再帰を用いた深層学習による時系列データの学習

Deep learning for time series with recurrent neural network

水川 徳之 † 松原 崇 ‡ 上原 邦昭 ‡
Mizukawa Noriyuki Matsubara Takashi Uehara Kuniaki

1. はじめに

ニューラルネットワーク (Neural Network；NN) は様々なデータの分類・回帰・予測などに利用されるが，特に言語や音声などの時系列データを処理するためのものとして再帰ニューラルネットワーク (Reccurent Neural Network ; RNN) がある．通常の NN に比べて，RNN は時刻 t の隠れ層の値が時刻 t の入力データだけでなく，時刻 $t-1$ の隠れ層の値も用いて計算される．これによって，過去のデータを考慮した学習がなされる．学習は通常の NN と同様に，教師データを利用した誤差逆伝播法が用いられる [7]．しかしこの方法では，データが長くなるにつれて，計算時間が極端に長くなるといった欠点がある．また離れたデータ間では誤差が拡散してしまい，うまくデータの相関を得ることができないといった問題も挙げられる．

これらの問題を解決するために，RNN のユニットをデータの記憶に特化したノードに置き換えた Long-Short Term Memory (LSTM) が提案されている [5]．LSTM のノードは，入力・出力の経路に加えて，入力・出力の大きさをそれぞれ制御するゲート，内部データを保持・忘却させるゲートを備えている．これら複数のゲートの重みも学習することによって，時系列データの記憶に特化することができ，長い時系列での相関をもつデータの学習が可能となる．

RNN のよりシンプルな改良法として，RNN の係数行列の初期値に単位行列 (Identity matrix) を用いるという，新しい手法が挙げられる [6] (以降，この手法を IRNN と表記する)．乱数を初期値とする一般的な RNN に比べて，離れたデータ間での相関が得られやすく，さらに活性化関数に Rectified Linear Unit (ReLU)[8] を用いることによって，LSTM 同様に，長い時系列データをより保持しやすくなっている．IRNN は，LSTM に比べてシンプルな構造であるにもかかわらず，いくつかのテストで LSTM より高い精度が得られることが示されている [6]．

長い時系列データの処理の例として，プログラムコードを自然言語として処理するものが挙げられる．Zaremba ら [12] は LSTM を用いた RNN で，四則演算や条件分岐を含む Python コードを時系列データとして扱い，計算式の結果を予測する学習器を提案している．

本稿では，RNN，LSTM，IRNN それぞれの構造を確認しつつ，プログラムコードを LSTM ではなくシンプルな IRNN を用いて処理する方法を提示し，その性能を確認する．

2. 各ネットワークの構造と学習

RNN，LSTM，IRNN は，いずれも隠れ層の状態を次の時刻での処理にも用いるという特徴を持つ再帰ニューラルネットワークである．本章では各ネットワークの構造から，長い時系列データへの対応がどのように行われているかを確認する．

2.1 RNN

入力層，隠れ層，出力層からなる RNN の概要を図 1 に示す．左が入力層で，$\mathbf{x}^t$ が時刻 t での入力，$\mathbf{h}$ が一つ前の時刻での隠れ層の値で，それぞれ係数行列 $\mathbf{W}^{in}$，$\mathbf{W}^{hid}$ を通して，隠れ層に伝達される．中央が隠れ層で，隠れ層の値 $\mathbf{h}$ は $\mathbf{W}^{out}$ を通して出力層に伝達されるだけでなく，図中の破線矢印で示されているように，次の時刻の入力にも使われる．右が出力層で，出力値 $\mathbf{y}$ が得られる．

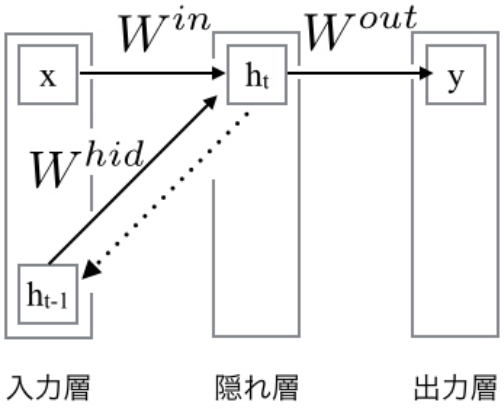

図 1: RNN のグラフィカルモデル

このように RNN では隠れ層の値を再帰的に利用するため，時刻 t での隠れ層の値 $\mathbf{h}^t$ は，時刻 t での入力

† 神戸大学 工学部 情報知能工学科
Faculty of Engineering, Kobe University
‡ 神戸大学 大学院 システム情報学研究科
Graduate School of System Informatics, Kobe University

の値 $\mathbf{x}^t$ および時刻 $t-1$ での隠れ層の値 $\mathbf{h}^{t-1}$ に依存する．これを踏まえて，時刻 1〜T に渡る時系列データ $\mathbf{X}=\{\mathbf{x}^1,...,\mathbf{x}^T\}$ が入力されたときの様子を図 2 に示す．

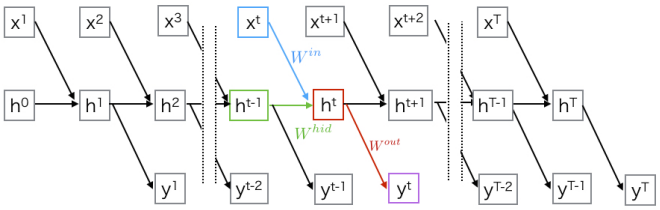

図 2: RNN への時系列データの入力

入力層，隠れ層，出力層の次元の添字をそれぞれ i, j(および j'), k とすると，値の伝達は次式のように表される．

$$h_j^t = f_u\left(u_j^t\right) \tag{1}$$
$$u_j^t = \sum_i W_{j,i}^{in,t} x_i^t + \sum_{j'} W_{j,j'}^{hid,t} h_j^{t-1} \tag{2}$$
$$y_k^t = f_v\left(v_k^t\right) \tag{3}$$
$$v_k^t = \sum_j W_{k,j}^{out,t} h_j^t \tag{4}$$

x_i はベクトル $\mathbf{x}$ の第 j 成分を，$W_{i,j}$ は行列 $\mathbf{W}$ の第 i,j 成分を表す．ただし $\forall j\{W_{j,0}=1\}$ で，バイアスパラメータは x_0, h_0 として含まれている．$\mathbf{u}^t, \mathbf{v}^t$ は時刻 t における隠れ層 $\mathbf{h}^t$ および出力層 $\mathbf{y}^t$ への入力，f_u, f_v は活性化関数で，sigmoid 関数や tanh が用いられる．

ネットワークの学習は，出力 $\mathbf{y}$ とそれに対応する教師信号 $\hat{\mathbf{y}}$ について何らかのコスト関数 $E(\mathbf{y},\hat{\mathbf{y}})$ を設け，コスト値 E が最小となるような各行列 $\mathbf{W}$ を求めることによって行われる．そのためにコスト関数 $E(\mathbf{y},\hat{\mathbf{y}})$ を $\mathbf{W}$ で微分し，次の勾配法を用いる．

$$\mathbf{W}_{new} = \mathbf{W}_{old} - \eta \frac{\partial E}{\partial \mathbf{W}} \tag{5}$$

ここで η は学習率を表す．初期値は適当な確率分布 (ガウス分布，一様分布など) に従う乱数とする [2]．微分に関しては，具体的に次の式が成り立つ [9]．

$$\frac{\partial E}{\partial u_j^t} = \left\{ \sum_{j'} W_{j,j'}^{hid,t+1} \delta_{j'}^{t+1} + \sum_k W_{k,j}^{out,t} \delta_k^t \right\} \frac{\partial f_u(u_j^t)}{\partial u_j^t}$$
$$\delta_{j'}^t = \frac{\partial E}{\partial u_{j'}^t}, \quad \delta_k^t = \frac{\partial E}{\partial v_k^t} \tag{6}$$

式 (6) は $\delta_{j'}^t$ について漸化式になっているので，$t=T$ から再帰的に計算できる．時系列にわたってすべての δ が計算されると，誤差逆伝播法により各行列 $\mathbf{W}$ を更新することができる．

このように，RNN を時系列長のニューラルネットワークに展開することで，通常のニューラルネットワークと同様の誤差逆伝播法を適用することができる [11]．しかし，活性化関数に sigmoid 関数や tanh を用いると，隠れ層の値が再帰的に加算されるため，隠れ層の出力が飽和しやすい．このため，長い時系列データを入力すると，離れたデータ間での相関が得られにくくなるという欠点がある．

2.2 LSTM

長い時系列データが RNN に不向きなのは，特定の時刻の入力データを保持することが難しいためである [3]．このため，隠れ層をデータの記憶に特化したユニットに置き換えた，Long-Short Term Memory (LSTM) が考案されている [5]．LSTM の概要を図 3 に示す．図中の赤い四角が RNN での隠れ層に相当する．RNN と比較すると，入力層からの入力 $\mathbf{x}^t$，および前の時刻の隠れ層からの入力 $\mathbf{h}^{t-1}$ の他に，3 つのゲートを制御するパラメータ $\mathbf{g}^{I,t}, \mathbf{g}^{F,t}, \mathbf{g}^{O,t}$ が加えられている．ゲートはそれぞれ，入力層からの入力値の大きさ，隠れ層内でのフィードバックの大きさ (つまり記憶)，出力層への出力値の大きさを制御する．たとえば，入力ゲート $\mathbf{g}^{I,t}$ と出力ゲート $\mathbf{g}^{O,t}$ が閉じられていて，記憶ゲート $\mathbf{g}^{F,t}$ が内部の値 $\mathbf{s}^t$ を一定に保つような場合，隠れ層内の値 $\mathbf{s}^t$ は入力によらず記憶され，その値は出力層に影響しなくなる．このようにして，任意の時刻だけ離れたデータを記憶し，相関を得ることができる．

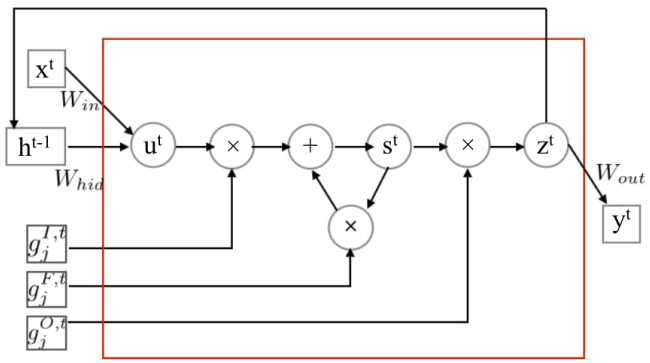

図 3: LSTM のグラフィカルモデル

LSTM は，RNN に比べて長い時系列データを扱えるが，パラメータが多くなり，その分学習も極端に遅くなる．ゲートの値は，入力層 $\mathbf{x}^t, \mathbf{h}^{t-1}$ および内部状態 $\mathbf{s}^t$ の重み付け線形和を活性化関数に通した出力値が用いられ，重み付けの係数も学習パラメータとして扱われる．すなわち LSTM では，隠れ層について，通常の RNN での入力層からの係数行列に加えて，3 つのゲートを制御する

係数行列も更新しなければならない．従って単純に隠れ層 1 個あたりでは，学習パラメータの数が 4 倍になる．

2.3 IRNN

LSTM のようにパラメータ数を増やすことなく，データを効率よく記憶しながら時系列データを処理する手法として，IRNN が提案されている [6]．RNN では初期値として適当な乱数を用いるが，IRNN では隠れ層を次の時刻に伝える係数行列 $\mathbf{W}^{hid}$ の初期値を単位行列とし，さらにバイアス項を 0 とする．これにより，隠れ層は変化することなく，値が次の時刻に伝えられる．さらに，この時入力データの係数を 0 とすると，入力データは隠れ層に影響せず，その間，隠れ層の値は記憶されることになる．これは LSTM において，入出力ゲートを閉じ，記憶ゲートで内部状態を保持することに相当する．IRNN のもう一つの特徴として，活性化関数に ReLU を用いる点が挙げられる．隠れ層の値が記憶されていて，さらに入力の係数行列が適当な重みを持つとき，各時刻において入力が隠れ層に加算されるが，隠れ層からの出力の活性化関数が sigmoid 関数や tanh の場合，次第に飽和していき，最終的には入力データの情報がほとんど得られなくなってしまう．一方，ReLU は入力が正数ならそのまま出力される (負数は 0 に打ち切られる) ので，過去の情報を記憶しつつ，新たな入力も適切に加味することができる．

行列内のバイアス項 $\mathbf{b}^{in}, \mathbf{b}^{hid}$ を明示的に書き出して，IRNN が隠れ層の値を記憶することを確認してみる．入力層および出力層からの出力 $\mathbf{u}^t = \{u_1^t, \ldots, u_j^t, \ldots\}, \mathbf{h}^t = \{h_1^t, \ldots, h_j^t, \ldots\}$ は，

$$\mathbf{u}^t = \left[\begin{array}{cc|cc} \mathbf{b}^{in} & \widetilde{\mathbf{W}}^{in} & \mathbf{b}^{hid} & \widetilde{\mathbf{W}}^{hid} \end{array}\right] \left[\begin{array}{c} 1 \\ \mathbf{x}^t \\ \hline 1 \\ \mathbf{h}^{t-1} \end{array}\right] \quad (7)$$

$$\mathbf{h}^t = \mathbf{f}(\mathbf{u}^t) = \begin{bmatrix} f(u_1^t) \\ f(u_2^t) \\ \vdots \end{bmatrix} \quad (8)$$

と表される．係数行列およびバイアス項が乱数のとき，隠れ層の値 $\mathbf{h_t}$ の情報が失われてしまう．ここで $\mathbf{b}^{in} = \mathbf{b}^{hid} = \{0, \ldots, 0\}, \widetilde{\mathbf{W}}^{in} = \mathbf{O}$(零行列), $\widetilde{\mathbf{W}}^{hid} = \mathbf{I}$(単位行列) と初期値を設けると，

$$\mathbf{u}^t = \left[\begin{array}{cc|cc} \mathbf{0} & \mathbf{O} & \mathbf{0} & \mathbf{I} \end{array}\right] \left[\begin{array}{c} 1 \\ \mathbf{x}^t \\ \hline 1 \\ \mathbf{h}^{t-1} \end{array}\right] = \mathbf{h}^{t-1} \quad (9)$$

となる．また活性化関数 f が ReLU の場合，長い時系列データの入力によって隠れ層の値 u_j^t が十分大きくなっているなら，$f(u_j^t) \simeq u_j^t$ なので，結局

$$\mathbf{h}^t \simeq \mathbf{h}^{t-1} \quad (10)$$

となり，隠れ層の値が次の時刻の隠れ層に記憶されることがわかる．一方，活性化関数が sigmoid 関数や tanh の場合は，$f(u_j^t) \to 1 (u_j^t \to \infty)$ と飽和してしまい，新たな入力の情報が反映されなくなる [4].

一般に，活性化関数を ReLU とすることのデメリットとして，隠れ層の値が著しく大きくなることが挙げられる．特に音声信号など，短い連続した時間で同じような入力が続いた場合，値が爆発的に増加する．IRNN では，隠れ層の係数行列の初期値を，単位行列に小さなスカラー値をかけたものを使うことによって，これを回避している．これは，前の時刻から再利用される隠れ層のゲインを小さくすることによって，長時間データが存続することを防ぐ仕組みで，LSTM における記憶ゲートを弱めて，徐々にデータを忘却することに相当する．

IRNN は，元来のシンプルな RNN の構造を受け継ぎながら，LSTM の長い時系列データを処理できる長所を踏襲しつつ，その学習の遅さというを欠点を克服している．次章では LSTM を用いたプログラムコードの処理を見ながら，IRNN を用いて同じ処理をしていくことを試みる．

2.4 プログラムコードの学習

Zaremba ら [12] は LSTM アーキテクチャを用いて，Python コードを時系列データとして読み込み，プログラムの出力を予測する学習器を提案している．プログラムコードには四則演算，if 文，for 文，変数 (整数) が含まれている．またハイパーパラメータとして，使用される整数の桁数 (length) と構文木の深さ (nesting) が設定されている．学習には確率的勾配降下法 (Stochastic Gradient Descent；SGD) による誤差逆伝播法が用いられているが，length や nesting が大きい場合，乱数を初期値とする LSTM では学習がうまく進まない．そのため，まず小さな length および nesting から学習し，徐々にその数を上げていく方法 (naive curriculum learning[1]) や，一定の範囲の length および nesting をランダムに選んで学習する方法 (mixed strategy) が挙げられる．結果的には，LSTM を用いたプログラムコード処理には両手法を合わせた手法 (combined strategy) で精度が極めて高くなることが示されている．また，足し算がひとつだけ書かれたコード (演算) や，入力値をそのまま返すコード (記憶) を学習させ，データの演算と記憶において，それぞれどの手法による学習が効率的かの実験が示されている．さらに数字の入力について，桁を逆にして入力したり (inverting[10])，同じ数字を繰り返し入力する (doubling) 試みも述べられている．結果として演算に

は combined strategy が，記憶には入力に inverting および doubling を用いた mixed strategy が望ましいことが示されている．

3. 実験

前章で，IRNN の性質が LSTM に似ていることを述べたが，学習においても，LSTM と同じ手法が IRNN に有効かどうかを示す．そのために，2.4 節の問題に対する両手法それぞれの予測誤差の比較を行う．また，IRNN が LSTM に比べてシンプルな構造を持っていることによる収束の速さの違いを示し，さらに IRNN での演算・記憶の学習には，前章の学習手法 (naive curriculum learning, mixed strategy，および combined strategy) のいずれが有効かを示す．LSTM と IRNN は，ともに隠れ層の値を入力に再利用するという共通の構造を持つため，有効な学習手法も同じであると予想される．また IRNN は LSTM にくらべてパラメータ数が少ないことから，予測誤差は早く収束すると予想される．実験内容および結果については口頭で発表する．

参考文献

[1] Bengio, Y., Louradour, J., Collobert, R. and Weston, J.: Curriculum learning, *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 41–48 (2009).

[2] Boedecker, J., Obst, O., Mayer, N. M. and Asada, M.: Initialization and self-organized optimization of recurrent neural network connectivity, *Human Frontier Science Program Journal*, Vol. 3, No. 5, pp. 340–349 (2009).

[3] El Hihi, S. and Bengio, Y.: Hierarchical Recurrent Neural Networks for Long-Term Dependencies., *Advances in Neural Information Processing Systems 8*, pp. 493–499 (1995).

[4] Hochreiter, S.: The vanishing gradient problem during learning recurrent neural nets and problem solutions, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, Vol. 6, No. 2, pp. 107–116 (1998).

[5] Hochreiter, S. and Schmidhuber, J.: Long short-term memory, *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780 (1997).

[6] Le, Q. V., Jaitly, N. and Hinton, G. E.: A Simple Way to Initialize Recurrent Networks of Rectified Linear Units, *arXiv preprint arXiv:1504.00941* (2015).

[7] Mozer, M. C.: A focused back-propagation algorithm for temporal pattern recognition, *Complex Systems*, Vol. 3, No. 4, pp. 349–381 (1989).

[8] Nair, V. and Hinton, G. E.: Rectified linear units improve restricted boltzmann machines, *Proceedings of the 27th International Conference on Machine Learning*, pp. 807–814 (2010).

[9] Prokhorov, D. V.: Backpropagation Through Time and Derivative Adaptive Critics: A Common Framework for Comparison, *Handbook of Learning and Approximate Dynamic Programming*, Vol. 2 (2004).

[10] Sutskever, I., Vinyals, O. and Le, Q. V.: Sequence to sequence learning with neural networks, *Advances in Neural Information Processing Systems 27*, pp. 3104–3112 (2014).

[11] Williams, R. J. and Zipser, D.: A learning algorithm for continually running fully recurrent neural networks, *Neural Computation*, Vol. 1, No. 2, pp. 270–280 (1989).

[12] Zaremba, W. and Sutskever, I.: Learning to Execute, *arXiv preprint arXiv:1410.4615* (2014).