System Evolution Analytics: Deep Evolution and Change Learning of Inter-Connected Entities

Animesh Chaturvedi
Indian Institute of Technology Indore
Indore, India
animesh.chaturvedi88@gmail.com

Aruna Tiwari
Indian Institute of Technology Indore
Indore, India
artiwari@iiti.ac.in

***Abstract*— Entities (or components) in an evolving system keeps on evolving, which makes a state series SS = {S_1, S_2… S_N}, where S_i is the ith state of the system. There exist connections (or relationships) between entities, which also evolve over system state, and make a series of evolving networks EN = {EN_1, EN_2… EN_N}. We can use these evolving networks to do learning over evolving system states for system evolution analysis. In this paper, we introduce a *System Evolution Analytics* model, which is based on proposed *System Evolution Learning*. The network pattern information is trained using *graph structure learning*. The evolution information is trained using *evolution and change learning*. We accomplish this by implementing a *deep evolution learning*. This technique uses an evolving matrix to generate *evolving memory* in the form of a proposed System Neural Network (SysNN). The SysNN is useful to predict and recommend based on system evolution learning. The technique is prototyped as a tool, which is used to do experiments on six evolving systems. We applied our tool to do system evolution analysis. The experiments are conducted to generate and report *evolving memory* as SysNN that helps to do recommendation about system.**

***Keywords*— evolving system, graph (network) learning, machine learning, deep learning, artificial neural network**

I. Introduction

Evolving systems [1][2][3][4][5] are the systems which keep on evolving in a changing, dynamic, and evolving environment. Suppose a state series S = < S_1, S_2… S_N > of an evolving system at different instances of time < t_1, t_2… t_N >. The state series is similar to the well-known time series and data series [6]. An evolving system analysis deals with control, prediction, classification, data processing, unpredictable environments, and adaptable systems. Example of evolving system is an evolving software system [7] and an evolving business system [8]. An evolving system can be pre-processed to make non-stationary data and time variant data that can be used for machine-learning purpose. An evolution and change analysis [9][10] or a non-stationary learning [11] extracts useful information from time-variant or non-stationary data. Such analysis and learning techniques are applied in many domains e.g., source code change analysis [12].

Artificial Neural Network (ANN) is a paradigm of machine learning, which is inspired from biological systems. The ANN models are based on simplified topology and tries to mimic information processing capabilities of the human nervous systems in a computational domain. *Deep Neural Network* (DNN) [13] is a class of ANNs, which are trained using *deep learning* algorithms. The *DNN* models have been receiving a lot of research interest recently, especially because of promising results in image or speech or natural language processing.

Naturally created data often contain patterns, it is possible for humans to identify and label such patterns. These labels prove quite useful to train machine-learning models, known as '*supervised learning*' models. However, in other cases, it becomes quite tedious or even impossible to do such a labeling of patterns in data. For such cases, there exist different classes of machine learning models called '*unsupervised learning*' models. The ability of unsupervised techniques to learn directly from unlabeled data makes them very useful, especially for our purpose, to detect the history of change patterns in an evolving system. We will discuss three well-known unsupervised neural network models: *Restricted Boltzmann Machines* (*RBM*), *Deep Belief Networks* (*DBN*), and *denoising Autoencoders* (*dA*).

First DNN is based on the work of Paul Smolensky [14], who invented Harmoniums in 1986, which is now known as RBM. The RBM gained popularity quite later in 2006, when Geoffrey Hinton et al. [15] described a fast learning algorithm to train DBN. The persistent contrastive divergence algorithm proposed by Tieleman [16], who introduced a fast and simple way to train such models. *Autoencoders* have been receiving a lot of interest in the research. Some of the promising variants of *autoencoders* are the *denoising Autoencoders* (*dA*), *Sparse Autoencoders* and *Variational Autoencoder*. However, we restrict ourselves only with the dA [17], due to its simplicity among all. In short, we used three *deep learning* algorithms (RBM, DBN, and dA); thus, we explicitly give a preliminary of them.

***Restricted Boltzmann Machine* (*RBM*)** is a type of neural network, where the neurons in the network topologically form a complete bipartite graph, separating the neurons into visible units and hidden units. The weights between the nodes are denoted by a matrix W, where W_{ij} represents the weight between ith visible unit and jth hidden unit. The state of ith visible neuron is denoted by the variables v_i and that of the jth hidden neuron by h_j. Energy of the machine is defined by the formula

$$E(v, h) = -\sum_{i,j} v_i h_j W_{i,j} - \sum_i v_i b_i - \sum_j h_j b_j$$

where b stands for the biases. This energy formula is used to make a probability equation defined by

$$P(v, h) = e^{-E(v, h)/Z} / Z$$

where Z is the normalization constant $Z = \sum_{v,h} e^{-E(v, h)}$. Therefore, the probability of a data point can be defined as

$$P(v) = \sum_h P(v, h) \qquad \textbf{...Eq1}$$

Now, the network can be trained by adjusting the weights and biases, which maximizes the probability for the training set. To do this, the most common and efficient way is to use the approximate gradient of the contrastive divergence [18].

***Deep Belief Network* (*DBN*):** There are two phenomena *shallow neural network* and *deep neural network*, which explain the fundamental understanding of DBN. A *shallow neural network* can classify a data with the large error rate, which is good for surface level learning. *Deep neural networks* can classify a data with the small error rate, which is good for deeper level learning. Before DBN, the *shallow networks* dominated the field of neural networks, and the *deep architectures* (or *deep networks*) suffered from vanishing gradient problem [19]. However, due to the novel learning techniques developed by Hinton et. al. [15], even deep architectures can now be efficiently trained by training each layer greedily. Train the first, second, and other layers as greedy layer-wise pre-training using RBM is given by the formula

$$P(x, h^1, h^2\ldots, h^L) = ({}^{L-2}\pi_{k=0}\, P(h^k \mid h^{k+1}))\, P(h^{L-1}, h^L) \quad \textbf{...Eq2}$$

where x represents input vector, h represents hidden layer and L is the number of hidden layers. Then fine-tune the parameters of deep network using supervised gradient descent of the negative log-likelihood cost function [15]. Such neural network gives better results in different applications.

***Autoencoders*:** With the rise of popularity in deep architectures a new *deep learning* technique *autoencoders* also gain popularity. The *autoencoders* learns internal patterns of the input data x by encoding it into some representation c(x) so that it can be later reconstructed. Simplest *autoencoders* consist input, hidden, and output layers.

Bengio et al. [20][21] suggested that stochastic gradient descent yields useful results. The network is usually trained by minimizing the negative log-likelihood of the reconstruction error. Thus, this minimizes error of reconstruction. In case of binary inputs, the error of reconstruction is calculated as the sum of Bernoulli cross-entropies, which is given by formula

$$L(x, z) = - \textstyle\sum_i x_i \log[z_i + (1-x_i) \log(1- z_i)] \quad \textbf{...Eq3}$$

where $z_i = d(c(x_i))$ represent reconstructed x_i and where, c(x) represents the coded x and d(c(x)) represents decoded c(x).

***Denoising autoencoders* (*dA*)** is introduced by Vincent et al. [17]. The dA is modification of the original *Autoencoder*. In dA, during the training phase instead of the actual input data, a slightly corrupted version of the input is given to the neural network. This makes a neural network more robust to changes in input data comparatively to its predecessor.

In this work, we propose an algorithm and a tool to learn time variant data of evolving system. We make deep learning to extract useful information from *time variant datasets* of an *evolving system*. In our approach, we describe how the deep learning can be used to create DNNs for an *evolving system*. This technique reduces human intervention by recommending about evolution of the evolving system. The recommender system is constructed to help in decision making about evolving system.

The major contributions of the paper, we describe a model for system evolution analytics based on *system evolution learning* to train a machine for system graph evolution and change(s). The learning generates meaningful knowledge (in the form of a neural network) for evolution happened to the source and target entity-connection patterns.

Rest of the paper is organized as following. In Section 2, we describe the system evolution analytics model based on evolution and change learning of evolving system. In Section 3, we describe the resulting report of the experiments on six evolving systems. Then description of related contributions in Section 4 and concluding remarks in Section 5.

II. System Evolution Analytics Based on Deep Learning of Evolving System

In this section, we describe our proposed system evolution analytics model for a state series. We start by defining a state series of an evolving system.

Definition 1: State Series SS is defined as "a collection of states (or data points) at various time points $\{t_1, t_2, t_3 \ldots t_N\}$, which makes a state series $SS = \{S_1, S_2 \ldots S_N\}$. Such state series can be pre-processed to make a series of an evolving network $EN = \{EN_1, EN_2\ldots EN_N\}$ corresponding to the system states. Thus, we can define (S_i, EN_i, t_i) such that the S_i represents system state and the EN_i represents its evolving network at the i^{th} time point t_i, where 'i' varies from 1 to N. Collection of such temporal states as a *state series*. An example of an evolving state series is natural language based evolving document versions.

A machine learning technique can learn from an *evolution representor* (such as an unlabeled evolving matrix), which is useful to detect change patterns. The evolution learning depends upon machine learning that uses evolution representor (evolving matrix). Now, we describe the detail of proposed evolution and change learning of time-variant data. First, we describe the *evolution and change learning* technique using an evolution representor.

Definition 2: Evolution and Change learning is a machine learning of evolution and change(s) happened to a time-variant data at various states, using a representation of evolution (e.g., evolving matrix). It uses an evolution information to learn the evolution of a time-variant data of a state series. This makes a computer capable enough to understand the evolution of a time-variant data without any explicit programming.

Evolution and Change learning trains itself from an evolution representor built for a state series of evolving system. An existing machine learning technique can learn from the evolution representor stored in an evolving matrix. Such technique learns and understands the evolution and change(s) in an evolving system. *Evolution and Change Learning* technique can learn evolution and change(s) happened during the evolution of an evolving system. Such techniques can be applied to train a machine for hidden, non-trivial, and non-obvious information about evolution happened in an evolving system. Evolution and Change learning adapts, controls, and analyzes according to the evolution and change(s) automatically. Such learning process takes two or more states of a time-variant data as input to generates evolution information that helps to study temporal states of an evolving system. Next, we define the evolution and change learning for an evolving system.

Definition 3: ***System Evolution Learning*** is a machine learning of evolution and change(s) happened over evolving system states at various time points. With this, a machine becomes capable to understand the system evolution without any explicit programming.

The system evolution learning can be accomplished by learning its evolving system graphs as representation of various states. Thus, next we define *graph evolution learning*, which aids to explain about *network evolution learning* and *system graph evolution learning*.

Definition 4: ***Graph Evolution Learning*** is machine learning of evolution and change(s) happened to evolving graphs from an evolution representor with the help of learning evolution and change patterns. With this, a machine becomes capable to understand the graph evolution without any explicit programming.

- The GEL may also be referred as *Network Evolution Learning* (NEL).

- *System Graph Evolution Learning* is machine learning of evolving system graphs, which makes machine capable enough to understand system graph evolution and change information about evolving inter-connected entities in an evolving system graph.

In our case, for machine learning we used deep learning that forms a deep neural network (a kind of artificial neural network). Thus, due to the use of deep learning we formed a new kind of artificial neural network that we defined as follows.

Definition 5: ***System Neural Network*** (SysNN) is an *artificial neural network* that contains information and understanding of evolution and change(s) happened to an evolving system. The weights are adjusted such that neurons provide probable connections (occurrences) of two entities (features) together. The SysNN help to recommend about the time-variant data (or non-stationary data) by learning and memorizing evolution information about system data.

Problem definition: Let evolution and change(s) happened in an evolving system makes several transition states named as state series. Each state is represented as a system network (or graph) of inter-connected entities, such that each node represents an entity in a state and each edge represents a connection between two entities. Here entity-connections are prone to undergo evolution and change(s). It is challenging to study the changeability and evolution in an evolving system represented as an evolving network series.

The evolution and change learning of an evolving system retrieves hidden evolving information that is further helpful in decision-making. There are following two challenges to do such machine learning of an evolving system:

A. Generally, evolving systems are complex in nature, thus pre-processing of an evolving system is a challenging task and requires a system domain expert.

B. The data mining or machine learning on single state of an evolving system is a straightforward technique. However, it is challenging to process multiple states of an evolving system with the same technique. Therefore, modification of the algorithm is required.

Solution definition: The model has few steps to do evolution and change learning of an evolving system, which are shown in the Figure 1 and described as follows:

1. Preprocess multiple states of an evolving system, which makes a series of evolving networks $\{EN_1, EN_2 \dots EN_{N+1}\}$. Where an evolving network EN_i represents state S_i. Here, each evolving network represents a relationship between evolving inter-connected system entities.
2. Thereafter, represent each of an evolving network as a connection matrix for a state. This makes (N+1) number of connection matrixes to do machine learning on the evolution and change patterns of connections between system entities. Out of (N+ 1) number of matrixes (*N+ 1_Matrices*), only N number of matrixes (*N_Matrices*) are used for training and the remaining 1 matrix is used for the testing purpose. The *N_Matrices* are used to make an evolution representor referred as evolving matrix. This evolving matrix is used for evolution and change patterns learning based on machine learning technique such as active or deep learning.
3. Thereafter, the learning of the evolving connection matrix makes evolution information for prediction. In case of deep learning, the evolution information is in the form of a System Neural Network (hidden black box realized as weight matrix) that can be used to reconstruct a matrix. This makes an output matrix, which is a kind of evolving memory about evolving system's evolution.
4. Then compare the binary output matrix with a testing matrix (representing a state not used for training purpose). To compare, we can use metrics like accuracy, f-measure, precision, and recall. Then, we can demonstrate the metrics result values in the form of graphs, charts, or tables. Then, we can use the report to analyze the quality of recommendation provided by the machine learning for the evolving system.

The four steps of the model can also be represented as four steps in the following algorithm.

```
Algorithm System_Evolution_Learning(repository)
Retrieve N+1 states ($S_1$, $S_2$… $S_N$, $S_{N+1}$) of an evolving system at a repository
1. evolvingNetworks = preprocess(repository)
2. N+1_Matrices = convert(evolvingNetworks)
    ➢ evolving_matrix = evolution_representor(N_Matrices)
3. graph_evolution_learning(evolving_matrix)
    ➢ binary_output_matrix = deep_evolution_learning(evolving_matrix)
4. binary_classifier_metric(binary_output_matrix, testing_matrix)
```

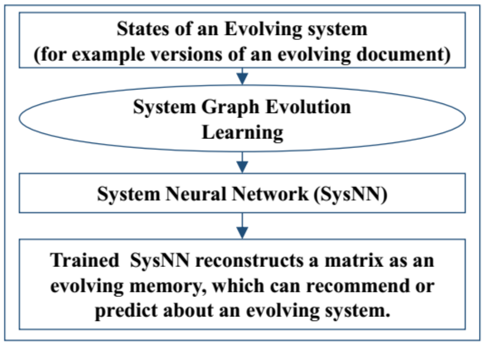
States of an Evolving system
(for example versions of an evolving document)
System Graph Evolution Learning
System Neural Network (SysNN)
Trained SysNN reconstructs a matrix as an evolving memory, which can recommend or predict about an evolving system.

Fig. 1. Overview of the proposed system evolution analytics model.

Usually deep learning technique uses input matrix to generate a reconstructed matrix (as output). We represented each evolving system state as an evolving network, which makes N+1 evolving networks (*evolvingNetworks*) for a state series. Then each evolving network information is transformed to a matrix corresponding to each state. This makes N+1 matrixes (*N+1_Matrices*). Thereafter, we merged N matrixes (*N_Matrices*) to make an *evolving_matrix* that contains information about multiple states for learning purpose.

After this, graph evolution learning uses deep evolution learning to make a neural network that contains system states information. Thereafter, the information is stored as an output matrix that is transformed to a *binary_output_matrix* (an evolving memory). This matrix can be used for recommendation purpose.

The *System Evolution Learning* (*SEL*) generates *SysNN* (as a system graph evolution and change information), which needs to be accurate. Several states of an evolving system can undertake testing of *SEL* performance and accuracy based on information generation. User will classify correctness of the outputs. To improve and achieve correct output, the *SEL* algorithm is continuously adapted and fine-tuned. Fine-tuning will generate information satisfactory according to the human (user). The user will use his experience and understanding (visually and heuristically) to check correctness of the generated information. The *SEL* learns from its user 'how to use *deep learning* and *evolution learning* together', 'how to perform *deep evolution learning*', and 'how to generate *SysNN*'. This process will continue to generate the improved information.

The *System Evolution Learning* underlies upon *graph structure learning* as well as *evolution and change learning* as shown in the Figure 2. Specifically, the *SEL* algorithm takes N+1 state of an evolving system (as input), uses *deep evolution learning* and generates the *SysNN* (as output).

We prototyped the approach as a tool based on the four steps described in the *System Evolution Analytics* model. Specifically, we implemented the *SEL* algorithm in the tool as java application. The tool learns evolution and change patterns from evolving matrix. The tool requires machine enabled with JRE and JDK 7th version or higher.

In the next section, we describe the empirical evaluation of the tool. The evaluation is done for the recommendation provided by the trained SysNNs, which are generated from the system evolution learning on six evolving systems.

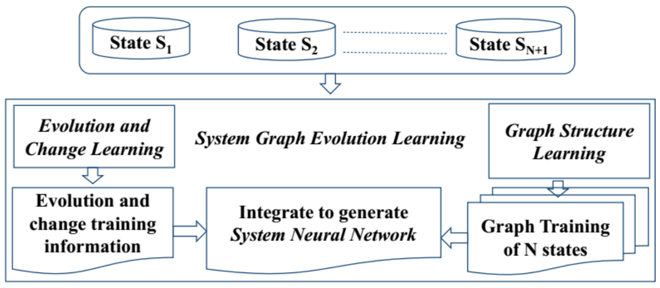
State S_1
State S_2
State S_{N+1}
Evolution and Change Learning
System Graph Evolution Learning
Graph Structure Learning
Evolution and change training information
Integrate to generate System Neural Network
Graph Training of N states

Fig. 2. The System Evolution Learning (SEL) on N states of an evolving system.

III. Experiments

In this section, we describe experimentation, which is done using tool that we implemented for the proposed model. The application of the tool is demonstrated on six different evolving systems by performing experiments. We generated experimental result that provides accurate and precise recommendation about the system network. With this, we demonstrate application of *SEL* on different evolving system. This section is divided into two parts. First part describes the dataset used in the experiments. Second part describes the evaluation of the binary output matrixes built from different *system neural networks*.

Dataset used in the experiment: The information about six evolving system used for the experiments is mentioned in Table II with following columns: evolving system names, links, number of states used for training. The table also mentions the deep evolution learning based on the fundamental deep learning technique. In the last column, it mentions Average of Binary Metrics (explained later) for each of the experiment. We did the experiments independent of each other for all deep evolution learning (i.e. mentioned in a row). In total, we mentioned 18 experiments for 3 deep evolution learning variants applied on 6 evolving systems i.e. ($18 = 3 \times 6$).

We made series of evolving networks for each evolving system by using their entity-connections in the following way. For software, we used a procedure as an entity and a procedure-call as the connection. For list of bible-translations and multi-sport-events, we used word (of natural language) as an entity and sequence between two words as connection. For the retail market system, we used words in product description and customer ID as source and target entities respectively; this makes network of the product purchasing. For the positive and negative sentiment of movie-genre, we used words in the movie name and its genre as the source and target entities respectively; this makes network of movie name words and genres.

Evaluation of System Neural Network (SysNN): To do System Graph Evolution Learning, we extended three deep learning (RBM, DBN, and dA) as three different deep evolution learning approaches. For each evolving system, we used the three deep evolution learning approaches separately, which resulted in three different SysNN.

The SysNN internally aids to make a binary output matrix. This resulting output matrix is compared with the testing matrix to find similarity between the two binary matrixes. For similarity comparison, we used four well-known binary metrics: accuracy, precision, recall, and f-measure. After finding the values of these metrics, we calculated their average referred as Average of Binary Metrics (ABM)

$$\mathrm{ABM} = \frac{(\mathrm{accuracy} + \mathrm{precision} + \mathrm{fmeasure} + \mathrm{recall})}{4}$$

For each evolving system, the three SysNN (for RBM, DBN, and dA) resulted in three value for Average of Binary Metrics. The experimental results are in terms of ABM, which is given in the last column of Table I. The Figure 3 depict a visual representation for the ABM values (as depicted in vertical axis of the figure) over the type of deep learning on an evolving system (as depicted in horizontal axis of the figure).

TABLE I. INFORMATION OF THE EVOLVING SYSTEMS AND SYSTEM NEURAL NETWORK PREDICTION IN THE EXPERIMENTS.

Evolving systems	Internet links	Training States (N)	Deep Evolution learning	ABM
Hadoop HDFS-Core	https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs	15	RBM	0.888
			DBN	0.886
			dA	0.883
Bible Translation	https://en.wikipedia.org/wiki/List_of_English_Bible_translations	13	RBM	0.649
			DBN	0.749
			dA	0.749
Multi-sport Events	https://en.wikipedia.org/wiki/List_of_multi-sport_events	13	RBM	0.662
			DBN	0.714
			dA	0.696
Retail market system	https://archive.ics.uci.edu/ml/datasets/Online+Retail	13	RBM	0.755
			DBN	0.760
			dA	0.737
Positive sentiment of movie genres	http://www.imdb.com/interfaces/ https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html	16	RBM	0.68
			DBN	0.68
			dA	0.680
Negative sentiment of movie genres	http://www.imdb.com/interfaces/ https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html	16	RBM	0.623
			DBN	0.639
			dA	0.631

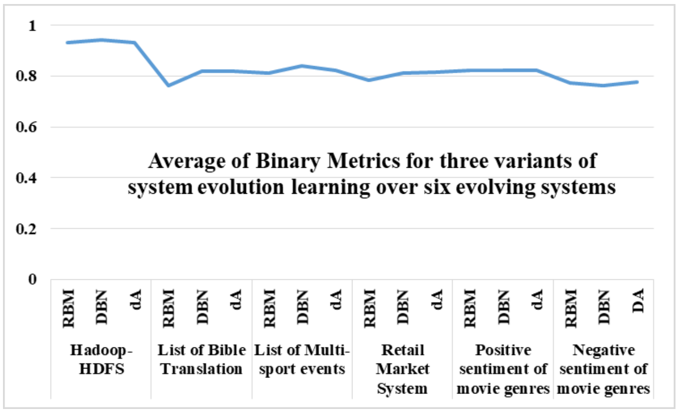

Fig. 3. Graphical demonstration of the recommendation (in terms of ABM) provided by the three variants of system evolution learning applied on the six evolving systems.

IV. RELATED WORKS

In this section, we present contributions related to this paper. To begin with, Yang et al. [22] described Concept Graph Learning (CGL) to study academic graphs of courses and concepts links. Yanardag and Vishwanathan [23] described Deep Graph Kernels to learn sub-structures for graphs using dependencies between sub-structures. Cao et al. [24] proposed a model for learning graph by encoding each vertex as a low dimensional vector representation for learning based on stacked denosing autoencoders. Bui et al. [25] presented Neural Graph Machines using a graph-regularized based on feed-forward neural networks (CNNs and LSTM RNNs) on label/unlabeled data. Somanchi et al [26] proposed an approach of Learning Graph Structure (LGS) a framework for learning graph structures.

Deep RNN is a type of recurrent neural network (RNN), which is constructed using a deep learning approach [27][28]. Deep RNN is useful in many field such as speech recognition [29][30], natural language processing [31], and image recognition [32]. Nasraoui et al. [33] presented an approach to do mining of web usage patterns from web log files for discovering and tracking evolving user profiles. Analogously, we demonstrate experiments on six real-world evolving systems.

Few possible application of our approach is as follows. Our approach can help to construct a recommender system using matrix factorization technique along with temporal information; for further reading refer to Koren and Robert [34], which described matrix factorization techniques for recommender systems. The System Evolution Analytics can improve the system thinking; for further reading Whitehead et al. [35] described about the system thinking. Our technique can also be applicable on other kind of evolving systems to do temporal analysis e.g. recognizing temporal facial action [36] and facial pain expressions [37].

V. CONCLUSIONS

We introduced a model for *System Evolution Analytics*, which is based on *system network evolution learning* over a state series of an evolving system. This learning is a hybrid of *graph learning* with *evolution learning* over multiple states of a system. This learning is realized with the deep learning techniques: RBM, DBN, and dA. The proposed approach is used to implement a System Evolution Analytics tool that learns multiple states of an evolving system. We used the tool to do experiments on six different evolving systems. In the experiments, we have generated the *System Neural Network* (SysNN) for each evolving system. The SysNN can be used to do recommendation or prediction. For recommendation purpose, the SysNN reconstructed a binary output matrix. To measure the quality of recommendation, we compared binary output matrix against a testing matrix of a state that remain unused during training phase. The SysNN can be helpful in system development and maintenance. In future, some other machine learning techniques with evolution and change learning can produce another hybrid evolution learning.

REFERENCES

[1] P. Angelov, and N. Kasabov. "Evolving intelligent systems, eIS." *IEEE SMC eNewsLetter* 15 (2006): 1-13.

[2] N. Kasabov, and D. Filev. "Evolving intelligent systems: methods, learning, & applications." *Evolving Fuzzy Systems, 2006 International Symposium on*. IEEE, 2006.

[3] N. Kasabov. "Evolving intelligence in humans and machines: Integrative evolving connectionist systems approach." *IEEE Computational Intelligence Magazine* 3.3 (2008).

[4] S. A. Frost, and M. J. Balas. "Evolving Systems and Adaptive Key Component Control." (2009).

[5] P. Angelov, D. P. Filev, and N. Kasabov, eds. Evolving intelligent systems: methodology and applications. Vol. 12. *John Wiley & Sons*, 2010.

[6] T. Palpanas. "Data series management: The next challenge." *Data Engineering Workshops (ICDEW), 2016 IEEE 32nd International Conference on.* IEEE, 2016.

[7] T. Mens, A. Serebrenik, and A. Cleve. “Evolving Software Systems”. *Springer Publishing Company, Incorporated*. 2014.

[8] M. Böttcher, D. Nauck, C. Borgelt, and R. Kruse. “A framework for discovering interesting business changes from data”. *BT Technology Journal* 24.2 (2006): 219-228.

[9] Böttcher Mirko, Frank Höppner, and Myra Spiliopoulou. “On exploiting the power of time in data mining.” *ACM SIGKDD Explorations Newsletter* 10.2 (2008): 3-11.

[10] A. M. Ross, D. H. Rhodes, and D. E. Hastings. "Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value." *Systems Engineering* 11.3 (2008): 246-262.

[11] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. “Learning in nonstationary environments: a survey.” *IEEE Computational Intelligence Magazine* 10.4 (2015): 12-25.

[12] D. Binkley. "Source code analysis: A road map." *2007 Future of Software Engineering*. IEEE Computer Society, 2007.

[13] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning.” *Nature* 521.7553 (2015): 436-444.

[14] P. Smolensky. *Information processing in dynamical systems: Foundations of harmony theory.* No. CU-CS-321-86. Colorado Univ at Boulder Dept of Computer Science, 1986.

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

[16] T. Tieleman. “Training restricted Boltzmann machines using approximations to the likelihood gradient.” *Proceedings of the 25th International Conference on Machine learning (ICML)*. ACM, 2008.

[17] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. “Extracting and composing robust features with denoising autoencoders.” *Proceedings of the 25th International Conference on Machine learning (ICML)*. ACM, 2008.

[18] G. E. Hinton “Training products of experts by minimizing contrastive divergence.” *Neural computation* 14.8 (2002): 1771-1800.

[19] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult.” *IEEE Transactions on Neural Networks,* 5.2 (1994): 157-166.

[20] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. “Greedy layer-wise training of deep networks.” *Advances in neural information processing systems* 19 (2007): 153.

[21] Y. Bengio. “Learning deep architectures for AI.” *Foundations and trends in Machine Learning* 2.1 (2009): 1-127.

[22] Y. Yang, H. Liu, J. Carbonell, and W. Ma. "Concept graph learning from educational data." *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. ACM, 2015.

[23] P. Yanardag, and S. V. N. Vishwanathan. "Deep graph kernels." *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015.

[24] S. Cao, W. Lu, and Q. Xu. "Deep Neural Networks for Learning Graph Representations." *AAAI*. 2016.

[25] T. D. Bui, S. Ravi, and V. Ramavajjala. "Neural Graph Machines: Learning Neural Networks Using Graphs." *arXiv preprint arXiv:1703.04818*(2017).

[26] S. Somanchi, and D. B. Neill. "Graph Structure Learning from Unlabeled Data for Early Outbreak Detection." *IEEE Intelligent Systems* 32.2 (2017): 80-84.

[27] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio “*How to construct deep recurrent neural networks.*” arXiv preprint arXiv:1312.6026 (2013).

[28] M. Hermans, and B. Schrauwen. “Training and analysing deep recurrent neural networks.” *Advances in Neural Information Processing Systems.* 2013.

[29] A. Graves, A. R. Mohamed, and G. Hinton. “Speech recognition with deep recurrent neural networks.” *IEEE International Conference on Acoustics, Speech and Signal Processing.* IEEE, 2013.

[30] H. Sak, A. W. Senior, and F. Beaufays. “Long short-term memory recurrent neural network architectures for large scale acoustic modeling.” *INTERSPEECH.* 2014.

[31] O. Irsoy, and C. Cardie. “Opinion Mining with Deep Recurrent Neural Networks.” *EMNLP*. 2014.

[32] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. "DRAW: a recurrent neural network for image generation." *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*. 2015.

[33] O. Nasraoui, M. Soliman, E. Saka, A. Badia, and R. Germain. "A web usage mining framework for mining evolving user profiles in dynamic web sites." *IEEE Transactions on Knowledge and Data Engineering* 20.2 (2008): 202-215.

[34] Y. Koren, R. Bell, and C. Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009).

[35] N. P. Whitehead, W. T. Scherer, and M. C. Smith. "Systems thinking about systems thinking a proposal for a common language." *IEEE Systems Journal* 9.4 (2015): 1117-1128.

[36] M. F. Valstar, and M. Pantic. "Fully automatic recognition of the temporal phases of facial actions." *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.1 (2012): 28-43.

[37] P. Rodriguez, et al. "Deep Pain: Exploiting Long Short-Term Memory Networks for Facial Expression Classification." *IEEE Transactions on Cybernetics* (2017).