Memory Augmented Neural Networks における記憶利用についての局所解を考慮した Controller の設計

田口 直弥　　鶴岡 慶雅
東京大学大学院 工学系研究科　東京大学大学院 情報理工学研究科
{guchio, tsuruoka}@logos.t.u-tokyo.ac.jp

1 はじめに

近年，自然言語処理などの系列データ処理を行う Recurrent Neural Networks (RNNs) の一種である Memory Augmented Neural Networks (MANNs) が注目を集めている．従来の RNN が記憶量の制限，および時点毎に区切った記憶の行い辛さという問題点を持つ一方，MANN はメモリをベクトルの集合として表現しており，記憶容量の拡大および時点を意識した記憶の選択的利用に成功している [5].

本稿では，幾つか存在する MANN の内 Neural Turing Machine (NTM) [1] に基づくモデルに着目する．NTM に基づく MANN では Controller と呼ばれる素子が中心となり情報処理が行われるが，Controller は任意の Neural Networks (NNs) により実現され，特に経験的に性能が良いため RNN で表現される事が多い．一方，RNN を Controller に利用した場合，後に示すように Controller に使われる RNN 自身の記憶能力を用いてタスクを解くという解が存在し，これが学習に悪影響を与える場合がある．そこで本稿では，RNN を Controller に使用する上でのメリットは残しつつ，Controller 自身の記憶能力を用いてタスクを解く解を取り除くよう Controller を設計することで，MANN の性能向上を図る．

2 背景知識

2.1 MANN

MANN は多数のスロットを持つメモリを用いて記憶を行う．中でも，本稿で扱う NTM に基づく MANN は図 1 に示すように，NN により表現される三つの素子 Controller，Read Heads，および Write Head と一つのメモリにより構成される．各素子は時刻 t において以下 (i) から (iv) に示す処理を行う．

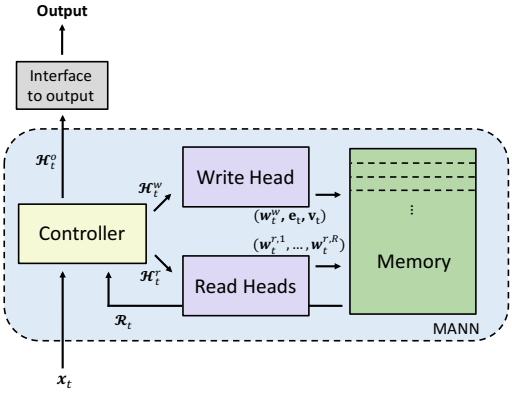

図 1: MANN の概要図.

(i) モデルへの入力 $\boldsymbol{x}_t \in \mathbb{R}^I$ に応じ，Controller が t における Read Heads および Write Head を操作するための情報 $\boldsymbol{\mathcal{H}}_t^r$ および $\boldsymbol{\mathcal{H}}_t^w$ をそれぞれ生成する [1]．Controller は直前の時刻 $t-1$ に R 個の Read Heads によりメモリから読み込まれた情報を保持するタプル $\boldsymbol{\mathcal{R}}_{t-1} = (\boldsymbol{r}_{t-1}^1, ..., \boldsymbol{r}_{t-1}^R)$ を保持しているため，これを使用した各情報の生成が可能である．

(ii) $\boldsymbol{\mathcal{H}}_t^w$ および現状のメモリ $\boldsymbol{M}_{t-1} \in \mathbb{R}^{N\times W}$ に基づき，Write Head が情報を書き込むことにより $\boldsymbol{M}_{t-1}$ を $\boldsymbol{M}_t$ に更新する．ここで，N はメモリのスロット数を，W は各スロットの中身を表すベクトルの大きさを示す．また，スロットとはメモリの各行のことを表す．メモリへの情報の書き込みは式 (1) に示される操作により実現される事が多い．式 (1) で，$\boldsymbol{E} \in \mathbb{R}^{N\times W}$ の要素は全て 1 である．$\boldsymbol{e}_t \in [0,1]^W$ および $\boldsymbol{v}_t \in \mathbb{R}^W$ は t においてメモリの情報を消すおよびメモリに情報を加えるために使用されるベクトルであり，$\boldsymbol{w}_t^w$ は t で各スロットにおける情報の削除および追加を行う強度を決めるベクトルで $\sum_j \boldsymbol{w}_t^w(j) \leq 1$ かつ $\forall j, 0 \leq \boldsymbol{w}_t^w(j) \leq 1$

[1] $\boldsymbol{\mathcal{H}}_t^r$，$\boldsymbol{\mathcal{H}}_t^w$，および $\boldsymbol{\mathcal{H}}_t^o$ は一般にタプルとして表現されるが，本稿では基本的にこれらの要素数が 1 の場合を扱うため，ここでは可読性の観点からこれらをベクトルとする．

を満たす．$\boldsymbol{e}_t$ および $\boldsymbol{v}_t$ は通常 $\boldsymbol{\mathcal{H}}_t^w$ を入力とする一層の NN などにより生成される．

$$\boldsymbol{M}_t = \boldsymbol{M}_{t-1} \odot (\boldsymbol{E} - \boldsymbol{w}_t^w \cdot \boldsymbol{e}_t^T) + \boldsymbol{w}_t^w \cdot \boldsymbol{v}_t^T \quad (1)$$

(iii) $\boldsymbol{\mathcal{H}}_t^r$ および $\boldsymbol{M}_{t-1}$ に基づき，Read Heads がメモリから情報を読み込むことにより $\boldsymbol{\mathcal{R}}_t$ を生成し Controller がこれを保持する．各 Read Head によるメモリからの情報の読み込みは式 (2) に示される操作により実現されることが多い．式 (2) において，$\boldsymbol{w}_t^{r,i}$ は識別子 i により指定される Read Head が各スロットからの情報の読み込みを行う強度を決めるベクトルで $\sum_j \boldsymbol{w}_t^{r,i}(j) \leq 1$ かつ $\forall j,\ 0 \leq \boldsymbol{w}_t^{r,i}(j) \leq 1$ を満たす．

$$\boldsymbol{r}_t^i = \boldsymbol{M}_t^T \cdot \boldsymbol{w}_t^{r,i} \quad (2)$$

(iv) Controller がモデルの出力に使用する情報 $\boldsymbol{\mathcal{H}}_t^o$ を生成する [1]．$\boldsymbol{\mathcal{H}}_t^o$ の生成には $\boldsymbol{x}_t$ および $\boldsymbol{\mathcal{R}}_{t-1}$ に加え，$\boldsymbol{\mathcal{R}}_t$ も使用可能である．

上記の処理において，各素子は t 以前の自身の処理に関する情報を使用できる．例えば Controller は $\boldsymbol{\mathcal{H}}_t^w$ を $\boldsymbol{\mathcal{H}}_1^w$, ..., $\boldsymbol{\mathcal{H}}_{t-1}^w$ に基づき生成でき，また Write Head は $\boldsymbol{w}_t^w$ を $\boldsymbol{w}_{t-1}^w$ に基づき生成できる．

本稿では上記のように設計される MANN の内，NTM および Differentiable Neural Computer (DNC) [2] を扱う．各素子詳細な設計はモデルや実装により違いがあるため，文献 [1] および [2] を参考にされたい．本稿で着目する Controller については 2.2 節で説明する．

2.2 Controller

Controller の設計は通常 NTM や DNC などのモデルにより異なるが，本稿では全てのモデルについて $\boldsymbol{\mathcal{H}}_t^r = \boldsymbol{\mathcal{H}}_t^w = \boldsymbol{\mathcal{H}}_t^o = \boldsymbol{h}_t$ を基準として議論を行う．また，本稿では $\boldsymbol{h}_t$ を生成する NN として式 (3)，(4)，および (5–10) に示す一層の Feedforward Neural Networks (FNNs)，Elman Networks (ENs) および，Long Short-Term Memory (LSTM) を用いる Controller を FNN Controller，EN Controller，および LSTM Controller と呼ぶ．また，EN Controller および LSTM Controller を RNN Controller と総称する．なお，式 (3)，(4)，および (5–10) で σ および σ' はそれぞれ活性化関数およびゲートを表現できる活性化関数で，$\boldsymbol{r}_t = [\boldsymbol{r}_t^1; ...; \boldsymbol{r}_t^R]$ である．また，$\boldsymbol{h}_0 = \boldsymbol{s}_0 = \boldsymbol{0}$ である．

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_{xh} \cdot \boldsymbol{x}_t + \boldsymbol{W}_{rh} \cdot \boldsymbol{r}_t + \boldsymbol{b}) \quad (3)$$

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_{xh} \cdot \boldsymbol{x}_t + \boldsymbol{W}_{rh} \cdot \boldsymbol{r}_t + \boldsymbol{W}_{hh} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}) \quad (4)$$

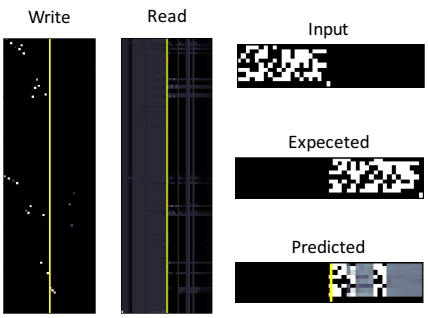

図 2: Odd First において LSTM Controller が DNC の学習に悪影響を及ぼす例．全ての図について横軸は時間を示し，Read および Write で縦軸はメモリのスロットを，Input，Expected，および Output で縦軸は NTM に入出力されるベクトルの各要素を示している．Read および Write では白くなっているメモリのスロットが読み書きに使用されたことを表す．

$$\boldsymbol{z}_t = tanh(\boldsymbol{W}_{xz} \cdot \boldsymbol{x}_t + \boldsymbol{W}_{rz} \cdot \boldsymbol{r}_t + \boldsymbol{W}_{hz} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_z) \quad (5)$$

$$\boldsymbol{i}_t = \sigma'(\boldsymbol{W}_{xi} \cdot \boldsymbol{x}_t + \boldsymbol{W}_{ri} \cdot \boldsymbol{r}_t + \boldsymbol{W}_{hi} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \quad (6)$$

$$\boldsymbol{f}_t = \sigma'(\boldsymbol{W}_{xf} \cdot \boldsymbol{x}_t + \boldsymbol{W}_{rf} \cdot \boldsymbol{r}_t + \boldsymbol{W}_{hf} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \quad (7)$$

$$\boldsymbol{o}_t = \sigma'(\boldsymbol{W}_{xo} \cdot \boldsymbol{x}_t + \boldsymbol{W}_{ro} \cdot \boldsymbol{r}_t + \boldsymbol{W}_{ho} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \quad (8)$$

$$\boldsymbol{s}_t = \boldsymbol{f}_t \odot \boldsymbol{s}_{t-1} + \boldsymbol{i} \odot \boldsymbol{z}_t \quad (9)$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot tanh(\boldsymbol{s}_t) \quad (10)$$

3 提案

3.1 RNN Controller の問題点

RNN Controller を使用した場合，$\boldsymbol{h}_t$ の生成において系列情報を加味できるため，Read Heads および Write Head における情報処理を系列情報を加味して行えるなどの利点がある．一方で $\boldsymbol{h}_t$ は $\boldsymbol{\mathcal{H}}_t^o$ としてモデルの出力にも利用されるため，Controller 自身の記憶能力を利用してタスクを解くという解が存在し，これが学習に悪影響を与えることがある [3, 4]．図 2 にその一例を示す．図 2 において，DNC は 4.2 節に示す Odd First を解くために適切なメモリの利用を行っていないにも関わらず，早期に入力された，もしくは早期に出力するベクトル系列は適切に出力できている．

3.2 提案手法

本稿では RNN Controller を利用する恩恵を受けつつも，Controller 自身の記憶能力を利用してタスクを解く解を避ける新しい Controller を提案する．提案

する Controller は $\boldsymbol{\mathcal{H}}_t^r = \boldsymbol{\mathcal{H}}_t^w = \boldsymbol{h}_t$ かつ $\boldsymbol{\mathcal{H}}_t^o = \boldsymbol{h}'_t$ とし，$\boldsymbol{h}_t$ は Controller 自身の記憶を利用して生成するが，$\boldsymbol{h}'_t$ はこれを利用せず生成する．EN Controller において $\boldsymbol{h}'_t$ の生成は式 (11) のように設計される．

$$\boldsymbol{h}'_t = \sigma(\boldsymbol{W}_{xh} \cdot \boldsymbol{x}_t + \boldsymbol{W}_{rh} \cdot \boldsymbol{r}_t + \boldsymbol{b}) \qquad (11)$$

なお，$\boldsymbol{h}_t$ の生成は式 (4) のように行われ，式 (4) および式 (11) で共通のパラメータは共有されることに注意されたい．同様の設計は EN Controller 以外でも可能であり，本稿では LSTM Controller における $\boldsymbol{h}'_t$ の生成を式 (12) に示す形式で設計する．

$$\boldsymbol{h}'_t = tanh(\boldsymbol{W}_{xz} \cdot \boldsymbol{x}_t + \boldsymbol{W}_{rz} \cdot \boldsymbol{r}_t + \boldsymbol{b}_z) \qquad (12)$$

なお，この場合も同様に $\boldsymbol{h}_t$ の生成は式 (5–10) のように行われ，共通のパラメータが共有される．

4 実験

4.1 実験設定

本稿では提案手法の有効性を検証するため，Toy Task により実験を行った．実験では NTM および DNC に FNN Controller，EN Controller，提案 EN Controller，LSTM Controller，そして提案 LSTM Controller を用いこれらの性能を比較した．学習は RMSProp を学習率を 0.0001，momentum を 0.9 に設定してオンライン学習により行い，勾配を要素毎に $[-10, 10]$ の範囲でクリッピングした．NTM および DNC においてメモリは 128×20，Controller のユニット数は 128 とし，NTM において畳み込みシフトを行う step 数は ± 3 までとした．また，Priority Sort については $R = 4$，それ以外の場合は $R = 1$ とした．比較のため EN および LSTM を用いた評価も行ったが，これらは一層 128 ユニットのものを用いた．なお，実験結果はテストデータセットを用いて算出したが，テスト時のモデルパラメータには学習中のバリデーション誤差の値が最も小さい時点のものを用いた．また，実験は各設定について 10 回ずつ行った．

本稿では文献 [6] を参考に，表 1 に示す 6 つの Toy Task を使用して実験を行った．各タスクでモデルは大きさ 9 で各要素が 0 もしくは 1 の二値ベクトルが系列長 $T \in \mathbb{N}$ だけ入力された後，ベクトルの 9 要素目で表現される入力の終了を知らせるフラグを受け取り，その後表 1 に示されるタスク毎に適切なベクトル系列を出力するように学習が行われる．T は入力系列毎にランダムに設定される．学習は全てのタスクで 300,000 系列ずつ行い，1,000 系列のデータセットによりバリデーションを，10,000 系列のデータセットによりテストを行った．また，Repeat Copy では $T \in [1, 10]$ および繰り返し回数 $M \in \mathbb{N}$ を $M \in [1, 10]$ とし，それ以外のタスクでは $T \in [1, 20]$ とした．なお，モデルは自身の出力が終わると終端記号を出力するよう学習され，また Repeat Copy においては各繰り返しが終わるごとにこれを出力する．

	Input	Output
Copy	$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_3\boldsymbol{a}_4 \dots \boldsymbol{a}_T$	$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_3\boldsymbol{a}_4 \dots \boldsymbol{a}_T$
Reverse	$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_3\boldsymbol{a}_4 \dots \boldsymbol{a}_T$	$\boldsymbol{a}_T\boldsymbol{a}_{T-1}\boldsymbol{a}_{T-2}\boldsymbol{a}_{T-3} \dots \boldsymbol{a}_1$
Bigram Flip	$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_3\boldsymbol{a}_4 \dots \boldsymbol{a}_T$	$\boldsymbol{a}_2\boldsymbol{a}_1\boldsymbol{a}_4\boldsymbol{a}_3 \dots \boldsymbol{a}_T$
Odd First	$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_3\boldsymbol{a}_4 \dots \boldsymbol{a}_T$	$\boldsymbol{a}_1\boldsymbol{a}_3 \dots \boldsymbol{a}_2\boldsymbol{a}_4 \dots$
Repeat Copy	$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_3\boldsymbol{a}_4 \dots \boldsymbol{a}_T M$	$\boldsymbol{a}_1 \dots \boldsymbol{a}_T \dots \boldsymbol{a}_1 \dots \boldsymbol{a}_T\,(M\,times)$
Priority Sort	$\boldsymbol{a}_1^5\boldsymbol{a}_2^{10}\boldsymbol{a}_3^1\boldsymbol{a}_4^T \dots \boldsymbol{a}_T^4$	$\boldsymbol{a}_3^1\boldsymbol{a}_7^2\boldsymbol{a}_8^3\boldsymbol{a}_T^4 \dots \boldsymbol{a}_4^T$

表 1: 各 Toy Task 詳細．Priority Sort における上付き文字は各ベクトルの優先度を示す．

図 3: Copy において DNC に LSTM Controller を用いて行った場合の出力の例．

4.2 実験結果

実験結果を表 2 に示す．表 2 より，いずれかの提案手法を使用した場合において各タスクの最良値が出されていることがわかる．また，それぞれのモデルについて DNC に LSTM Controller を使用した場合以外で基本的に提案手法を利用した方が良い結果となっている．一方，非常に悪い精度を出した DNC で提案 LSTM Controller を用いた場合では図 3 に例が示されるようにベクトルの要素の内フラグを表す部分で常に黒の出力を行い，そうでない部分では全て白の出力を行う局所解に収束している．この局所解は記憶を必要としないものであるため，我々は DNC に提案 LSTM Controller を用いたモデルはメモリを利用した解に収束することが従来の Controller を用いた場合よりも難しいと考える．T の平均値が小さくかつ終端記号の数が多いためこの局所解に収束し辛い Repeat Copy および $R = 4$ でメモリが利用しやすい Priority Sort で比較的良い精度が出ていることも，我々の主張を裏付ける．

	EN	LSTM	NTM					DNC				
			FNN	EN	proposed EN	LSTM	proposed LSTM	FNN	EN	proposed EN	LSTM	proposed LSTM
Copy	39.2	30.4	**0.0**	**0.0**	**0.0**	6.8	**0.0**	4.3	14.8	2.2	9.8	44.4
Reverse	27.1	19.9	4.3	2.1	**0.0**	2.4	6.9	9.0	**0.0**	13.8	14.3	44.4
Bigram Flip	38.5	21.5	4.9	1.4	0.8	17.0	0.5	16.1	12.7	**0.0**	10.9	44.4
Odd First	38.6	19.5	10.6	8.7	3.9	16.7	**0.9**	23.4	8.4	7.0	17.3	44.4
Repeat Copy	31.6	24.2	6.0	10.4	**0.3**	22.4	20.0	12.2	5.8	5.4	21.8	19.7
Priority Sort	30.4	19.7	18.5	21.2	**11.4**	18.6	16.3	27.2	26.4	22.9	18.3	34.0

表 2: 各 Toy Task における平均のビットエラーレート．太字は各タスクにおける最良値を示す．

5 関連研究

MANN に関する研究の多くでは新たな構造により設計される MANN を提案しているが，その殆どが Read Heads および Write Head，そしてメモリに関して改良を加えるものである．本稿は Controller の改良に着目するという点でこれらの研究とは異なる．一方，文献 [3] では本稿同様 Controller 自身の記憶能力を使ってタスクを解くことを避ける試みを行っているが，これを $\boldsymbol{\mathcal{H}}_t^o = (\boldsymbol{x}_t, \boldsymbol{\mathcal{R}}_t)$ とすることにより実現している．これは MANN の性能を向上するという目的を $\boldsymbol{\mathcal{H}}_t^o$ を工夫することにより実現するという点で本稿と同様だが，本稿はこれを Controller に使用する NN の改良により行うという点で異なる．

また，MANN と呼ばれるモデルは NTM に基づくもの以外にも様々なものがあり，特に Memory Networks [5] に基づくモデルについては多く研究が行われている．Memory Networks に基づくモデルは，時刻 t において複数回メモリから読み込みを行える点や，出力を行うためにメモリから読み込んだ情報の処理に RNN を用いる点など，NTM に基づく MANN とは幾つかの相違点がある．

6 おわりに

本稿では，Controller 自身の記憶の利用法を改良することにより NTM に基づく MANN の性能向上を試みた．Toy Task による実験により，提案手法は MANN の性能を向上させる上で非常に有効なことが分かった反面，特に LSTM Controller に適用した場合にメモリを使用した学習の難易度が増大するという懸念点も抱えることが分かった．

Toy Task は人工データであるため，今後の方向性として実データによるより実践的な実験を行うことが挙げられる．

謝辞

本研究は JST CREST(課題番号: JPMJCR1513) の支援を受けて行った．

参考文献

[1] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[2] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwiska, Sergio Gmez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, and others. Hybrid computing using a neural network with dynamic external memory. *Nature*, Vol. 538, No. 7626, pp. 471–476, 2016.

[3] Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory Augmented Neural Networks with Wormhole Connections. *arXiv:1701.08718 [cs, stat]*, January 2017. arXiv: 1701.08718.

[4] Carol Hsin. Implementation and Optimization of Differentiable Neural Computers. *A class in Stanford University (CS224), 2753780*, 2017.

[5] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[6] Greg Yang and Alexander M. Rush. Lie-Access Neural Turing Machines. *arXiv:1611.02854 [cs]*, November 2016. arXiv: 1611.02854.