to S_3.

Structure, designing, or representation of a system can support system analysis. We can represent a system state with a *Design Structure Matrix* (*DSM*) [2][3], which is a simple, compact, and visual representation of a system (or project) in the form of a square matrix. The DSM has various purposes with multiple names like: Dependency source matrix, Problem Solving Matrix (PSM), and Design precedence matrix. Our approach is based on repository or database learning of batch off-line training applied over system states represented as DSM.

We use Artificial Neural Network (ANN) recommendation power by using three famous unsupervised deep learning techniques [4]: Restricted Boltzmann Machines (RBM) [5], Deep Belief Networks (DBN) [6], and denoising Autoencoders (dA) [7]. We have chosen them to express and apply *system evolution analytics* [9][10][11], which is done by learning useful knowledge representation from time-variant (or non-stationary [8]) data of evolving states.

In general, ANN and DNN will have better predictive capabilities due to complex non-linear optimisation (or activation) functions (e.g. sigmoid, ReLU, Gaussian, RBM, encoders etc.), which aims to minimise the recommendation error. Whereas, other techniques deal with simple equations e.g. the linear-regression has linear equations f(x) = ax +b, the rule-mining measures frequency and conditional probability, and so on. However, there are other complex techniques like ensemble

Animesh Chaturvedi is with the Indian Institute of Information Technology Dharwad, Dharwad Karnataka. Email: animesh.chaturvedi88@gmail.com.

Aruna Tiwari is with the Indian Institute of Technology Indore, Indore, MP, India. E-mail: artiwari@iiti.ac.in.

Shubhangi Chaturvedi is with the Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, Jabalpur, MP, India. E-mail: chaturvedishubhangi51@gmail.com.

Pietro Liò is with the University of Cambridge, Cambridge, U. K. Email: Pietro.Lio@cl.cam.ac.uk

learning, active learning etc., which have their pros and cons.

Motivation is to perform system learning to learn evolving system states over time, which can assist system evolution analysis. We do system learning on inter-connected entities represented in DSMs of system state series with evolution information (or an evolving repository). To do system learning, we are extending the system evolution recommender theory [12][13], which made following two extensions in two sections:

- In Section II, we formalized Evolution and Change Learning (ECL) to form our proposed System Neural Network (SysNN), which is a new kind of ANN.

- In Section III, we proposed an algorithm to do *System Structure Learning* (SSL) based on *deep evolution learner* on time-variant data of an evolving system. Firstly, we represented an evolving system in the form of an evolving matrix, which captures multiple system states as an Evolving DSM. Secondly, we expressed system learning to learn evolution and change information from the Evolving DSM. This learning forms a Deep SysNN, which helps to construct an output matrix (as a *disk memory*) that predicts the evolving system.

These two contributory Sections (II and III) are followed by Section IV that presents evaluation of the SSL using two cross validations: k-Fold and forward-chaining, which helps to empirically evaluate our algorithm. Section V demonstrates the applications and experiments on six real-world evolving systems of four domains. Thereafter, related works (in Section VI) and concluding remarks (in Section VII).

II. Evolution and Change Learning Based System Neural Network

This section describes our key idea to do Evolution and Change Learning (ECL) that forms a *System Neural Network* (SysNN). Suppose each state (S_i) of the state series has a data (D_i). This makes a time-variant dataset TVD = {D_1, D_2 … D_N, D_{N+1}} of a state series SS = {S_1, S_2 ... S_N, S_{N+1}} at (N+1) time points {t_1, t_2 … t_N, t_{N+1}}. Fundamentally, three types of changes are possible: addition (or insertion), modification (or alteration), and deletion (or removal), which need to be represented for learning.

The proposed approach ECL has two steps to generate a SysNN. In the first step, the state series of an evolving system is represented into an evolution representor. To intelligently manage a system (or project), we denote an evolution representor as **ER** for a state series of an evolving system. In the second step, the algorithm uses this hidden vector **ER** to calculate an output vector **Y**. We describe this in the context of system learning of time-variant or non-stationary data.

Definition 1: *Evolution and Change Learning* (ECL) is defined as a machine learning that learns evolution and changes from an *Evolution Representor* **ER** that represents previous states in a state series SS. The **ER** is determined as follows.

- The **ER** represents time-variant data TVD in a SS by a function f(**SS**) given as equation (1)

$$\mathbf{ER} = f(\mathbf{SS}) \quad \ldots (1)$$

$\mathbf{ER} = f(\{a_{11}.. a_{jk} \ldots a_{mm}\}, \{b_{11}.. b_{jk} ... b_{mm}\} \ldots \{x_{11}.. x_{jk} \ldots x_{mm}\})$.

- The calculation is done by a hidden function g(**ER**), named as *Evolution Learner* in equation (2). The purpose of the function g(**ER**) is to learn evolution and changes happening to the system entities.

$$\mathbf{Y} = g(\mathbf{ER}) = g(f(\mathbf{SS})) \quad \ldots (2)$$

$\mathbf{Y} = (\{y_{11}.. y_{jk} \ldots y_{1m}\}, \{y_{21}.. y_{jk} \ldots y_{2m}\} \ldots \{y_{m1}.. y_{jk} \ldots y_{mm}\})$.

The **ER** are 2-dimensional tensor or array mentioning (m×m) × N connectivities. The tensor contains *entity-connections*, a_{jk}, b_{jk}… x_{jk} is 1 or 0, the 'jk' represents between two entities 'j' and 'k'. The a, b, x denotes input states and y denotes output. Series of 'a', 'b' and 'x' represent 1st, 2nd and Nth states. In state 'a', if there exists a connection between j^{th} and k^{th} entities, then a_{jk} is 1 otherwise 0, similarly for b_{jk}… x_{jk}. Here, m is the total number of distinct entities in all N+1 states. The **Y** is a desired *output_vector* that has a variable y_{jk} whose value is in between 0 to 1. The y_{jk} gives probability in range $1 \leq y_{jk} \leq 0$ for existence of a connection between j^{th} and k^{th} entities, and $y_{jk} = 0$ means *no-connection* between j^{th} and k^{th} entities. This **ER** constructed from a state series is used for learning evolution information.

The ECL learns from **ER** make a computer capable enough to understand the evolution and changes of a system without any explicit programming. While learning, the input layer depends on the size of elements in the evolution representor **ER**, a user provides the number of hidden layers L, and the output layer depends on the size of the output vector. Initialize these three parameters for evolution and change learning, which forms a neural network defined as follows.

Definition 2: *System Neural Network* (SysNN) is a feed-forward *artificial neural network*, which contains information and understanding of system structure by learning evolution represented in an evolution representor **ER** of a state series SS. The SysNN contains evolution information in the form of adjustment between weights and neurons such that it makes a matrix, which gives probable connections (occurrences) of two entities (features) together in a system state. The ECL forms a SysNN, which recommends system evolution using memorized output vector **Y**.

A machine with SysNN can recommend the time-variant (or non-stationary) data of an evolving system. The SysNN learns evolution information that constructs an output matrix (in a *disk memory* as a trained model). The evolution information in the memory can be useful to forecast the possible future of the system data. It is perceivable that the SysNN is a unique and novel kind of cybernetics.

The SysNN learns system evolution information by adjusting weights between its neurons. The neuron weight adjustments are according to the probability for existence of connections between system entities. The SysNN reconstruct a *zero vector* into an *output vector*, both containing m×m elements. Convert the *output vector* into *output matrix* M_O of size m×m such that this M_O is an ECL information. The M_O is a disk memory that stores information about the evolving states of a system. This memory will help to make recommendations about the evolving system.

Next section describes the algorithmic form of the proposed approach to realize our approach for ECL that makes a SysNN.

III. System Structure Learning

This section describes contributory algorithm *System Structure Learning* (SSL). It uses N Design Structure Matrices (*N_DSMs*) to represent N states of an evolving system. Fig. 2 gives an overview of SSL that internally uses Evolving DSM

and Deep Evolution Learner (DEL). The DEL uses evolution representor to learn the evolution of a time-variant data of a state series. The DEL reconstructs an input zero vector to an output vector, which is transformed to an output matrix that is further normalized. First, we define the Evolving DSM (*EDSM*) as shown in the left-top of Fig. 2.

Definition 3: *Evolving Design Structure Matrix* (EDSM) is a type of DSM that represents an ordered series of N vectors for time-variant data of N states in a state series SS of an evolving system (or project). EDSM intelligently designs a state series to represent evolution as the 2-dimensional tensor of (m×m) × N containing *entity-connections* (ec_{jk}) between entity 'j' and 'k'.

Algorithm ***SSL***(*N_DSMs*)

Initialize a zero matrix as M_Z
zero_vector = ***matrixToRowVector***(M_Z)
EDSM = ***evolutionRepresentor***(*N_DSMs*)
output_vector = ***deep_evolution_learner*** (*EDSM*, *zero_vector*)
M_O = ***rowVectorToMatrix***(*output_vector*)
M_{NO} = ***normalize***(M_O)
Return M_{NO}

Next, we describe details about the Algorithm SSL. We elaborate the time-variant data of ECL, which works as input for unsupervised system learning. The ***evolutionRepresentor*** transforms N DSMs (*N_DSMs*) into N vectors, which represents N training states in an EDSM. The algorithm transforms a DSM of state S_i into *vector_i*. For a state, its DSM has size m×m and each vector (*vector_i*) has m×m elements. Then, each *vector_i* combines to form an EDSM. The EDSM given in equation (3) is a type of evolution representor **ER** as described in the equation (1).

$$\text{EDSM} = f(\mathbf{SS})$$
$$\text{EDSM} = \boldsymbol{evolutionRepresentor}(N_{DSMs}) \quad \dots (3)$$

Algorithm ***evolutionRepresentor***(*N_DSMs*)

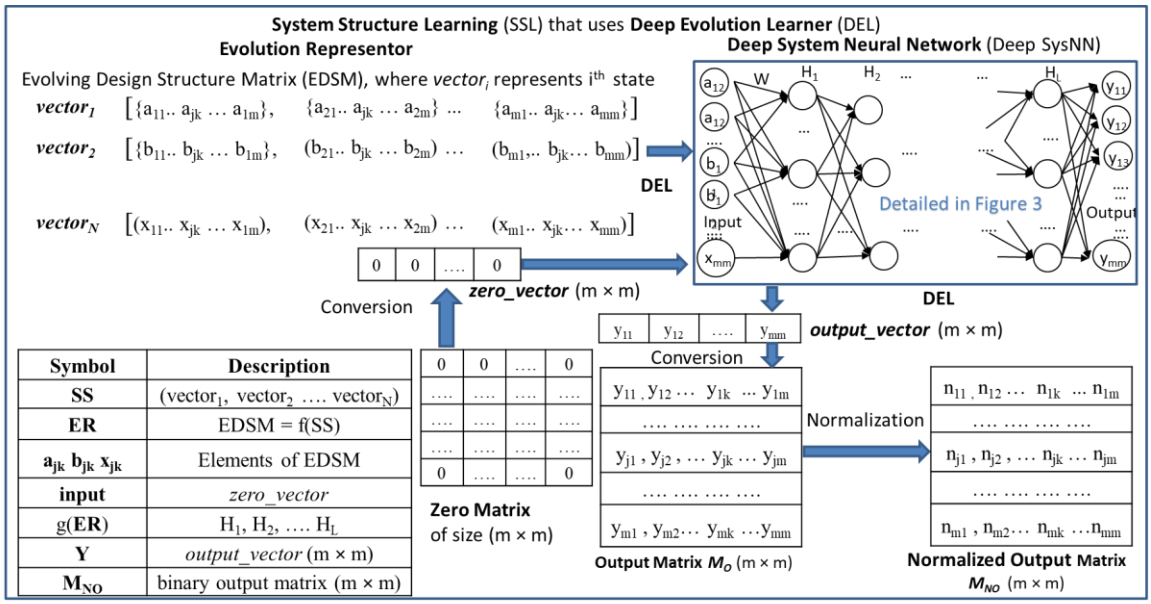
For each *DSM_i* in *N_DSMs* where i ∈ state number
vector_i = ***matrixToRowVector***(*DSM_i*)

$$EDSM = \left\{ \begin{array}{c} vector\_i \\ + \\ EDSM \end{array} \right\}$$

End for
Return *EDSM*

Fig. 2 shows a graphical schematization of the relationships between SSL, DEL, EDSM, ECL, and SysNN. The ***SSL*** uses DEL (***deep_evolution_learner***) that elaborates the evolution learner g(**ER**) mentioned as equation (2). An unsupervised learning on the EDSM (unlabeled data) can detect evolution and change patterns. The SSL depends on the EDSM and the DEL (unsupervised learning). The DEL learns evolution and change patterns in the EDSM (time-variant data) of a state series. The DEL makes a machine capable enough to understand the evolution and changes happening between states without explicit programming. The DEL is an extension of deep learning that learns from an EDSM and outputs a vector (*output_vector*). The DEL makes the machine intelligent by generating *Deep System Neural Network* (Deep SysNN) shown in Figs. 2 and 3.

In Figs 2 and 3, the ***deep_evolution_learner*** takes the EDSM of size N × (m×m) for training purposes, where N is the number of states and m × m is the size of square DSM. The DEL makes a Deep SysNN (in the form of weight matrices) based on the training by ***delTrain*** (means deep evolution learner training). The three main training parameters (*trainParameters*) control deep learning: learning rate (LR), epoch (Ep), and number of hidden layers (L). Deep learning can use matrices, but the DEL uses evolution representor (i.e., EDSM), which is the crux of



Here, the 'jk' represents connection between two entities 'j' and 'k'.
The a, b, x denotes input states, y denotes outputs, and n denotes normalized values.
Series of 'a', 'b' and 'x' represent 1st, 2nd and Nth state. The L is the number of hidden layers H with weight set W.
The a_{jk}, b_{jk}, x_{jk}, n_{jk} are either 1 or 0 depending upon connection exist or not. The y_{jk} is the range between 0 to 1 such that $0 \le y_{jk} \le 1$
The EDSM size is (m×m) × N, where the m is the total number of entities in all the (N+1) states and N is number of training states .

Fig 2. An overview of Algorithm SSL using DEL on EDSM as a time-variant data (or non-stationary data) consisting of the N data vector of a state series to construct a Deep SysNN, whose three variants are explained in Fig. 3.

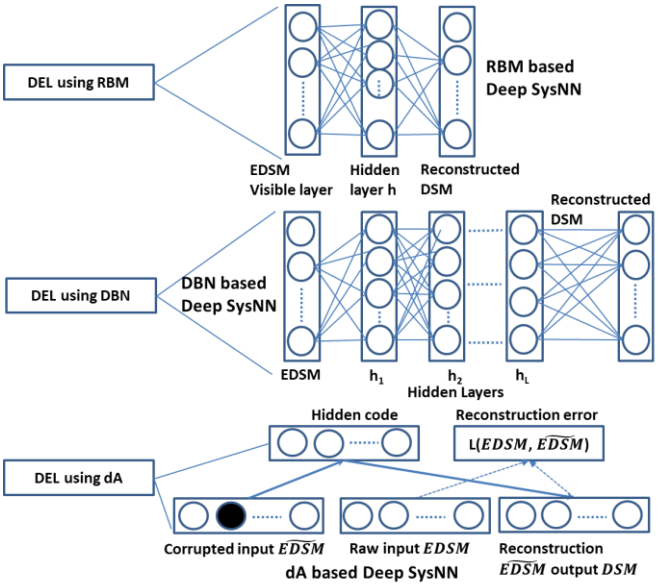


Fig 3. Three kinds of Deep SysNN constructed by three DEL variants based on the: RBM (in top), DBN (in middle), and dA (in bottom).

our approach. The Deep SysNN consists of information about patterns of entity-connections in the form of a matrix of neural networks. The training time mainly depends upon the number of states and number entities i.e. N×m×m. The ***delReconstruct*** uses Deep SysNN to reconstruct a *zero_vector* to an *output_vector* with m×m elements.

Algorithm ***deep_evolution_learner*** **(***EDSM*, *zero_vector***)**

Initialize List *trainParameters*< LR, Ep, L >
Initialize a matrix *Deep_SysNN*
Deep_SysNN = ***delTrain*****(***EDSM*, *trainParameters***)**
// three variant of Deep SysNN based on Equation 4 to 10
output_vector = ***delReconstruct*****(***Deep_SysNN*, *zero_vector***)**
Return *output_vector*

The computational complexity of the SSL algorithm depends on the ***deep_evolution_learner*** algorithms, which further depends upon the ***delTrain*** and ***delReconstruct*** approach. Both are optimization problems (not decision problems) for the purpose of probabilistic solutions to do recommendations or predictions. Our technique uses optimization functions to provide an approximate solution based on probabilistic modeling in polynomial runtime $O(N \times m^2)$, where N is the number of states and m is the number of entities. In Fig. 3, we describe how ***delTrain*** (hidden layer or code) and ***delReconstruct*** works in the DEL using extended objective functions of three well-known deep learning techniques: RBM, DBN, and dA. RBM and DBN are used because they are fundamental techniques of Deep learning.

First, the DEL reformulates the Restricted Boltzmann Machine's energy model as equation (4) with $E(\boldsymbol{EDSM}, \boldsymbol{h}) =$

$$\sum_i (a_i\, ec_{jk}) + \sum_l (b_l\, h_l) + \sum_i \sum_l (ec_{jk}\, w_{il}\, h_l) \quad \ldots (4)$$

where i represent i^{th} state, the $\boldsymbol{EDSM}$ represents an evolution representor that contains ec_{jk} as an entity-connection between two entities j *and* k. In the *Deep_SysNN* matrix of size (m×m) × L, this w_{il} represents the weight of i and l position. The weight matrix represents connections in SysNN such that a DSM is a visible unit of size m×m and the $\boldsymbol{h}$ is hidden unit. The RBM equation, there are two bias weights (offsets): the a_i and b_l for visible and hidden units respectively. Reformulate the energy model $E(\boldsymbol{EDSM}, \boldsymbol{h})$ into a probabilistic model makes equation

$$P(\boldsymbol{EDSM}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{EDSM},\ \boldsymbol{h})} \quad \ldots (5).$$

Then find conditional probability h of hidden layer when *EDSM* is given as input, which makes equation

$$\mathrm{g}(\mathbf{ER}) = P(\boldsymbol{h}|\boldsymbol{EDSM}) = \prod_{l=1}^{L} P(\boldsymbol{h_l}|\boldsymbol{EDSM}) \quad \ldots (6).$$

For best recommendation by equation (6), the reconstruction error needs to be minimized objective function as equation

$$\frac{d log P(\boldsymbol{h}|\boldsymbol{EDSM})}{d\boldsymbol{W}_{jkl}} \approx < ec_{jk} h_l >_{data} - < ec_{jk} h_l >_{reconstruct} \quad \ldots (7).$$

Second, the DEL reformulates the Deep Belief Network as a stack of RBMs working together for training on EDSM. This greedy learning forms the Deep Belief Network. Therefore, the reformulation makes equation (8) with $\mathrm{g}(\mathbf{ER}) =$

$$P(\boldsymbol{EDSM}, h^1, h^2 \ldots h^L) = \left(\prod_{l=0}^{L-2} P(h^l | h^{l+1}) \right) P(h^{L-1}, h^L) \quad \ldots (8)$$

where L denotes a user-defined number of hidden layers and h^l denotes the l^{th} hidden layer.

Third, the DEL reformulates the auto-encoder to make a neural network by *encoding* the input $\boldsymbol{EDSM}$ into $c(\boldsymbol{EDSM})$. The objective is to reconstruct $c(\boldsymbol{EDSM})$ using a *decoder* function with least error. The DEL minimizes reconstruction error by minimizing the objective function of negative log-likelihood of conditional entropy given as equation (9)

$$RE = -\log P\big(\boldsymbol{EDSM} \big| c(\boldsymbol{EDSM})\big) \quad \ldots (9)$$

$$RE = -\sum_i ec_{jk}\, log f_i\big(c(\boldsymbol{EDSM})\big) + \big(1 - ec_{jk}\big) \log\Big(1 - f_i\big(c(\boldsymbol{EDSM})\big)\Big)$$

where $f_i(x)$ decode x_i, and $f_i\big(c(\boldsymbol{EDSM})\big)$ reconstruct i^{th} state of $\boldsymbol{EDSM}$ using the Deep SysNN.

The input of denoising Autoencoder (dA) is stochastically corrupted, and uncorrupted input is used to reconstruct the target. The dA recommends the entity-connection patterns with reformulated equation to reconstruct negative log-likelihood

$$RE = -\log\big(\mathrm{g}(\mathbf{ER})\big) = -\log P\left(\boldsymbol{EDSM} \middle| c\big(\widetilde{\boldsymbol{EDSM}}\big)\right) \quad \ldots (10)$$

where $\boldsymbol{EDSM}$ is the uncorrupted input, $\widetilde{\boldsymbol{EDSM}}$ is the stochastically corrupted input, and $c(\widetilde{\boldsymbol{EDSM}})$ is the encoded form of $\widetilde{\boldsymbol{EDSM}}$. To obtain best recommendations, minimize the equation (10) as an objective function.

In SSL algorithm, the *output_vector* is converted to form a matrix M_O of size (m×m). The elements (y_{jk}) in M_O are between 0 and 1, which gives probability of connections between two entities. The M_O is normalized to a *normalized matrix output* M_{NO}. In normalization, the element (y_{jk}) in the M_O is converted to element n_{jk} of M_{NO} according to the normal distribution

threshold. Thus, M_{NO} is a binary matrix with n_{jk} is either 1 or 0 depending upon whether the connection exists or not. The M_{NO} is a kind of memorized information about an evolving system, which is useful for decision-making and taking action. This helps to deal with adaptation, control, and analysis of evolution and changes in a system over evolving states.

For example, In Table I, an evolving system named as "List of Multi-sport events" has 7 states for training and 1 state for testing. The Fig. 4 show 7 adjacency matrix, which makes 7 DSMs as input to the SSL algorithm. This will result in three different reconstructed matrices, which is compared with testing matrix. The results of comparisons are given in Table II.

IV. Evaluation Using K-Fold and Forward-Chaining

This section describes evaluation of our System Structure Learning (SSL) approach using two techniques for experimental cross validation: k-Fold and forward-chaining (also known as time series cross validation). Our model has four parts to test SSL algorithms: training data, testing data, target output, and classifier metrics. The **training data** of DEL is an EDSM. The DEL learns evolution patterns from EDSM of a state series in an 'unsupervised manner'. The training data is the connection patterns between a set of entities (features) in the form of EDSM for a state series. After training, the DEL creates Deep SysNN. Learning and recommendation of changes rely on frequency of occurrence of connection between two entities (features) in an EDSM. The Deep SysNN supports reconstruction of a *zero matrix* M_Z to an *output matrix* M_O.

The **testing data** is a testing matrix M_T, which represents entity-connections of a state for testing. The testing data is a DSM of a state that is unused during the training phase. The **target output** is the output matrix M_O that can reflect evolution information about the evolving system. Using Deep SysNN, we can automatically produce an *output matrix* M_O (containing patterns of entity-connections) that recommends evolving entity-connections of an evolving system.

There are four well-known **binary classifier metrics** (*accuracy*, *precision*, *recall*, and *f-measure*), which checks whether the output is correct or incorrect. The four metrics can analyze the output M_{NO} according to the desired correctness.

Next, we discuss two cross validation techniques using the four binary classifier metrics. The k-Fold is a well-known technique [14], thus we skip its basic explanation. The forward-chaining [15] is used for time (state) series based data where learning from earlier states has importance. To do the two types of cross validation, we made two types of directory (folder) that contains training and testing data.

1. *k-Fold*: We made a *k-Fold folder* containing N+1 subfolder, where each subfolder contains N training matrix and 1 testing matrix M_T_i of state S_i. The name of the subfolder is the testing state name (or number), and we denote it as 'i'. Here, k stands for the N states used for training purpose.

2. *forwardChaining*: We made a *forwardChaining folder* containing N subfolders. Each subfolder contains training matrices of *previous states* and 1 *testing matrix* M_T_i of current (i.e., testing) state S_i. The name of the subfolder is the current state name (or number), and we denoted it as 'i'. For example,

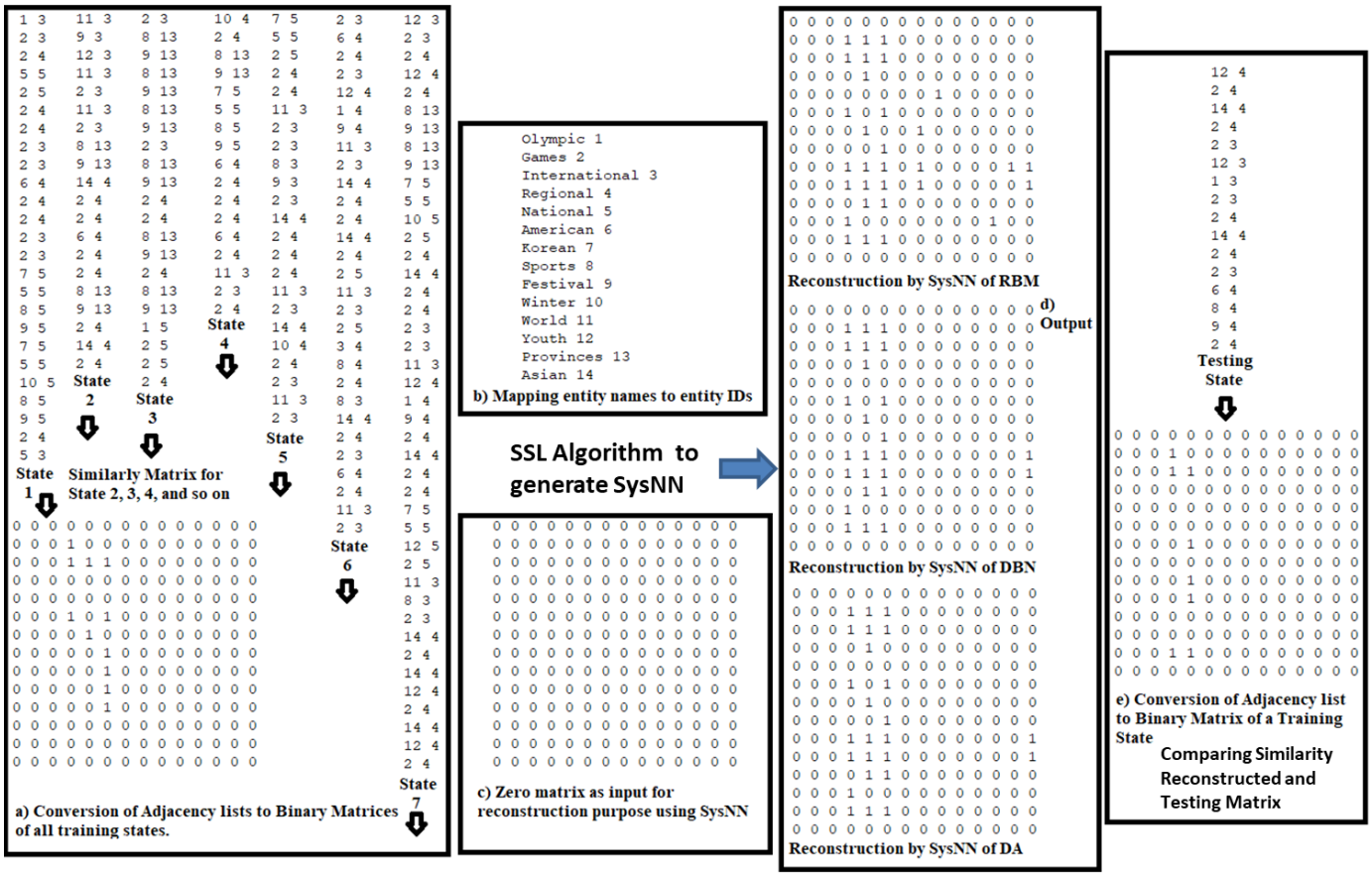


Fig 4. Illustrative example of Algorithm SSL using DEL on input: 7 learning states and a testing state. This output 3 reconstructed matrices.

Algorithm ***k-Fold_forwardChaining***( )

Initialize *accuracy*, *precision*, *recall*, *f-measure*, *test_metric*
For each i^{th} subfolder in *k-Fold* and *i* ∈ state number
  M_{NO} = ***SSL***(*Matrix*$_N$)
  accuracy = ***accuracy***(M_{NO}, M_T_i)
  precision = ***precision***(M_{NO}, M_T_i)
  recall = ***recall***(M_{NO}, M_T_i)
  f-measure = ***f-measure***(*precision*, *recall*)
  test_metric = ***line***(*i*, *accuracy*, *precision*, *f-measure*, *recall*)
End for
Calculate four averages as (*Acc.*, *Pre.*, *Fme.*, *Rec.*) of all four metrics as over the number of states in *test_metric*.
Then, calculate the average (Avg.) of the four averages.
For each i^{th} subfolder in *forwardChaining* and *i* ∈ state number
  M_{NO} = ***SSL***(*Matrix*$_{previous\_states}$)
  accuracy = ***accuracy***(M_{NO}, M_T_i)
  precision = ***precision***(M_{NO}, M_T_i)
  recall = ***recall***(M_{NO}, M_T_i)
  f-measure = ***f-measure***(*precision*, *recall*)
  test_metric = ***line***(*i*, *accuracy*, *precision*, *f-measure*, *recall*)
End for
Calculate four averages as (*Acc.*, *Pre.*, *Fme.*, *Rec.*) of all four metrics over the number of states in *test_metric*.
Then, calculate the average (Avg.) of the four averages.
Return *k-Fold*<*Acc.*, *Pre*., *FMe*., *Rec*., Avg.> and
forwardChaining<*Acc.*, *Pre*., *FMe*., *Rec*., Avg.>

suppose a training data varies from 1 to N states, where N = 4.

k-Fold folder where k = 4:	*forwardChaining folder*
folder 1: train [2 3 4] and test [1]	folder 1: train [1] and test [2]
folder 2: train [1 3 4] and test [2]	folder 2: train [1 2] and test [3]
folder 3: train [1 2 4] and test [3]	folder 3: train [1 2 3] and test [4]
folder 4: train [1 2 3] and test [4]	

The ***k-Fold_forwardChaining*** algorithm performs two types of cross validations: k-Fold and forward-chaining. The first and second 'for-loop' identifies the result for the *k-Fold folder* and the *forwardChaining folder* respectively. Both the loop calculates four binary classifier metrics over their folder, and then stores the calculated values in their *test_metric*. After each loop, the *test_metric* is used to calculate the four averages (Acc., Pre., FMe., Rec.) of accuracy, precision, f-measure, and recall over the number of subfolders in the *k-Fold folder* and *forwardChaining folder*. Then, for both the validations, these four averages are further averaged as (Avg.). All this will return five results to both: *k-Fold*<*Acc.*, *Pre*., *FMe*., *Rec*., Avg.> and *forwardChaining*<*Acc.*, *Pre*., *FMe*., *Rec*., Avg.>.

In the next section, we present analysis of six different evolving systems. Our approach helps to intelligently manage an evolving system represented as a state series of DSM.

V. Experiments of SSL and SysNN

This section demonstrates intelligent project management application of our approach on the six real-world evolving systems of four domains. We collected the six evolving systems from four kinds of repositories: software (Maven), natural language (Wikipedia), retail market (UCI), and movies genres (IMDb). Four applications of these domains include: software evolution analytics, natural-language evolution analytics, market evolution analytics, and movie evolution analytics, respectively (as mentioned in the first column of Table I).

Based on the proposed SSL approach, we developed and experimented with a prototype tool named as SysEvoRecomd-Tool to analyze time variant (or non-stationary) data of the six evolving systems (mentioned in second column). For each evolving system, we evaluate recommendations by SysEvoRecomd-Tool. For each experiment, there are N+1 states of an evolving system (as mentioned in the third column of Table I). Only N states are used in the training phase and the remaining one state is used for the testing phase. Each of the evolving systems is represented with a set of N+1 evolving graphs over time that is further converted to N+1 DSMs. To demonstrate applied intelligent management, we performed the following three steps on each evolving system.

- **First**, in the SSL, the ***evolutionRepresentor*** algorithm transforms N DSMs into N vectors such that each vector contains m×m elements. Then, the algorithm combines the N vectors to form an EDSM as training input. An EDSM represents the non-stationary data due to its property of varying with time. Each EDSM has size (N × (m×m)), given in the fourth column, where N is the number of states (e.g. software versions) and m is the number of entities (e.g. procedures in software).

- **Second**, in the SSL, the ***deep_evolution_learner*** (DEL) uses an EDSM to generate a Deep SysNN. The DEL variants have its own advantages to analyze the system evolution. The learning an evolving matrix (i.e. EDSM) over time is a non-stationary learning of a full system data. This learning depends upon the following variables: number of states (N), number of entities (m), and training parameters: Learning Rate (LR), Epoch (Ep), and number of hidden units/layers of neurons (L). The (LR-Ep-L) value used in each experiment is mentioned in the sixth column of Table I. These training parameter values resulted in high accuracy and f-measure. The DEL works like a feed-forward neural network learning algorithm, which uses many iterations (i.e. epochs) in the learning phase. Learning rate controls the learning of an algorithm. To keep it simple, for all the dA experiments data corruption rate used is 0.3, and for DBN the same learning rate for pre-training and for fine-tuning is used.

The DEL in the SysEvoRecomd-Tool extended deep learning (source code of RBM [5], DBN [6], and dA [7] written in java https://github.com/yusugomori/DeepLearning by Sugomori Yusuke [18]). We used these codes to make three variants of *deep evolution learner* (DEL). For each evolving system, the fifth column of Table I contains a group of three rows for three DEL variants: RBM, DBN, and dA.

We did many experiments to determine the best training parameters. Like other optimization techniques, while performing the experiments, we used Explore (search region for candidate solutions outside neighborhood) and Exploit (search best solution within the neighborhood) to tune the parameters [16]. Initially, we keep on exploring the region (by making large changes in the parameter values) till we get significantly better results (f-measure and accuracy). Thereafter, we exploit the region (by making small changes in the parameter values) till we get the best result of that region.

TABLE I
INTELLIGENT MANAGERIAL INFORMATION ABOUT EVOLVING SYSTEMS, STATE SERIES, EDSM SIZE, AND EXPERIMENTAL RESULTS

Domain of Evolving System	Evolving system	States (N+1)	N_DSMs makes EDSM size N × m×m	DEL Variant	Training parameters LR-Ep-L	k-Fold, where k is (N) number of states, which is constant for an experiment					Forward-chaining for time (state) series, here number of states used for training varies from 1 to N				
						Acc.	Pre.	FMe.	Rec.	Avg.	Acc.	Pre.	FMe.	Rec.	Avg.
(A) Evolving Software System	Hadoop-HDFS[1]	15	14 × 3129 × 3129	RBM	0.5-50-50	0.999	0.742	0.838	0.973	0.888	0.999	0.901	0.886	0.875	0.915
				DBN	0.01-10-(10× 5)	0.999	0.718	0.831	0.997	0.886	0.999	0.883	0.913	0.952	0.936
				dA	0.1-10-10	0.999	0.726	0.830	0.978	0.883	0.999	0.889	0.907	0.931	0.931
(B) Evolving Natural language Systems	List of Bible Translation[2]	5	4 × 25 × 25	RBM	0.01-100-100	0.882	0.378	0.464	0.872	0.649	0.96	0.612	0.522	0.657	0.687
				DBN	0.001-100-(100×5)	0.966	0.548	0.598	0.886	0.749	0.977	0.685	0.615	0.657	0.733
				dA	0.01-100-100	0.966	0.548	0.598	0.886	0.749	0.977	0.685	0.615	0.657	0.733
	List of Multi-sport events[3]	8	7 × 14 × 14	RBM	0.1-100-100	0.895	0.32	0.475	0.958	0.662	0.93	0.394	0.515	0.784	0.655
				DBN	0.001-100- (100×5)	0.921	0.408	0.565	0.964	0.714	0.936	0.418	0.531	0.748	0.658
				dA	0.01-100-100	0.918	0.383	0.536	0.947	0.696	0.932	0.404	0.524	0.784	0.661
(C) Evolving Retail Market System	Retail Market[4]	13	13 × 118 × 118	RBM	0.2-100-100	0.965	0.582	0.663	0.81	0.755	0.969	0.66	0.69	0.784	0.775
				DBN	0.001-100-(100×5)	0.965	0.555	0.664	0.859	0.760	0.97	0.651	0.686	0.770	0.769
				dA	0.1-100-100	0.953	0.484	0.616	0.896	0.737	0.966	0.624	0.685	0.806	0.770
(D) Evolving IMDb movie genre systems[5]	Positive sentiment[6] of movie genres[5]	12	11 × 25 × 25	RBM	0.01-100-100	0.914	0.349	0.496	0.961	0.68	0.935	0.405	0.529	0.823	0.673
				DBN	0.001-100-(100×5)	0.914	0.349	0.496	0.961	0.68	0.945	0.444	0.564	0.823	0.694
				dA	0.01-100-100	0.914	0.35	0.498	0.961	0.680	0.945	0.444	0.565	0.823	0.694
	Negative sentiment[6] of movie genres[5];	12	11 × 45 × 45	RBM	0.01-100-100	0.89	0.277	0.404	0.924	0.623	0.937	0.374	0.479	0.689	0.619
				DBN	0.001-100-(100×5)	0.915	0.313	0.446	0.885	0.639	0.947	0.435	0.518	0.682	0.645
				dA	0.001-100-100	0.891	0.291	0.421	0.921	0.631	0.947	0.435	0.518	0.682	0.645

1. https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs Oct 2016.
2. https://en.wikipedia.org/wiki/List_of_English_Bible_translations Oct 2016.
3. https://en.wikipedia.org/wiki/List_of_multi-sport_events Oct 2016.
4. https://archive.ics.uci.edu/ml/datasets/Online+Retail Oct 2016.
5. http://www.imdb.com/interfaces/ Oct 2016.
6. https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

For each evolving system, the DEL variants (RBM, DBN, and dA) make three Deep SysNNs as described in Fig. 3. Each Deep SysNN reconstructs a *zero matrix* M_Z into an *output matrix* M_O of size (m×m). The M_O is transformed to a binary *normalized output matrix* M_{NO} of size m×m.

- **Third**, in testing, the ***k-Fold_forwardChaining*** algorithm compares the *normalized output matrix* M_{NO} with the *testing matrix* M_T. The similarity between the binary matrices (M_{NO} and M_T) will provide a measure of – "how well the M_{NO} mimics the pattern of the old states". The algorithm calculates similarity between the two binary matrices (M_{NO} and M_T) using statistical metrics: *accuracy*, *precision*, *recall* and *f-measure*.

The accuracy is the measure of "how many of all connections and no-connections have been recommended correctly". The precision is the measure of "how many of all connections have been recommended correctly". The recall is the measure of "how many connections have been recommended correctly". The f-measure is the harmonic mean of the precision and recall. These four metrics results are used to evaluate the SysEvoRecomd-Tool. Higher value of a metric represents high similarity (means good result) and low value of a metric represents dissimilarity (means bad result).

In both the matrices (M_{NO} and M_T), an *entity-connection* ec_{jk} (at j^{th} row and k^{th} column), each '1' represents there exist connection and each '0' represents no-connection (i.e. connection do not exist) between two entities ('j' and 'k'). The accuracy metric represents the similarity between all the values (0's and 1's) of two binary matrices, whereas precision, recall, and f-measure represent the similarity between all the values of 1's. The matrices used in the experiments have imbalancing of few 1s and many 0s.

Mathematically, we re-formulate these well-known metrics for our goal in the following way, where, # denotes "count of"

$$accuracy = \frac{\left(\#\begin{matrix} correctly\ recommended\ connections + \\ correctly\ recommended\ no\ connections \end{matrix}\right)}{\#\ all\ possible\ entity\ connections\ i.e\ matrix\ size}$$

$$precision = \frac{\#\ correctly\ recommended\ connections}{\#\ all\ connections\ recommended\ by\ the\ tool}$$

$$recall = \frac{\#\ correctly\ recommended\ connections}{\left(\#\begin{matrix} correctly\ recomended\ connections + incorrectly \\ recommended\ connections\ as\ no\ connections \end{matrix}\right)}$$

In Table 1, k-Fold and forward-Chaining results provide an assessment of the tool's performance. In Figs. 5 and 6, we plotted all the results mentioned in Table 1. The plots show our recommender system is good at learning binary input patterns to generate binary output matrices. The k-Fold and forward-chaining is used to provide explainability and interpretability, we inferred following conclusions.

- In Fig. 5, we presented the two kinds of cross-validation results in the two graphical plots. The figure provides explainability that the accuracy and recall are high as compared to the f-measure and precision.
- The Fig. 6 demonstrates interpretability that the k-Fold and the forward-chaining produced almost similar results.

Graph Neural Network and System Network Reconstruction: There are three reconstructed networks M_{NO}, which is *system network reconstructions* generated from 3 variants SysNN. The recommendation by *system network reconstruction*, we used four metrics: *precision*, *recall*, *accuracy*, and *f-measure*. The quality of recommendation by the *system network reconstruction* can be seen in the Figs. 5 and 6.

Intelligent Managerial Observations: Finally, we discuss our experience while conducting experiments. Our experiments are novel for studying system evolution. Nevertheless, we present following outcomes of our observations.

1. For learning purposes, our DEL solves the problem for learning of an evolving system. As expected, we observed that recommendation done by DBN and dA has outperformed the RBM. The DEL generates Deep SysNN that has information about evolution and changes happened in a system state series. *We found SysNN consist of information about the relationship between system entities in various evolving system states.*

2. Like human beings, a machine also needs both logical and memory oriented learning. Logical learning (or generalization) provides analytical power to the machines, whereas memorized learning (or specialization) provides recommendation power to the machine. Thus, memorization of old memory helps machine to recommend an upcoming and non-existing data. Generally, the patterns are almost constant in every system state, which can be learned and stored as a memory. We did *entity-connection* patterns learning and then stored in disk memory in the form of a reconstructed information (*normalized output matrix* M_{NO}). This makes our results accurate and precise. *Hence, we found learning and memorization both are useful for a machine to mimic patterns like a human.*

3. Figure 5 and 6 shows natural-language based domains (B and D) are harder to learn as compared to software and market based domains (A and C). We made two inferences. First, natural-language based system data (list of bible translation and multi-sport events) are hard to learn for recommendation.

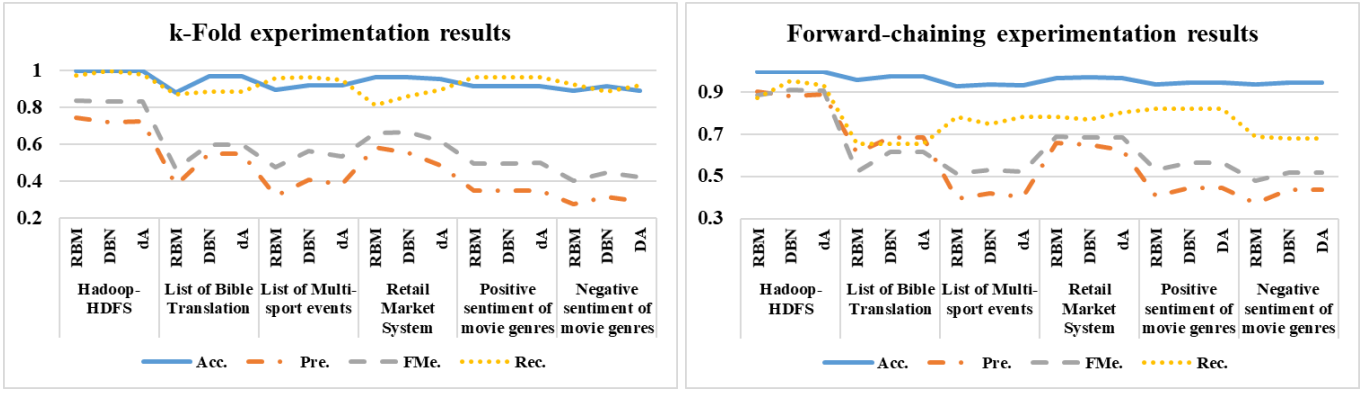


Fig 5. **Explainability:** Each plot shows 4 series for 4 metrics results on 18 experiments (3 DEL variants × 6 evolving systems). The experiments on four domains (A, B, C, and D) at the horizontal axis as mentioned in Table 1 (with the same sequence of the values).

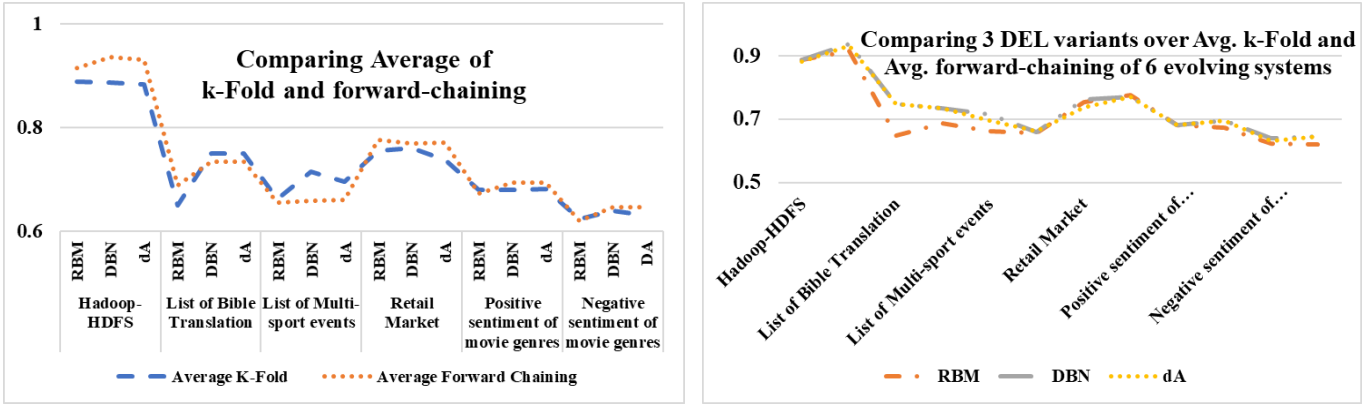


Fig 6. **Interpretability:** Inferred the Average of k-Fold and Forward-chaining over all the 18 experiments are almost similar. Inferred the 3 DEL variants are producing almost similar recommendations with Average k-Fold and Average forward-changing of all evolving systems.

Second, inter-procedural calls of software are easier to learn for recommendation. The medium of communications (e.g. natural-languages) used by humans are significantly complex, whereas the medium of communications (e.g. inter-procedural calls) used by the computers are relatively less complex in nature. *Hence, we found learning of software systems (used by machines) relatively easier as compared to learning of natural-language systems (used by humans).*

4. When an entity (or feature) is present in some states of training data but missing in other states of training data, it is hard to learn and recommend about such an entity. Generally, in machine learning algorithms the size of features are fixed. However, in our case the number of entities (as features) is different in the different states. For example, some entities (software procedures or natural-language words) exist in some states but do not exist in other states. Thus, in the DSM, the entities that do not exist in the state are used with zeros at its row and column. For example, when new procedures are added to a new version of an evolving software corresponding to new functionalities. It is hard to recommend these new procedures that never existed in old software versions used for training. *Hence, we found it is hard to recommend the future connections for the entities (or features) that are absent in many training states or present in few training states.*

5. Accuracy metrics alone cannot give a justified measure for the correctness of a model, thus we used precision, recall and f-measure. It is because of imbalanced data, which means classes (0s and 1s) are not distributed equally and learning from an imbalanced dataset is a challenging task [17]. The datasets used in our experiments are imbalanced, which means number of entity-connections (ec_{jk} = 1 or 0) are unevenly distributed in EDSM. In our case (see Table II), distribution of zeroes ('0's) and ones ('1's) are uneven i.e. the number of '0's are significantly higher than the number of '1's. For entity-connection recommendation, true positive TP (connection recommended correctly) has more significance as compared to TN (true negative). Similarly, false positive FP (connection recommended incorrectly) has more significance as compared to FN (false negative). Due to an imbalanced dataset, we are getting a high value of accuracy metric as compared to f-measure. Because accuracy metric compares the two matrices bit-by-bit assuming each recommendation has the same significance, whereas this fails on imbalanced data. Thus, we used precision, recall and f-measure metric. *Hence, for imbalanced data, we found that along with accuracy metrics, it is also required to use other evaluation metrics: precision, recall, and f-measure.*

Intelligent Managerial Applications: Our approach helps to intelligently automate the management process of system development, evolution, and maintenance. For both existing and upcoming states, we used the time variant (or non-stationary) data of four system domains. This leads to the following advantages for a project manager. First, it can help to predict connection between entities: call between two procedures, link between two words, purchasing of items by a customer, and relation between two movies. Second, it can also be helpful while upgrading software systems, correcting natural-language errors, improving retail market distribution, and targeting audience for a genre. Third, it is helpful during software development, rephrasing a text, doing target marketing, and while naming a movie. Fourth, it can assist system employees e.g., software programmer, writers (of book, article, and novel), retail market (sales, finance, and marketing team), and film production team. Fifth, it can help to speed-up the software development, predictive text, customer billing, and selecting movie names. Sixth, it can do automatic correction of some errors during software debugging, autocorrecting while writing text, customer satisfaction, and re-making a movie. Seventh, it can determine the possible future of the software, text usage trend, market analysis, and videos naming.

VI. Related Works and Comparisons

This section describes and compares the current state-of-the-arts with our work. Our approach is sufficiently new, best to our knowledge it is hard to find other quantitatively comparable approaches that provide statistical or empirical evaluation. We will discuss qualitative comparison of our System Structure Learning (SSL) and SysNN with other machine learning techniques. Assuming every technique has its pros and cons, we discuss comparison under the following three categories.

a) Well-known DSM techniques: Kosari et al. [19] provided equivalency between properties of dynamic systems modeled in state space and numerical process DSMs. Representation Learning techniques use graphs efficiently e.g. Biomedical networks [20]. We extended DSM and graph theory as EDSM for training a SysNN. As compared to other techniques, the advantage of SysNN is that it is used to model non-linear relationships between entities (or features) in time-variant data.

b) Advanced ANN: There are different types of ANN and DNN e.g. *Graph Neural Network* (GNN) [21], *Deep Recurrent Neural Network* (RNN) [22], *Long Short-Term Memory* (LSTM) [23], and *Graph Attention Network* (GAT) [24], which has directed cycle to capture dynamic behavior of time-variant or sequence data. The ANN is extended to RNN and GNN. Similarly, we extended the DNN to SysNN, which learns from multiple evolving graphs of a system state series. The Deep RNN, GNN, GAT, and SysNN are advanced ANNs, but they are defined for different purposes. SysNN is better than RNN and GNN for system analysis because SysNN has system information as it is trained from entity-connections in DSM of multiple states. The SysNN covers broader applications for evolving systems as compared to other NN.

Memory Networks [25] is a network that learns a given knowledge for answering questions, e.g. Sukhbaatar et al. [26] demonstrated an ANN trained using back-propagation and

TABLE II. Information about imbalanced dataset Used.

Evolving Systems	Testing matrix (M_T)	Number of 1s	Number of 0s
Hadoop HDFS	Version 2.7.2	2938	9787703
List of Bible Translation	20th Century	46	579
List of Multi-sport Events	201 i.e. 2010-2017 decade	8	188
Frequent Market Basket	1211 i.e. Dec. 2011	617	13307
Positive sentiment of movie genres	201 i.e. 2010-2019 decade	47	578
Negative sentiment of movie genres	201 i.e. 2010-2019 decade	177	1848

Ankit et al. [27] introduced Dynamic Memory Network (DMN). Gulli et al. [31] invented a method and a system to store, extract, and analyze entities from the temporal content to present temporal trends according to user search query.

Lin et al. [32] proposed Knowledge Graph Neural Network (KGNN) that learns structures of drugs from a knowledge graph to predict drug-drug interaction. Zhang et al. [33] proposed Heterogeneous Graph Neural Network (HetGNN) that aims to resolve learning issue due to heterogeneous information in nodes; this to helps in graph mining tasks for example link prediction, recommendation, and node classification & clustering. Wang et al. [34] proposed Heterogeneous graph Attention Network (HAN) using hierarchical attentions at node and semantic-level; this helps in classification and clustering. Kosan et al. [35] proposed Dynamic Graph Event Detection (DyGED, which combines a Graph CNN and RNN to learn labeled events.

There are some analogies between a natural neural network, and a SysNN. In biology, memories are also found in dolphins [28], chimpanzees and orangutans [29], and hyenas [30]. Similarly, SysNN enhanced capablity to remember and understand relationships between two separate entities, objects, and features. Extending the state-of-the-art, we proposed an ECL model that creates SysNN (a novel feed-forward ANN), which represents a system structure. The Deep SysNN learns and memorizes useful repeating (or mimic) patterns of system entity-connections for intelligent state series management. We found that SysNN generates M_{NO} stored in disk memory, which is useful to make recommendation about the evolving systems. Usually, the SSL will be superior to most of the simple machine learning techniques. It is hard to do comparative analysis with complex techniques.

c) System Network Evolution mining: Our system network evolution subgraph (graphlet and motif) mining [36], which provides frequency and complexity information about patterns formed by connections between entities over multiple states. Our Stable Network Evolution Rule mining [37], which provides rules only for frequently co-occurring entities over multiple states. The six evolving systems used in experiments are also used in our previous publications to retrieve evolution subgraphs and rules for calculating system network complexity [36] and system changeability metric [37]. Both approaches [36][37] have their own importance and applications, the SSL (with the SysNN) is better to make recommendations for all connections between entities i.e. (m×m × N).

Limitations: We found following two limitations

- As the approach extends the deep learning and the deep neural network, which also have some drawbacks. Our SSL and SysNN also has similar drawbacks e.g. it is complex to understand, significant amounts of states (as data) required for training, and it takes time to learn. Relatively some algorithms (e.g. support vector machine, decision trees, and regression) are simpler, faster, and easier to learn a dataset. The SSL approach is computationally expensive as compared to our other approaches on the same evolving system's dataset [36][37]. Still we found SSL is much superior at recommendations as compared to both [36][37].

- Large size matrice take a long time to do deep learning. For example, as Hadoop-HDFS has the large EDSM size 14 × (3129 × 3129), it took more time to learn as compared to other five systems. In the same environment, we found learning smaller EDSM took lesser time as compared to the large EDSM. The six EDSM used as input are huge and computationally expensive for real-world applications. However, our efficient Java codes of SysEvoRecomd-Tool based on SSL algorithm solved such real-world applications in feasible-time.

VII. Conclusions

We introduced Evolution and Change Learning (ECL) that forms *System Neural Network* (SysNN). We elaborated this to propose a *System Structure Learning* (SSL) algorithm, which internally uses *evolution representor* (Evolving DSM) and *deep evolution learner* (DEL). The DEL uses EDSM to construct a Deep SysNN representing information of system states. Three DEL variants generated three Deep SysNNs. We conducted experiments on six evolving systems collected from open internet repositories of four domains. This demonstrated four intelligent applications with satisfactory accuracy, precision, f-measure, and recall. The experiments present the study over challenging imbalanced data of the evolving systems.

In future, instead of unsupervised deep learning another technique can be used for example, supervised (e.g. active learning), semi-supervised, and reinforcement learning. Applying such techniques on evolving systems is still a research topic. Our approach helps to intelligently manage other kinds of systems in temporal analysis. The SysNN may be useful to realize a hardware device on Spiking Neural Network (SNN) [38]. Our approach could be applicable to systems including IoT and software network evolution [39], which may have devices, procedures, and components working together as interacting or connected entities. Our algorithms are linear, in future scalable parallel-programming can be introduced. Our approach could be an innovative way to analyse evolution of technological systems, Hughes [40]. Further we expect application of our approach to study and analyze the Complex Adaptive Systems (CAS) [41] and Systems of Systems (SoS) [42]. Effort toward developing CAS and SoS has raised maintenance and evolution issues of versioning or states, where our approach would be helpful.

Acknowledgment

The authors thank graduate students Krishna Chaitanya and Harsh Mohan of IIT Indore for their initial help to lead author.

References

[1] L. V. Bertalanffy, "General system theory." *General systems* 1.1 (1956): 11-17.

[2] D. V. Steward "The design structure system: A method for managing the design of complex systems." *IEEE Trans. on Engineering Management* 3 (1981): 71-74.

[3] T. R. Browning "Design structure matrix extensions and innovations: a survey and new opportunities." *IEEE Trans. on Engineering Management* 63.1 (2016): 27-52.

[4] Y. LeCun, Y. Bengio, an[1d G. Hinton. "Deep learning." Nature 521.7553 (2015): 436-444.

[5] G. W. Taylor, G. E. Hinton, and S. T. Roweis. "Modeling human motion using binary latent variables." Advances in Neural Information Processing Systems. 2006.

[6] G. E. Hinton, S. Osindero, and Y.-W. Teh. "A fast learning algorithm for deep belief nets." Neural computation 18.7 (2006): 1527-1554.

[7] P. Vincent, et al. "Extracting and composing robust features with denoising autoencoders." *25th Int. Conf. on Machine learning*. ACM, 2008.

[8] G. Ditzler, et al. "Learning in nonstationary environments: A survey." *IEEE Computational Intelligence Magazine* 10.4 (2015): 12-25.

[9] A. Chaturvedi, & A. Tiwari. "System Evolution Analytics: Deep Evolution and Change Learning of Inter-Connected Entities". *2018 IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)* (pp. 3075-3080).

[10] A. Chaturvedi and A. Tiwari. "System Evolution Analytics: Evolution and Change Pattern Mining of Inter-Connected Entities". *2018 IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)* (pp. 3877-3882).

[11] A. Chaturvedi, et al. "Service Evolution Analytics: Change and Evolution Mining of a Distributed System." *IEEE Trans. on Engineering Management* 68.1 (2020): 137-148.

[12] A. Chaturvedi, and A. Tiwari. "SysEvoRecomd: Graph Evolution and Change Learning Based System Evolution Recommender." *2018 IEEE Int. Conf. on Data Mining Workshops (ICDMW)* (pp. 1499-1500). IEEE.

[13] A. Chaturvedi, A. Tiwari and S. Chaturvedi, "SysEvoRecomd: Network Reconstruction by Graph Evolution and Change Learning," in *IEEE Systems Journal*, doi: 10.1109/JSYST.2020.2988037.

[14] J. D. Rodriguez, A. Perez, and J. A. Lozano. "Sensitivity analysis of k-fold cross validation in prediction error estimation." *IEEE Trans. on Pattern Analysis and Machine Intelligence* 32.3 (2010): 569-575.

[15] R. Kohavi, and G. H. John. "Wrappers for feature subset selection." *Artificial intelligence* 97.1-2 (1997): 273-324.

[16] M. Črepinšek, S.-H. Liu, and M. Mernik. "Exploration and exploitation in evolutionary algorithms: A survey." *ACM computing surveys (CSUR)* 45.3 (2013): 1-33.

[17] H. He, and E. A. Garcia. "Learning from imbalanced data." *IEEE Trans. on Knowledge and Data Engineering* 21.9 (2009): 1263-1284.

[18] S. Yusuke. "Java deep learning essentials." (2016).

[19] A. Kosari, M. H. Jafari, and M. Fakoor. "On Equivalency Between Numerical Process DSM and State-Space Representation." *IEEE Trans. on Engineering Management* 63.4 (2016): 404-413.

[20] M. M. Li, K. Huang, and M. Zitnik. "Representation Learning for Networks in Biology and Medicine: Advancements, Challenges, and Opportunities." *arXiv preprint arXiv:2104.04883* (2021).

[21] Z. Wu, et al. "A comprehensive survey on graph neural networks." IEEE Trans. on Neural Networks and Learning Systems (2020).

[22] M. Hermans, and B. Schrauwen. "Training and analysing deep recurrent neural networks." *Advances in neural information processing systems* 26 (2013): 190-198.

[23] S. Hochreiter, and J. Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[24] P. Veličković, et al. "Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

[25] J. Weston, S. Chopra, and A. Bordes. "Memory networks." *arXiv preprint arXiv:1410.3916* (2014).

[26] S. Sukhbaatar, J. Weston, and R. Fergus. "End-to-end memory networks." *Advances in Neural Information Processing Systems*. 2015.

[27] A. Kumar, et al. "Ask me anything: Dynamic memory networks for natural language processing." *Int. Conf. on Machine Learning*. 2016.

[28] J. N. Bruck "Decades-long social memory in bottlenose dolphins." *Proceedings of the Royal Society of London B: Biological Sciences* 280.1768 (2013): 20131726.

[29] G. Martin-Ordas, D. Berntsen, and J. Call. "Memory for distant past events in chimpanzees and orangutans." *Current Biology* 23.15 (2013): 1438-1441.

[30] H. E. Watts, and K. E. Holekamp. "Interspecific competition influences reproduction in spotted hyenas." *J. of Zoology* 276.4 (2008): 402-410.

[31] A. Gulli, F. Tanganelli, and A. Savona. "System and method for monitoring evolution over time of temporal content." *U.S. Patent* Application No. 11/313,584, 2007.

[32] Lin, Xuan, et al. "KGNN: Knowledge Graph Neural Network for Drug-Drug Interaction Prediction." IJCAI. Vol. 380. 2020.

[33] Zhang, Chuxu, et al. "Heterogeneous graph neural network." Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019.

[34] Wang, Xiao, et al. "Heterogeneous graph attention network." The World Wide Web Conference. 2019.

[35] Kosan, Mert, et al. "Event Detection on Dynamic Graphs." *arXiv preprint arXiv:2110.12148* (2021).

[36] A. Chaturvedi and A. Tiwari. "System Network Complexity: Network Evolution Subgraphs of System State series". *IEEE Trans. on Emerging Topics in Computational Intelligence* (2018).

[37] A. Chaturvedi, A. Tiwari, and N. Spyratos. "minStab: Stable Network Evolution Rule Mining for System Changeability Analysis." *IEEE Trans. on Emerging Topics in Computational Intelligence* (2019).

[38] W. Maass. "Networks of spiking neurons: the third generation of neural network models." *Neural networks* 10.9 (1997): 1659-167.

[39] M. A. Fortuna, J. A. Bonachela, and S. A. Levin. "Evolution of a modular software network." *Proceedings of the National Academy of Sciences* 108.50 (2011): 19985-19989.

[40] T. P. Hughes, "The evolution of large technological systems." *The social construction of technological systems: New directions in the sociology and history of technology* 82 (1987).

[41] J. H. Holland. "Complex adaptive systems." *Daedalus* (1992): 17-30.

[42] A. Gorod, B. Sauser, and J. Boardman. "System-of-systems engineering management: A review of modern history and a path forward." *IEEE Systems Journal* 2.4 (2008): 484-499.

Animesh Chaturvedi is working as an Assistant Professor at the Data Science and Intelligent Systems Department of the Indian Institute of Information Technology Dharwad. He was a Post-Doctoral Research Assistant at King's College London and working with The Alan Turing Institute. He completed his PhD degree from IIT Indore. He received the BEng degree from IET - DAVV, Indore and the MTech degree from IIITDM, Jabalpur. He did research work with the IIT-Kanpur, Motorola, and Arris. Dr. Chaturvedi has been selected as 200 young researcher for Heidelberg Laureate Forum (HLF) 2019.

Aruna Tiwari received the B.Eng., M.Eng., and Ph.D. degrees in computer engineering. She is working as an Associate Professor of Computer Science and Engineering at IIT Indore, since 2012. Dr. Tiwari has been a Reviewer for many journals and conferences. She has research collaborations with CSIR CEERI Pilani and the Indian Institute of Soyabean Research (Indian Council of Agriculture and Research).

Shubhangi Chaturvedi is working towards the PhD degree at the Indian Institute of Information Technology, Design and Manufacturing, Jabalpur. She received the BEng degree from RGPV University, and the MTech degree from National Institute of Technology Bhopal, Bhopal. She has been Assistant Professor of National Institute of Technology Bhopal, Bhopal. Her research interest is in Data mining and Big Data Analytics.

Pietro Lio is Full Professor at the Department of Computer Science and Technology of the University of Cambridge. He is a member of the AI group and the Cambridge Centre for AI in Medicine. He holds a PhD in Genetics and a second PhD in Non-Linear Dynamics and Complex Systems. He is fellow and member of the Council of Clare Hall College, the Ellis, the European Lab for Learning & Intelligent Systems and the Academia Europaea. His research interest focuses on AI, Computational Biology, Graph NN, Explainability and Interpretability.