3O1-1

リカレントニューラルネットワークによる遅延を伴う解釈遷移からの論理プログラム表現学習

Learning Logic Program Representation from Delayed Interpretation Transition using Recurrent Neural Networks

ポア インジュン *1 Yin Jun Phua　トゥレ ソフィ *2 Sophie Tourret　井上 克巳 *1*2 Katsumi Inoue

*1東京工業大学 Tokyo Institute of Technology　*2国立情報学研究所 National Institute of Informatics

Having a method to understand the interactions and delayed influences between components of dynamical systems can provide useful applications to biological and other dynamical systems. In this paper, we present a method relying on Recurrent Neural Networks (RNN) that can learn to distinguish the nature of different systems. This method utilizes Long Short-Term Memory (LSTM) to extract and encode certain features from the input sequence of time-series data. We also show that the produced high dimensional encoding can be used to represent different time series that are resulted from the same dynamical system.

1. はじめに

解釈遷移からの学習 (learning from interpretation transition; **LFIT**) は，あるダイナミック環境から得られた観測データから系のダイナミクスを自動的に学習する教師なし学習方式である．こうした状態変移に関する関係ダイナミクスを時系列データから学習することで，遺伝子調節ネットワーク学習，行動パターン学習，アクション規則学習などへの応用に繋がる．本論文ではリカレントニューラルネットワーク (recurrent neural network; **RNN**) を用いた **LFIT** アルゴリズムを提案する．提案手法は任意のダイナミック環境からその背後に存在すると考えられるルールの高次元ベクトル表現を自動的に学習し，その表現が標準論理プログラム (normal logic program; **NLP**) に相当することを示す．提案手法は [1] で提案されたフィードフォワードニューラルネットワークを用いた 1 ステップ遷移の学習を行う手法を拡張し，遅延の存在するシステムでの学習を行うものである．様々な予測学習や関数近似問題で高い性能を示したニューラルネットワーク (neural network; **NN**) を用いることで従来の論理的手法 [2] ではノイズや連続値変数に対応できないという欠点の解決が期待される．また従来の **NN** を帰納的論理プログラミングへの応用と異なり，**NN** にダイナミックシステムをモデル化するように学習を行うのではなく，**NN** にそのダイナミックシステムの高次元ベクトル表現を学習させる手法を提案する．

RNN にダイナミックシステムをモデル化できるように学習させる手法も提案されているが [6]，変数が増えるとモデルのパラメータが増え，性能を保つために学習に必要なデータ量も増える．しかし [6] では遺伝子制御ネットワークへの応用を前提とし，学習データが不足する状況で性能を高めることが必要である．本論文で提案する手法は学習データの量に頼らず，性能が保たれることを示す．また，本研究は，注目を浴びつつある記号表現とニューラル表現をつなぐ学習研究の一つであり，その中では遅延付き関係ダイナミクスを学習する初の試みである．

2. LFIT

一階述語論理でのエルブラン基底 $\mathcal{B}$ を考えた時，**NLP** は

$$A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n \qquad (1)$$

の形式をとったルールの集合である．ここで A 及び A_i はアトムである $(n \geq m \geq 0)$．任意の (1) の形式をとるルール R において，A は R のヘッドであり，$h(R)$ で表す．そして，$\leftarrow$ の右辺の連言は R のボディである．R のボディで現れるリテラルの集合を $b(R) = \{A_1, \ldots, A_m, \neg A_{m+1}, \ldots, \neg A_n\}$ で表し，R のボディで正リテラルとして現れるアトムを $b^+(R) = \{A_1, \ldots, A_m\}$，負リテラルを $b^-(R) = \{A_{m+1}, \ldots, A_n\}$ として表す．

エルブラン解釈 I は $\mathcal{B}$ の部分集合である．論理プログラム P 及びエルブラン解釈 I を考える時，T_P オペレータは $T_P : 2^{\mathcal{B}} \rightarrow 2^{\mathcal{B}}$ の対応である．この時 T_P オペレータは

$$T_P(I) = \{h(R) \mid R \in P, b^+(R) \subseteq I, b^-(R) \cap I = \emptyset\}$$

として定義できる．

解釈遷移の集合 E が与えられた時，**LFIT** アルゴリズムは E のダイナミクスを表現できる論理プログラム P を学習し出力する．その中で 1 ステップ遷移のみを考える **LFIT** アルゴリズムは **LF1T** と呼び，3 つの論理的アルゴリズムが開発されている [2] [3] [4].

2.1 LFkT

LF1T を Markov(k) システムに拡張し，k ステップまでの遷移を考える学習を **LFkT** と呼ぶ．時間を考慮した時，周期 k を持つ論理プログラム P のエルブラン基底は

$$\mathcal{B}_k = \bigcup_{i=1}^{k} \{v_{t-i} \mid v \in \mathcal{B}\}$$

として書ける．ある Markov(k) システム S を考えた時，S の全ての変数を $\mathcal{B}$ として，S のルール R を $h(R) \in \mathcal{B}$，$b(R) \in \mathcal{B}_k$ とすれば S は論理プログラムとして表すことができる．S の実行トレース T は S の状態の有限系列として考える．この時，実行トレースは $T := (S_0, \ldots, S_n), n \geq 1, S_i \in 2^{\mathcal{B}}$ として書ける．k ステップの解釈遷移はエルブラン解釈 (I, J) として表し，$I \subseteq \mathcal{B}_k, J \subseteq \mathcal{B}$ である．**LFkT** の論理的手法を用いたアルゴリズムも開発されている [5].

論理的手法を用いたアルゴリズムは，観測データが実数値の場合離散化する必要があり，その離散化の方法が間違っていたらそれ以降全てのステップも間違えてしまうので，現実的な

応用が困難である．**NN** を用いることで，その離散化するステップを省けると考えられている [1]．また，あるダイナミックシステムを学習したい時，そのシステムから生じるあらゆる状態遷移が得られるとは限らない．論理的手法では入力で与えられた全ての状態遷移に対して一貫する **NLP** を出力するが，それ以外の状態遷移への汎化ができない．一方，汎化性に強い **NN** を用いたアルゴリズムは入力で与えられていない状態遷移への汎化もできるので実用性も期待できる [1]．

3. RNN

RNN はフィードフォーワードニューラルネットワークを系列データ扱うために拡張したものである．系列データの入力 $(x_1, \ldots, x_T)$ が与えられた時，3 つの重み行列 W^{hx}, W^{hh}, W^{yh} 及び 3 つのバイアス項ベクトル b_h, b_y, h_0 を用いて，標準 **RNN** は

$$h_t = \sigma(W^{hx}x_t + W^{hh}h_{t-1} + b_h)$$
$$y_t = W^{yh}h_t + b_y$$

を時間ステップごとに計算し $(y_1, \ldots, y_T)$ を出力するものである．h_t は各時間ステップの隠れ状態である．ここで σ はシグモイド関数である．

しかし通常の **RNN** は長期的な依存性が存在する系列データの学習においてはとても困難である [7]．また勾配降下法を用いて学習する時に伝搬消失問題もある [7]，これに対して，Long Short-Term Memory (**LSTM**) は長期的な依存性が存在する問題も扱うことができるので，**LSTM** を用いて **LFIT** を行うことにした．

本論文では [8] で提案された **LSTM** モデルを用いる．系列データの入力 $X = \{x_1, x_2, \ldots, x_{n_X}\}$ が与えられた時，LSTM は各時間ステップに入力ゲート i_t，出力ゲート o_t 及び忘却ゲート f_t を与え，一つのメモリセルを形成する．[8] の記号に従い，各メモリセルの出力を h_t，各メモリセルへの入力を l_t，隠れ状態を c_t として表すと，その出力 h_t は

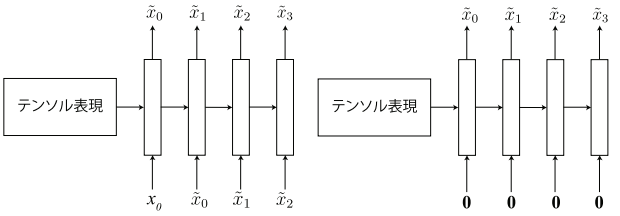
$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ l_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \cdot \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot l_t$$
$$h_t = o_t \cdot c_t$$

として表すことができる．

4. 論理プログラム表現学習の方法

提案手法では 3 つの **LSTM** モデルを用いて学習を行う．その 3 つの **LSTM** モデルの役割は主にエンコーダー，デコーダー及び予測モデルに分けられる．エンコーダーは入力の時系列データを求めたい論理プログラムの表現に変換する，デコーダーはエンコーダーが出力した論理プログラムの表現から元の時系列データに復元する．予測モデルは論理プログラムの表現及び入力時系列データの最後の時間ステップからその次の時間ステップを予測する．

ある時間ステップ t の状態をベクトル $x_t \in \mathbb{R}^{|\mathcal{B}|}$ で表現する．時間 t でリテラル l_i が真である時，$x_t^{(i)} = 1$ とし，l_i が



(a) 条件付き　(b) 条件なし

図 1: 条件付き及び条件なしデコーダー LSTM の構造

負である時は $x_t^{(i)} = 0$ とする．ここで $x^{(i)}$ はベクトル x の i 番目の要素である．この時，時系列データ X は，ベクトルのシーケンス $(x_1, x_2, \ldots, x_M)$ として表現する．

M ステップの時系列データ $X = (x_0, x_1, \ldots, x_M)$ が与えられた時，n 個のノードを含む m 個の隠れ層で構成した LSTM のエンコーダーを考えた時，エンコーダーの基本的な方程式は以下のように表せる：

$$T = \text{EncoderRNN}(x_0, x_1, \ldots, x_M)$$

得られた T はベクトルよりも階数が高い線形量であり，ベクトルや行列を一般化したものをテンソルとして呼ぶ．一般的に階数 1 のテンソルをベクトルとして呼び，階数 2 のテンソルを行列として呼ぶ．ここで T は階数 2 の $m \times n$ 次元のテンソルである．**LFIT** を行う時，エンコーダーへの入力の時系列データの長さ M は最大遅延 k より大きくなければならない．

元の時系列データを復元したい時，その論理プログラムがあれば初期状態から T_P オペレーターを M 回まで計算すればよい．なのでデコーダーを T_P オペレーターの計算をする関数だとして，T が論理プログラムを表現すると考える時，デコーダーは以下のように表せる：

$$\tilde{X} = \text{DecoderRNN}(x_0, T)$$

$\tilde{X} = (\tilde{x}_0, \tilde{x}_1, \ldots, \tilde{x}_M) \approx X$ である．デコーダーの実装は **LSTM** 及びその出力が $[0, 1]$ となるようにシグモイド関数を加えたものである．x_0 を初期状態として，出力で $\tilde{x}_0, \tilde{x}_1, \ldots, \tilde{x}_M$ が求められてるのでデコーダーは厳密的に T_P オペレーターの計算ではないことに注意されたい．デコーダーの構造を考える時，各時間ステップのメモリセルに入力を与える場合と与えない場合が考えられる．前の時間ステップでの出力をメモリセルへの入力とした時，このような構造を条件付きモデルとして呼び，各時間ステップでメモリセルへの入力を 0 とした時を条件なしモデルとして呼ぶ．図 1a にデコーダーの条件付きモデルを示し，図 1b に条件なしモデルを示す．

一方予測モデルでは，x_M を与えて $y_{M+1}, y_{M+2}, \ldots y_N$ までの状態を求めたい．この時予測モデルは以下のように表せる：

$$Y = \text{PredictorRNN}(x_M, T)$$

$Y = (y_{M+1}, y_{M+2}, \ldots, y_N)$ である．予測モデルもデコーダー同様，条件付き及び条件なしモデルが考えられる．

エンコーダー，デコーダー及び予測モデルを組み合わせ一つのネットワークにしたものを複合モデルとして呼ぶ．デコーダー及び予測モデルともに条件つきである場合は条件付き複合モデルと呼び，デコーダー及び予測モデルともに条件なしである場合は条件なし複合モデルとして呼ぶ．その構造を図 2 に示す．

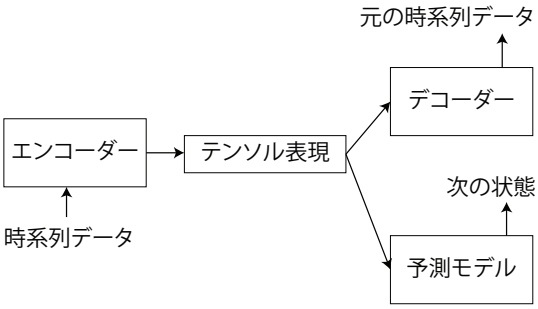


図 2: 複合モデルの構造

5. 実験

使用されたデータセットは，全てランダムに生成された物である．生成の方法は，まず変数 n 個の **NLP** のルールをランダムに生成し，そしてランダムに初期状態を設定し，生成された **NLP** に従って M ステップの遷移まで生成する．この時，初期状態を含む最初の M 遷移は **LSTM** モデルの入力となり，$M+3$ ステップの遷移は予測モデルのラベルとして学習を行った．

ここでは，元の入力を生成するエンコーダーと次の時間ステップの状態を予測する予測モデルを学習させることを目的とする．そのために，目的関数をある入力系列 S に対して，復元した入力 $\tilde{S}$ と次の遷移状態 T を出力する確率の最大化を計りたい．

$$1/|S| \sum_{(T,\tilde{S},S)\in\mathcal{S}} \log p(T,\tilde{S}|S)$$

ここで $\mathcal{S}$ は学習データの集合である．

実験を行う中で，深層 **LSTM** の方が性能が良いことがわかったので，以下の実験では隠れ層 5 層の **LSTM** を用いて行っているものである．深層 **LSTM** の方が性能が良いのは，隠れ層が多い時ほど表現の自由度が高いからだと考えられる [9].

- LSTM のパラメーターを全て平均 0.0 分散 1.0 の正規分布に従って初期化した；
- 学習は Adam 法で勾配降下法を行う時のモーメントや学習係数などを決めている [11]；
- 学習の過程ではバッチ数 64 で学習させた．

5.1 評価方法

本研究では，予測モデルの最初の出力，つまり y_{M+1} について評価する．学習データの遷移状態が p とすると，y_{M+1} の各要素について，

$$r^{(i)} = \begin{cases} 1, & y^{(i)}_{M+1} \geq \alpha \text{の時}, \\ 0, & \text{それ以外}. \end{cases} \tag{2}$$

の予測状態ベクトル r_{M+1} を作る．予測状態ベクトルの各要素は真陽性 (True Positive; TP)，偽陽性 (False Positive; FP)，真陰性 (True Negative; TN)，偽陰性 (False Negative; FN) に分けられる．それぞれの定義を表 1 に示す．

表 1: 予測状態の分類とその定義

TP	$r^{(i)}=1$ かつ $p^{(i)}=1$
TN	$r^{(i)}=0$ かつ $p^{(i)}=0$
FP	$r^{(i)}=1$ かつ $p^{(i)}=0$
FN	$r^{(i)}=0$ かつ $p^{(i)}=1$

表 2: 提案モデルの各手法とその精度

手法	精度
デコーダーなし	0.80
条件なし複合モデル	0.84
条件付き複合モデル	0.82

各手法や実験結果を評価するには，精度 (Accuracy; ACC) という尺度を用いる．精度は全ての結果から正しく予測できた比率と定義する．その数学的定義は式 (3) に示す．

$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{FN}+\mathrm{TN}} \tag{3}$$

以上に述べた評価手法で，まずは 14 変数の最大遅延 5 の異なるモデルについて調べてみた．そして，最大遅延を増やし，その時のモデルの精度の評価も調べた．最後に，変数の数を増やして問題の大きさについて評価をした．

学習及び実験は Tensorflow v0.12.0[12] を用いて実装した．

5.2 実験結果

まず各手法の実験結果を表 2 に示す．各手法共に 14 変数，最大遅延 $k=5$，メモリセル数が 6 の 5 層 LSTM 構造で，同一のデータセットから学習を行った．そのデータセットは 4 つの論理プログラムによって生成された 8,192 個の系列データである．評価時における α は 0.3 に設定した．テストのデータセットは学習時のデータセットと異なる 4 つの論理プログラムから 8,192 個の系列データを生成した．結果から見られるように，デコーダーなしでの学習は精度が落ちてしまい，条件付きよりも条件なしのモデルがよい性能を示している．

次に，時系列データの最大遅延を増やした時に，本研究で提案したモデルの精度が保たれているかを調べた．ここも変数の数が 14 個である，しかしそれぞれ異なっている論理プログラムにより生成されたデータセットで学習を行った．学習のデータセットは 4 つの論理プログラムによって生成された 8,192 個の系列データである．この実験結果を表 3 に示す．システムが取り得る最大遅延と精度には強い関係性は見られないが，これは遅延を増やしても精度の減少が起きないことと解釈することもできる．

[5] では変数が増えると計算量が指数関数的に増えるので，本研究で提案したモデルで変数の数を増やした時の精度について調べた．ここで生成されたデータセットでの最大遅延は $k=5$ とし，各実験では 4 つの異なる論理プログラムによって生成された 8,192 個の時系列データについて学習を行った．その結果を表 4 に示す．結果から，変数の数が増えても精度がほぼ一定であることが明らかである．

5.3 考察

図 3 にいくつかの LSTM が学習した表現を線形判別分析 (linear discriminant analysis; **LDA**) で 2 次元の座標に射影したものである．同じ論理プログラムによって生成された時系列データを読み込んだ後の隠れ状態は，明らかに分類できる．

表 3: 条件なしモデルでデータの最大遅延を変更した時の精度

最大遅延 k	入力系列の長さ	精度
6	8	0.83
7	8	0.87
8	10	0.85
9	10	0.80
10	12	0.83

表 4: 条件なしモデルで変数の数を変えた時の精度

変数の数 $\|\mathcal{B}\|$	精度
15	0.85
20	0.84
25	0.85
30	0.84

この分類を用いて，その特徴量について分析することで人間にも理解できるルール形式の学習に繋がると考えられる．

6. おわりに

本研究は，記号表現とニューラル表現をつなぐ学習研究を目指した．遅延を持つ時系列データからその背後に支配するルールを学習するために，オートエンコーダーモデルの **LSTM** を実装し，そのテンソル表現から入力の時系列データの特性を捉えたことを実証してきた．

従来とは異なり，調べたいネットワークについて **RNN** に学習させてモデル化させるのではなく，類似のネットワークを生成して分類問題に変換することで，**RNN** の大量の学習データがないと性能が上がらない欠点を解決した．調べたいネットワークの規模や特性に応じて，学習データを生成しモデルを学習させれば良い性能が得られることを示した．

時系列データからその系列を表現できるテンソル表現を得られることは，ダイナミック環境を理解する上では大きいな一つのステップとして考えられる．本研究の今後の展開では，そのようなテンソル表現を実際に論理プログラムに変換する．現状では，別の **RNN** を使うことで生成することに取り組んでおり，部分的にその有効性を確認している．

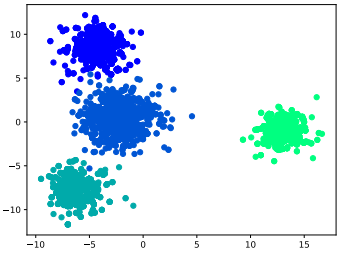


図 3: LDA を用いて LSTM が時系列データを読み込んだあとの隠れ状態を 2 次元に射影したグラフ．異なる色は異なる論理プログラムによって生成された時系列データを意味し，このグラフから異なる論理プログラムごとにクラスタリングが明確にされていることに注意されたい．

参考文献

[1] Enguerrand Gentet, Sophie Tourret, and Katsumi Inoue. Learning from Interpretation Transition using Feed-Forward Neural Network. Inductive Logic Programming, 2016.

[2] Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from Interpretation Transition. Machine Learning, Vol. 94(1), pp. 5179, 2014.

[3] Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. A BDD-Based Algorithm for Learning from Interpretation Transition. Inductive Logic Programming: Revised Selected Papers from the 23rd International Conference, Vol. 8812, pp. 4763, 2014.

[4] Katsumi Inoue and Tony Ribeiro. Learning Prime Implicant Conditions from Interpretaion Transition. Inductive Logic Programming: Revised Selected Papers from the 24th International Conference, Vol. 9046, pp. 108125, 2015.

[5] Tony Riberio, Morgan Magnin, Katsumi Inoue, and Chiaki Sakama. Learning delayed influences of biological systems. Frontiers in Bioengineering and Biotechnology, Vol. 2, p. 81, 2014.

[6] Abhinandan Khan, Sudip Mandal, Rajat Kumar Pal, and Goutam Saha. Construction of gene regulatory networks using recurrent neural networks and swarm intelligence. Scientifica, Vol. 2016.

[7] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. Trans. Neur. Netw., Vol. 5, No. 2, pp. 157166, 1994.

[8] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. Neural Comput., Vol. 9, No. 8, pp. 17351780, November 1997.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. CoRR, Vol. abs/1409.3215, 2014.

[10] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In David Blei and Francis Bach, editors, Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pp. 843852. JMLR Workshop and Conference Proceedings, 2015.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. CoRR, Vol. abs/1412.6980, 2014.

[12] Tensorflow. https://www.tensorflow.org/, 2017.