Learning Logic Program Representation for Delayed Systems With Limited Training Data

Yin Jun Phua[1], Tony Ribeiro[3], Sophie Tourret[2], and Katsumi Inoue[2,1]

[1] Tokyo Institute of Technology, Japan
{phua,inoue}@il.c.titech.ac.jp
[2] National Institute of Informatics, Japan
{tourret,inoue}@nii.ac.jp
[3] Laboratoire des Sciences du Numrique de Nantes, France
tony.ribeiro@ls2n.fr

Abstract. Understanding the influences between components of dynamical systems such as biological networks, cellular automata and social systems provides insights to their dynamics. Such influences of dynamical systems are represented by logic programs with delays. In this paper, we present a method for representation learning to distinguish different dynamical systems with delays based on Recurrent Neural Network (RNN). This method relies on Long Short-Term Memory (LSTM) to extract and encode features from input sequences of time series data. We show that the produced high dimensional encoding can be used to distinguish different dynamical systems and reproduce their specific behaviors.

Keywords: dynamical systems, Boolean networks, attractors, learning from interpretation transition, delayed systems

1 Introduction

Learning from Interpretation Transition (LFIT) [6] is an unsupervised learning algorithm which learns the dynamics of an environment just by observing state transitions. Applications for such learning algorithms can range from multi-agent systems, where learning other agents' behavior can be crucial for decision making, to systems biology, where knowing the interaction between genes can greatly help in the creation of drugs to treat sickness [10]. This paper introduces an algorithm based on Recurrent Neural Network (RNN) that performs LFIT. The proposed approach outputs high dimensional matrix representation of logic programs that describe the dynamics of various Boolean systems. In this paper, we show that the learned matrix representation is equivalent to the Normal Logic Program (NLP) that can be used to describe these dynamics. This paper extends an ongoing work [7] with new experimental results. By relying on neural networks, we are able to perform LFIT on noisy and continuous data, where traditional approaches [8] cannot be applied with the exception of [11]. Previous approaches involving neural networks with logic programming [3, 2] attempts to construct neural networks in a way that logical semantics can be mapped

into the network. However these approaches cannot be applied in a dynamical environment. Another approach on using neural networks in inductive logic programming [4] involved training the neural network to model the dynamics of systems from a sufficient amount of measurements. In most practical cases, especially in biological systems, sufficient training data cannot be obtained to rely on these methods. Thus, having a method that achieves high performance with a small amount of training data is of great importance. This is made possible because the neural network used in this paper is not trained to model the dynamical system, but rather to output a classification of different systems and can be trained on artificial data before being used on the real data.

The rest of the paper is organized as follows. We first cover the logical and neural background that is required to understand this paper. Then we present the RNN-LFIT approach. We pursue by presenting an experimental evaluation demonstrating the validity of an approach before concluding the paper.

2 Background

The main goal of LFIT is to learn a normal logic program (NLP) describing the dynamics of the observed system. Most existing approaches are purely logical. Some can learn systems with delays (LFkT) [9]. Only one of the past LFIT approaches, not handling delays, relies on a neural network [4]. In all cases, the usual terminology is used to refer to normal logic programs (NLP). The rules in the NLP are of the form $R = A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge A_{m+1} \cdots \wedge A_n$ where $0 \leq m \leq n$. The inputs are transitions $x(t) \rightarrow x(t+1)$ from a given time step t to the following one where x is a vector containing all the observed variables of the system. In the case of Markov(k) systems (i.e. systems with delayed effects of at most k time steps), the inputs are sequences $x(t-k), \ldots, x(t) \rightarrow x(t+1)$. The focus of this paper is the learning of Markov(k) systems.

The neural network that we use to learn the system's dynamics is Long Short-Term Memory (LSTM). LSTM is a form of Recurrent Neural Network (RNN) that, contrary to earlier RNNs, can learn long term dependencies and do not suffer from the vanishing gradient problem. It has been popular in many sequence to sequence mapping application such as machine translation [12]. An LSTM consists of a memory cell for each time step, and each memory cell has an input gate i_t, an output gate o_t and a forget gate f_t. When a sequence of n_X time steps $X = \{x_1, x_2, \ldots, x_{n_X}\}$ is given as input, LSTM calculates the following for each time step:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot l_t$$
$$h_t = o_t \cdot c_t$$

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ l_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \cdot \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

where W is a weight matrix, h_t is the output of each memory cell, c_t is the hidden state of each memory cell and l_t is the input to each memory cell. σ is the sigmoid function.
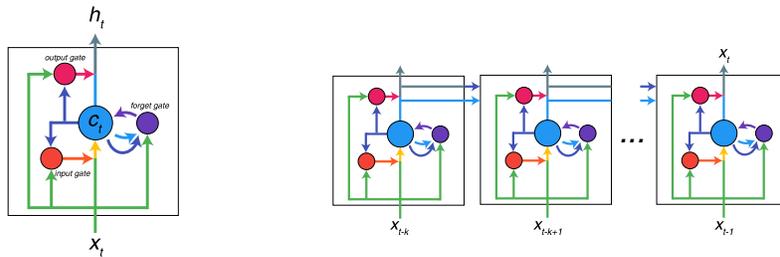
(a) An LSTM memory cell   (b) Unfolding of an LSTM network for BPTT training

Fig. 1: Long Short Term Memory

The input gate decides how much of the input influences the hidden state. The forget gate decides how much of the past hidden state influences the current hidden state. The output gate is responsbile for deciding how much of the current hidden state influences the output. A visual illustration of a single LSTM memory cell is shown in Figure 1(a).

LSTM networks can be trained by using backpropagation through time (BPTT) [5]. In BPTT, the LSTM is trained by unfolding across time steps, and then performing gradient descent to update the weights, as illustrated in Figure 1(b).

3   Model

In this section, we propose an architecture for performing LFIT. It consists of an encoder and decoder for the state transitions, and a neural network for performing LFIT. A visualization of the architecture is shown in Figure 2.

The autoencoder for the input sequences is responsible for encoding discrete time series into a feature vector that can later be manipulated by the neural network. This sequence of vectors is then encoded into one feature vector of dimension $2 \times k \times l$, where k denotes the number of memory cell units in the autoencoder LSTM, l denotes the number of LSTM layers, and this amount is doubled because both c and h, which represents the state of the memory cell, are considered.

The LSTM network performs LFIT, meaning that it takes as input the state transitions and an initial program encoding and outputs a program encoding that is consistent with the observations. The produced output is the representation of the normal logic program. The observations are the same input sequence as that given to the autoencoder, and in this work, the initial program is always set to $\emptyset$ and the LSTM network is trained to produce the complete normal logic program representation.

The goal of the architecture is to produce an encoding of past states, and an encoding of a normal logic program, that can then be multiplied together to predict the next state transition. This multiplication is a matrix $\times$ vector multiplication and produces a vector of $\mathbb{R}^n$ where n is the number of features in
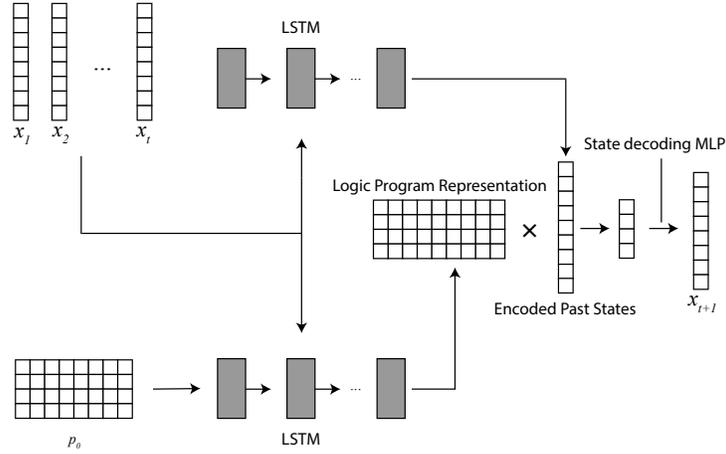
Fig. 2: A visualization of the proposed neural network models.

the logic program representation. This can be thought of as performing the T_p operator within linear geometric space. A multi-layered perceptron (MLP) then decodes this vector into the desired boolean state vector.

With the encoding of the state transition and an initial program, the LFIT network learns to produce an encoded program based on the observed state transitions. This encoded program can then be used for prediction, and in future work we plan to decode it into a normal logic programs thus making it possible to reason with it.

4   Evaluation

We applied our model to learn the dynamics of Boolean networks from continuous time series (please refer to the Appendix for more details). The Boolean network used in this experiments is taken from Dubrova and Teslenko [1] and represent the cell cycle regulation of mammalians. The Boolean network is first encoded as a logic program. Each dataset represents a time series generated from an initial state vector of continuous values. This initial state vector is generated by giving each of the 10 variables a random value between 0 and 1. Generated states are then mapped back to real values: 0 becomes $0.25 + \epsilon$ and 1 becomes $0.75 + \epsilon$, where ϵ simulates the measurement noise. The time series generated are then used to test the predictions of the proposed model. Table 1 shows the accuracy of the prediction made by this model. The accuracy is taken by calculating the mean-squared error (MSE) between the predicted state and the true subsequent state.

Figure 3 shows the graph of the learned representation for different NLPs based on principal component analysis (PCA). PCA is a popular technique for

Dataset	MSE (Original)	MSE (Noisy)
1	0.14	0.20
2	0.19	0.20
3	0.20	0.15
4	0.19	0.14
5	0.19	0.20

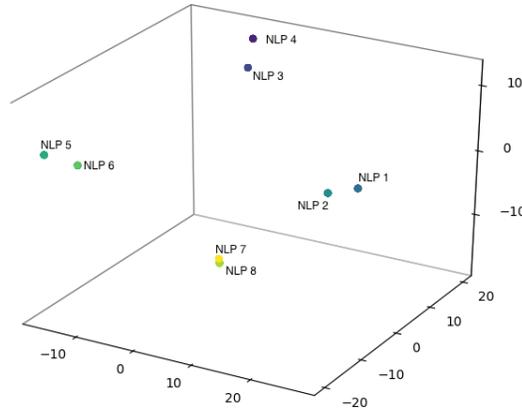Table 1: Results of the MSE of the prediction made by the proposed model on various datasets

Fig. 3: PCA plot of the learned representation for NLPs based on input time series

visualizing high dimensional embeddings. The logic representation obtained from our model is a 4×128 matrix, by using PCA and extracting 3 of the dimensions that separate the data the most, we can obtain the graph shown in the figure. Each dot in the graph are representations learned separately from various state transitions from the logic program. Note that learned representations that are from different logic programs are clearly separated, and those that are from the same logic programs converge to a single dot.

We observed that the model was very good at identifying the dynamics of the system based only on a sequence of state transitions, however more work has to be done on the prediction.

5 Conclusion and Future Work

In this paper we propose a method for learning a matrix representation of dynamical systems with delays. One of the interesting aspects of this work is that we produced a logic program representation in matrix form, which when multiplied with a feature vector of the past states, is able to produce a vector that

represents the predicted state. This could lead to future works such as reasoning and performing induction purely in the algebraic space.

The main contribution of this work is to devise a method of modeling systems where only limited amounts of data can be collected. Without sufficient amount of data, purely logical methods cannot provide useful information, and attempts at training neural networks to model the system will result in overfitting. Therefore we speculate that generating artificial data in order to train a more generalized neural network may be a more successful approach in such cases.

We are also planning to decode the NLP representation into logical form to allow humans to reason with it. Also, providing a partial program as background knowledge to the network is left for future work.

References

1. Dubrova, E., Teslenko, M.: A sat-based algorithm for finding attractors in synchronous boolean networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 8(5), 1393–1399 (2011)
2. d'Avila Garcez, A.S., Broda, K., Gabbay, D.M.: Symbolic knowledge extraction from trained neural networks: A sound approach. Artificial Intelligence 125(1), 155–207 (2001)
3. d'Avila Garcez, A.S., Zaverucha, G.: The connectionist inductive learning and logic programming system. Applied Intelligence 11(1), 59–77 (1999)
4. Gentet, E., Tourret, S., Inoue, K.: Learning from interpretation transition using feed-forward neural network. In: ILP 2016. pp. 27–33 (2016)
5. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks 18(5), 602–610 (2005)
6. Inoue, K., Ribeiro, T., Sakama, C.: Learning from interpretation transition. Machine Learning 94(1), 51–79 (2014)
7. Phua, Y.J., Tourret, S., Katsumi, I.: Learning logic program representation from delayed interpretation transition using recurrent neural networks. In: SNL 2017 (2017)
8. Ribeiro, T., Inoue, K., Sakama, C.: A BDD-based algorithm for learning from interpretation transition. In: ILP 2013. pp. 47–63. Springer (2013)
9. Ribeiro, T., Magnin, M., Inoue, K., Sakama, C.: Learning delayed influences of biological systems. Frontiers in Bioengineering and Biotechnology 2, 81 (2015)
10. Ribeiro, T., Magnin, M., Inoue, K., Sakama, C.: Learning multi-valued biological models with delayed influence from time-series observations. In: 14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, Miami, FL, USA, December 9-11, 2015. pp. 25–31 (2015)
11. Ribeiro, T., Tourret, S., Folschette, M., Magnin, M., Borzacchiello, D., Chinesta, F., Roux, O., Inoue, K.: Inductive learning from state transitions over continuous domains. In: Proceedings of ILP 2017, to appear. Springer (2017)
12. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. pp. 3104–3112 (2014)

A   Appendix

A.1   Model details

In this section we provide the parameters and details of the neural network model that we used to perform the experiments. See the main text for high-level summary of all model components.

Autoencoder model The state transition autoencoder takes a series of 10 state transitions, where each state is a 10 dimensional vector which represents the state of each variables within the system. The autoencoder LSTM model we trained have 2 layers, each with 512 memory cell units. The produced state representation is then multiplied by a $(2 \times 2 \times 512, 128)$ matrix, to produce a 128 dimension feature vector that represents the series of state transitions.

LFIT model The LFIT model takes the same input as the encoder model, but the LSTM model have 4 layers, which is the dimension of the resulting feature vector for the predicted state, and has 1,024 hidden units which is twice the number of hidden units of the autoencoder model. The produced logic program representation is then transformed into $(4, 128)$ matrix by multiplying with a $(2 \times 4 \times 1024, 4 \times 128)$ matrix and then reshaping.

Decoder model The decoder model takes the resulting feature vector for the predicted state, which is a vector of 4 dimension, and outputs a vector of 10 dimension with each dimension representing the state of the variables. The decoder model consists of a MLP with 2 hidden layers, and each layer has 32 hidden units. Each hidden layer is activated by ReLU (Rectified Linear Unit), which is a function that outputs 0 for all input less than 0, and is linear when the input is larger than 0.

A.2   Training details

We used the following training parameters for our experiment:

- Training steps: 10^4
- Batch size: 2
- Gradient descent optimizer: Adam, learning rate and various other parameters are left with the defaults for Tensorflow r1.2
- Dropout: probability of 0.3 per training step