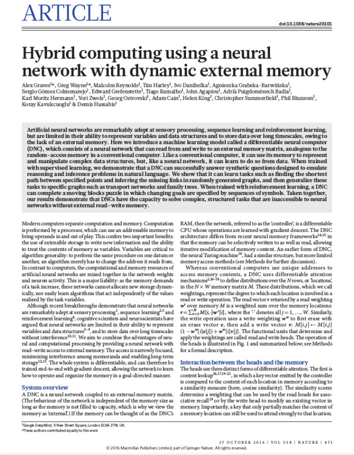
Differentiable Neural Computers

Oliver Gäfvert

KTH Royal Institute of Technology

oliverg@kth.se

- The Differentiable Neural Computer (DNC) was introduced by A.Graves et al. at DeepMind.
- Novel way of combining symbolic and numeric computation.
- First iteration called "Neural Turing Machine".
- "Neural network with an external memory".



ARTICLE

doi:10.1038/nature20101

Hybrid computing using a neural network with dynamic external memory

Alex Graves[1*], Greg Wayne[1*], Malcolm Reynolds[1], Tim Harley[1], Ivo Danihelka[1], Agnieszka Grabska-Barwińska[1], Sergio Gómez Colmenarejo[1], Edward Grefenstette[1], Tiago Ramalho[1], John Agapiou[1], Adrià Puigdomènech Badia[1], Karl Moritz Hermann[1], Yori Zwols[1], Georg Ostrovski[1], Adam Cain[1], Helen King[1], Christopher Summerfield[1], Phil Blunsom[1], Koray Kavukcuoglu[1] & Demis Hassabis[1]

Artificial neural networks are remarkably adept at sensory processing, sequence learning and reinforcement learning, but are limited in their ability to represent variables and data structures and to store data over long timescales, owing to the lack of an external memory. Here we introduce a machine learning model called a differentiable neural computer (DNC), which consists of a neural network that can read from and write to an external memory matrix, analogous to the random-access memory in a conventional computer. Like a conventional computer, it can use its memory to represent and manipulate complex data structures, but, like a neural network, it can learn to do so from data. When trained with supervised learning, we demonstrate that a DNC can successfully answer synthetic questions designed to emulate reasoning and inference problems in natural language. We show that it can learn tasks such as finding the shortest path between specified points and inferring the missing links in randomly generated graphs, and then generalize these tasks to specific graphs such as transport networks and family trees. When trained with reinforcement learning, a DNC can complete a moving blocks puzzle in which changing goals are specified by sequences of symbols. Taken together, our results demonstrate that DNCs have the capacity to solve complex, structured tasks that are inaccessible to neural networks without external read–write memory.

Modern computers separate computation and memory. Computation is performed by a processor, which can use an addressable memory to bring operands in and out of play. This confers two important benefits: the use of extensible storage to write new information and the ability to treat the contents of memory as variables. Variables are critical to algorithm generality: to perform the same procedure on one datum or another, an algorithm merely has to change the address it reads from. In contrast to computers, the computational and memory resources of artificial neural networks are mixed together in the network weights and neuron activity. This is a major liability: as the memory demands of a task increase, these networks cannot allocate new storage dynamically, nor easily learn algorithms that act independently of the values realized by the task variables.

Although recent breakthroughs demonstrate that neural networks are remarkably adept at sensory processing[1], sequence learning[2,3] and reinforcement learning[4], cognitive scientists and neuroscientists have argued that neural networks are limited in their ability to represent variables and data structures[5–9], and to store data over long timescales without interference[10,11]. We aim to combine the advantages of neural and computational processing by providing a neural network with read–write access to external memory. The access is narrowly focused, minimizing interference among memoranda and enabling long-term storage[12,13]. The whole system is differentiable, and can therefore be trained end-to-end with gradient descent, allowing the network to learn how to operate and organize the memory in a goal-directed manner.

System overview

A DNC is a neural network coupled to an external memory matrix. (The behaviour of the network is independent of the memory size as long as the memory is not filled to capacity, which is why we view the memory as 'external'.) If the memory can be thought of as the DNC's RAM, then the network, referred to as the 'controller', is a differentiable CPU whose operations are learned with gradient descent. The DNC architecture differs from recent neural memory frameworks[14,15] in that the memory can be selectively written to as well as read, allowing iterative modification of memory content. An earlier form of DNC, the neural Turing machine[16], had a similar structure, but more limited memory access methods (see Methods for further discussion).

Whereas conventional computers use unique addresses to access memory contents, a DNC uses differentiable attention mechanisms[2,16–18] to define distributions over the N rows, or 'locations', in the $N \times W$ memory matrix M. These distributions, which we call weightings, represent the degree to which each location is involved in a read or write operation. The read vector $\boldsymbol{r}$ returned by a read weighting $\boldsymbol{w}^r$ over memory M is a weighted sum over the memory locations: $\boldsymbol{r}=\sum_{i=1}^{N} M[i,\cdot]\boldsymbol{w}^r[i]$, where the '·' denotes all $j=1, \ldots, W$. Similarly, the write operation uses a write weighting $\boldsymbol{w}^w$ to first erase with an erase vector $\boldsymbol{e}$, then add a write vector $\boldsymbol{v}$: $M[i,j] \leftarrow M[i,j](1-\boldsymbol{w}^w[i]\boldsymbol{e}[j]) + \boldsymbol{w}^w[i]\boldsymbol{v}[j]$. The functional units that determine and apply the weightings are called read and write heads. The operation of the heads is illustrated in Fig. 1 and summarized below; see Methods for a formal description.

Interaction between the heads and the memory

The heads use three distinct forms of differentiable attention. The first is content lookup[16,17,19–21], in which a key vector emitted by the controller is compared to the content of each location in memory according to a similarity measure (here, cosine similarity). The similarity scores determine a weighting that can be used by the read heads for associative recall[19] or by the write head to modify an existing vector in memory. Importantly, a key that only partially matches the content of a memory location can still be used to attend strongly to that location.

[1]Google DeepMind, 5 New Street Square, London EC4A 3TW, UK.
*These authors contributed equally to this work.

Turing machine

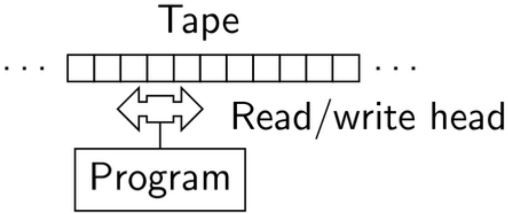


Figure : Illustration a Turing machine. (Image source: [1].)

- A Turing machine is a computational model similar to how computers work.
- Consists of a memory tape and read/write heads with a set of instructions.

DNC architecture

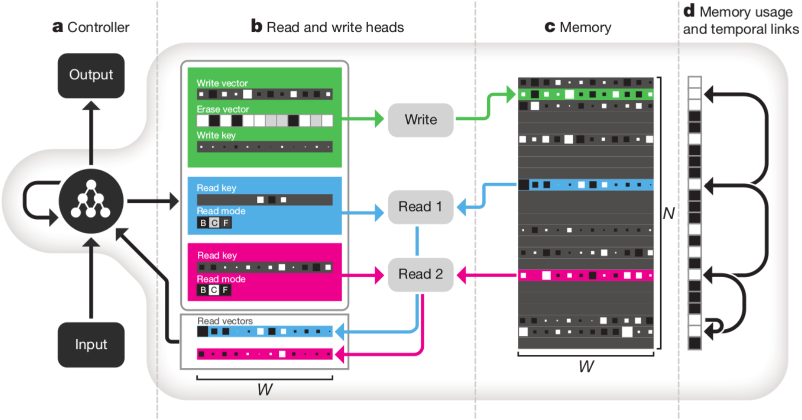


Figure : Illustration of DNC architecture. (Image taken from [3].)

Why is this interesting?

- It's believed to be necessary to introduce more structure to represent complex data structures and to store data over long timescales.

Why is this interesting?

- It's believed to be necessary to introduce more structure to represent complex data structures and to store data over long timescales.
- The model is in some sense interpretable.

Why is this interesting?

- It's believed to be necessary to introduce more structure to represent complex data structures and to store data over long timescales.
- The model is in some sense interpretable.
- The memory has parallels with how the brain represents information.

How Brains Represent Thousands of Objects

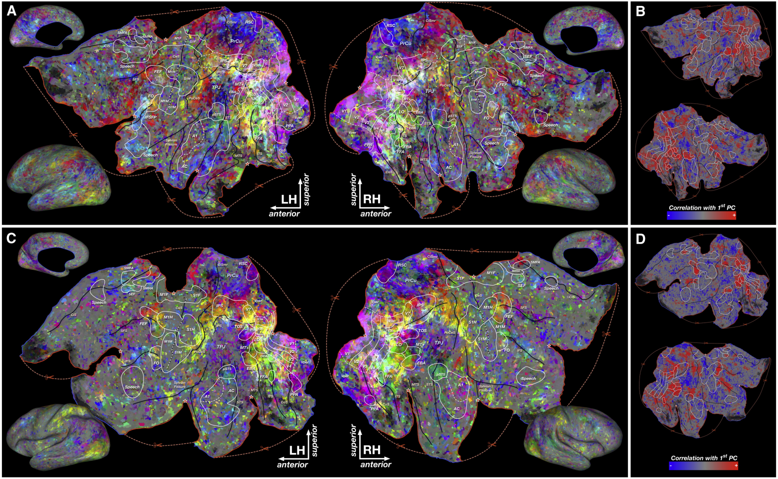


Figure : A semantic map of object representations in the brain, where each colour is an object category. (Image source: [7].)

Software

Software

An instructive implementation by Mostafa Samir can be found here (in modified form):
https://github.com/olivergafvert/DNC-tensorflow

Software

An instructive implementation by Mostafa Samir can be found here (in modified form):
https://github.com/olivergafvert/DNC-tensorflow

- `dnc.py`
- `controller.py`
- `memory.py`
- `metrics.py`
- `utility.py`

The Copy Task

Figure : Memory visualization of the copy task.

Details

The Controller

- A neural network (in the paper a deep LSTM).

The Controller

- A neural network (in the paper a deep LSTM).
- At every time-step the DNC gets input $\mathbf{x}_t \in \mathbb{R}^n$ and outputs $\mathbf{y}_t \in \mathbb{R}^m$.

The Controller

- A neural network (in the paper a deep LSTM).
- At every time-step the DNC gets input $\mathbf{x}_t \in \mathbb{R}^n$ and outputs $\mathbf{y}_t \in \mathbb{R}^m$.

The input to the controller is the following:

$$\chi_t = [\mathbf{x}_t; \mathbf{r}^1_{t-1}; \ldots; \mathbf{r}^R_{t-1}]$$

The Controller

- A neural network (in the paper a deep LSTM).
- At every time-step the DNC gets input $\mathbf{x}_t \in \mathbb{R}^n$ and outputs $\mathbf{y}_t \in \mathbb{R}^m$.

The input to the controller is the following:

$$\boldsymbol{\chi}_t = [\mathbf{x}_t; \mathbf{r}_{t-1}^1; \ldots; \mathbf{r}_{t-1}^R]$$

$$(\boldsymbol{\nu}_t, \boldsymbol{\xi}_t) = \mathcal{N}([\boldsymbol{\chi}_1; \ldots; \boldsymbol{\chi}_t], \boldsymbol{\theta})$$

The Controller

- A neural network (in the paper a deep LSTM).
- At every time-step the DNC gets input $\mathbf{x}_t \in \mathbb{R}^n$ and outputs $\mathbf{y}_t \in \mathbb{R}^m$.

The input to the controller is the following:

$$\chi_t = [\mathbf{x}_t; \mathbf{r}_{t-1}^1; \ldots; \mathbf{r}_{t-1}^R]$$

$$(\boldsymbol{\nu}_t, \boldsymbol{\xi}_t) = \mathcal{N}([\chi_1; \ldots; \chi_t], \boldsymbol{\theta})$$

$$\mathbf{y}_t = \boldsymbol{\nu}_t + W_{\mathbf{r}}[\mathbf{r}_t^1; \ldots; \mathbf{r}_t^R]$$

Deep LSTM controller

For each layer, $l \in \{1, \ldots, L\}$, we have the following equations:

$$
\begin{aligned}
\mathbf{i}_t^l &= \sigma(W_{\mathbf{i}}^l[\chi_t; \mathbf{h}_t^{l-1}; \mathbf{h}_{t-1}^l] + \mathbf{b}_{\mathbf{i}}^l) \\
\mathbf{f}_t^l &= \sigma(W_{\mathbf{f}}^l[\chi_t; \mathbf{h}_t^{l-1}; \mathbf{h}_{t-1}^l] + \mathbf{b}_{\mathbf{f}}^l) \\
\mathbf{s}_t^l &= \mathbf{f}_t^l \mathbf{s}_{t-1}^l + \mathbf{i}_t^l \tanh(W_{\mathbf{s}}^l[\chi_t; \mathbf{h}_{t-1}^l; \mathbf{h}_t^{l-1}] + \mathbf{b}_{\mathbf{s}}^l) \\
\mathbf{o}_t^l &= \sigma(W_{\mathbf{o}}^l[\chi_t; \mathbf{h}_t^{l-1}; \mathbf{h}_{t-1}^l] + \mathbf{b}_{\mathbf{o}}^l) \\
\mathbf{h}_t^l &= \mathbf{o}_t^l \tanh(\mathbf{s}_t^l)
\end{aligned}
$$

Deep LSTM controller

For each layer, $l \in \{1, \ldots, L\}$, we have the following equations:

$$
\begin{aligned}
\mathbf{i}_t^l &= \sigma(W_\mathbf{i}^l[\chi_t; \mathbf{h}_t^{l-1}; \mathbf{h}_{t-1}^l] + \mathbf{b}_\mathbf{i}^l) \\
\mathbf{f}_t^l &= \sigma(W_\mathbf{f}^l[\chi_t; \mathbf{h}_t^{l-1}; \mathbf{h}_{t-1}^l] + \mathbf{b}_\mathbf{f}^l) \\
\mathbf{s}_t^l &= \mathbf{f}_t^l \mathbf{s}_{t-1}^l + \mathbf{i}_t^l \tanh(W_\mathbf{s}^l[\chi_t; \mathbf{h}_{t-1}^l; \mathbf{h}_t^{l-1}] + \mathbf{b}_\mathbf{s}^l) \\
\mathbf{o}_t^l &= \sigma(W_\mathbf{o}^l[\chi_t; \mathbf{h}_t^{l-1}; \mathbf{h}_{t-1}^l] + \mathbf{b}_\mathbf{o}^l) \\
\mathbf{h}_t^l &= \mathbf{o}_t^l \tanh(\mathbf{s}_t^l)
\end{aligned}
$$

The outputs are the following:

$$
\begin{aligned}
\boldsymbol{\nu}_t &= W_\mathbf{y}[\mathbf{h}_t^1; \ldots; \mathbf{h}_t^L] \\
\boldsymbol{\xi}_t &= W_{\boldsymbol{\xi}}[\mathbf{h}_t^1; \ldots; \mathbf{h}_t^L]
\end{aligned}
$$

Memory Interface

We connect the controller to the memory via the interface vector $\boldsymbol{\xi}_t$ where we interpret the coordinates as follows:

$$\boldsymbol{\xi}_t = [\mathbf{k}_t^{\mathbf{r},1}; \ldots; \mathbf{k}_t^{\mathbf{r},R}; \hat{\beta}_t^{\mathbf{r},1}; \ldots; \hat{\beta}_t^{\mathbf{r},R}; \mathbf{k}_t^{w}; \hat{\beta}_t^{w}; \hat{\mathbf{e}}_t; \mathbf{v}_t;$$

$$; \hat{f}_t^{1}; \ldots; \hat{f}_t^{R}; \hat{g}_t^{a}; \hat{g}_t^{w}; \hat{\pi}_t^{1}; \ldots; \hat{\pi}_t^{R}]$$

Write to Memory

The memory is updated according to:

$$M_t = M_{t-1} \circ (E - \mathbf{w}_t^w \mathbf{e}_t^T) + \mathbf{w}_t^w \mathbf{v}_t^T$$

where E is an $N \times W$ matrix of ones and $\mathbf{w}_t^w \in \Delta_n$.

Write to Memory

The memory is updated according to:

$$M_t = M_{t-1} \circ (E - \mathbf{w}_t^w \mathbf{e}_t^T) + \mathbf{w}_t^w \mathbf{v}_t^T$$

where E is an $N \times W$ matrix of ones and $\mathbf{w}_t^w \in \Delta_n$.

$$\mathbf{w}_t^w = g_t^w [g_t^a \mathbf{a}_t + (1 - g_t^a) \mathbf{c}_t^w]$$

$$\begin{aligned} g_t^w &- \text{write gate } \in [0,1] \\ g_t^a &- \text{allocation gate} \in [0,1] \\ \mathbf{c}_t^w &- \text{content-based weighting} \\ \mathbf{a}_t &- \text{allocation-based weighting} \end{aligned}$$

Content-based addressing

First define

$$\mathcal{C}(M, \mathbf{k}, \beta)[i] := \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i, -])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j, -])\beta\}}$$

where $\mathcal{D}$ is some metric (here we use cosine-distance). Then,

$$\mathbf{c}_t^w := \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w)$$

Dynamic memory allocation

Usage vector

$$\mathbf{u}_t = (\mathbf{u}_{t-1} + \mathbf{w}^w_{t-1} - \mathbf{u}_{t-1} \circ \mathbf{w}^w_{t-1}) \circ \psi_t$$

where ψ is the memory-retention vector, representing the amount that should not be freed at each location:

$$\psi_t = \prod_{i=1}^{R} (\mathbf{1} - f^i_t \mathbf{w}^{r,i}_{t-1})$$

Dynamic memory allocation

Usage vector

$$\mathbf{u}_t = (\mathbf{u}_{t-1} + \mathbf{w}^w_{t-1} - \mathbf{u}_{t-1} \circ \mathbf{w}^w_{t-1}) \circ \psi_t$$

where ψ is the memory-retention vector, representing the amount that should not be freed at each location:

$$\psi_t = \prod_{i=1}^{R} (\mathbf{1} - f^i_t \mathbf{w}^{r,i}_{t-1})$$

Then,

$$\mathbf{a}_t[\phi_t[j]] = (1 - \mathbf{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \mathbf{u}_t[\phi_t[i]]$$

where ϕ_t is a sorted free-list where $\phi[1]$ is the "least used" index of $\mathbf{u}_t$.

Dynamic memory allocation

Usage vector

$$\mathbf{u}_t = (\mathbf{u}_{t-1} + \mathbf{w}^w_{t-1} - \mathbf{u}_{t-1} \circ \mathbf{w}^w_{t-1}) \circ \psi_t$$

where ψ is the memory-retention vector, representing the amount that should not be freed at each location:

$$\psi_t = \prod_{i=1}^{R} (\mathbf{1} - f^i_t \mathbf{w}^{r,i}_{t-1})$$

Then,

$$\mathbf{a}_t[\phi_t[j]] = (1 - \mathbf{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \mathbf{u}_t[\phi_t[i]]$$

where ϕ_t is a sorted free-list where $\phi[1]$ is the "least used" index of $\mathbf{u}_t$. Note that this introduces a discontinuity.

Reading from Memory

The read vectors are computes as follows:

$$\mathbf{r}_t^i = M_t^T \mathbf{w}_t^{r,i}$$

Reading from Memory

The read vectors are computes as follows:

$$\mathbf{r}_t^i = M_t^T \mathbf{w}_t^{r,i}$$

where $\mathbf{w}_t^{r,i}$ is a distribution over the rows of M_t computed as:

$$\mathbf{w}_t^{r,i} = \pi_t^i[1]\mathbf{b}_t^i + \pi_t^i[2]\mathbf{c}_t^{r,i} + \pi_t^i[3]\mathbf{f}_t^i$$

π_t^i – read mode

$\mathbf{b}_t^i$ – backwards temporal weighting

$\mathbf{f}_t^i$ – forwards temporal weighting

$\mathbf{c}_t^{r,i}$ – content-based weighting

Temporal Link Matrix

The tracking of temporal links is done via the matrix:

$$L_0[i,j] = 0 \; \forall i,j$$
$$L_t[i,i] = 0 \; \forall i$$
$$L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$$

where $\mathbf{p}_t[i]$ represents the degree to which row i was the last one written to.

Temporal Link Matrix

The tracking of temporal links is done via the matrix:

$$
\begin{aligned}
&L_0[i,j] = 0 \ \forall i,j \\
&L_t[i,i] = 0 \ \forall i \\
&L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j]) L_{t-1}[i,j] + \mathbf{w}_t^w[i] \mathbf{p}_{t-1}[j]
\end{aligned}
$$

where $\mathbf{p}_t[i]$ represents the degree to which row i was the last one written to. It is defined as

$$\mathbf{p}_0 = 0, \qquad \mathbf{p}_t = \left(1 - \sum_i \mathbf{w}_t^w[i]\right)\mathbf{p}_{t-1} + \mathbf{w}_t^w$$

$$\mathbf{f}_t^i = L_t \mathbf{w}_{t-1}^{r,i}, \qquad \mathbf{b}_t^i = L_t^T \mathbf{w}_{t-1}^{r,i}$$

Experiments

Graph tasks

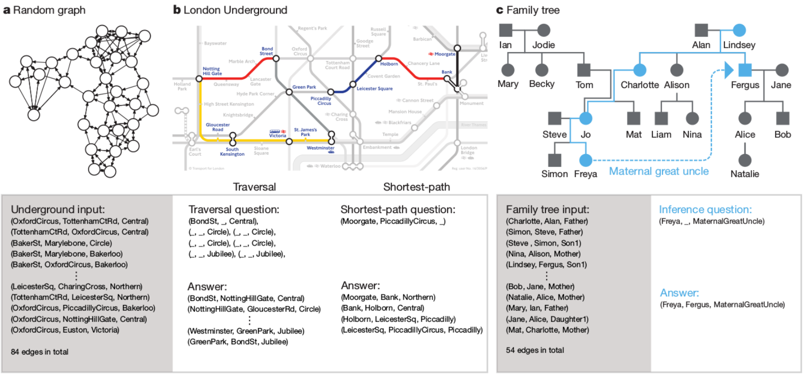


Figure : Illustration of a DNC solving graph problems. (Image taken from [3].)

Mini-SHRDLU

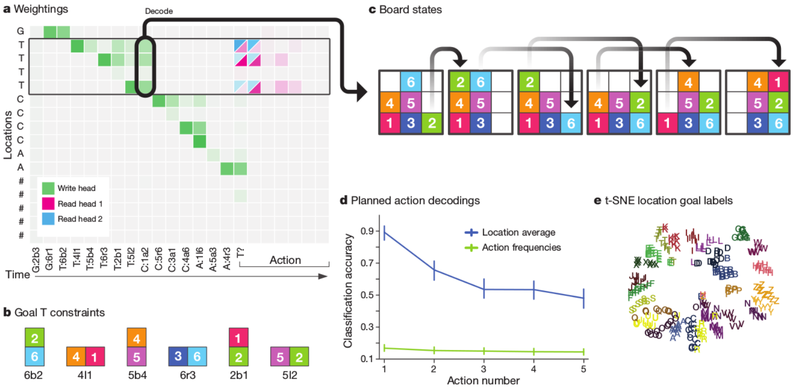


Figure : Illustration of the mini-SHRDLU task. (Image taken from [3].)

Alternative Architectures

There are a number of alternative architectures:

- Neural Turing Machine (NTM) [2]
- Dynamic Neural Turing Machine (D-NTM) [4]
- Recurrent Entity Network (EntNet) [5]
- Pointer Network (Ptr-Net) [9]
- End-to-end memory network (MemN2N) [8]
- Dynamic Memory Network (DMN+) [10]

Alternative Architectures

There are a number of alternative architectures:

- Neural Turing Machine (NTM) [2]
- Dynamic Neural Turing Machine (D-NTM) [4]
- Recurrent Entity Network (EntNet) [5]
- Pointer Network (Ptr-Net) [9]
- End-to-end memory network (MemN2N) [8]
- Dynamic Memory Network (DMN+) [10]

Detailed description of an implementation project of a DNC can be found in [6]. Discuss problems with instability and gradient spikes during training.

Comparison on the bAbI dataset

Task	NTM	D-NTM	MemN2N	DNC	DMN+	EntNet
1: 1 supporting fact	31.5	4.4	0	0	0	0
2: 2 supporting facts	54.5	27.5	0.3	0.4	0.3	0.1
3: 3 supporting facts	43.9	71.3	2.1	1.8	1.1	4.1
4: 2 argument relations	0	0	0	0	0	0
5: 3 argument relations	0.8	1.7	0.8	0.8	0.5	0.3
6: yes/no questions	17.1	1.5	0.1	0	0	0.2
7: counting	17.8	6.0	2.0	0.6	2.4	0
8: lists/sets	13.8	1.7	0.9	0.3	0.0	0.5
9: simple negation	16.4	0.6	0.3	0.2	0.0	0.1
10: indefinite knowledge	16.6	19.8	0	0.2	0	0.6
11: basic coreference	15.2	0	0.0	0	0.0	0.3
12: conjunction	8.9	6.2	0	0	0.2	0
13: compound coreference	7.4	7.5	0	0	0	1.3
14: time reasoning	24.2	17.5	0.2	0.4	0.2	0
15: basic deduction	47.0	0	0	0	0	0
16: basic induction	53.6	49.6	51.8	55.1	45.3	0.2
17: positional reasoning	25.5	1.2	18.6	12.0	4.2	0.5
18: size reasoning	2.2	0.2	5.3	0.8	2.1	0.3
19: path finding	4.3	39.5	2.3	3.9	0.0	2.3
20: agent's motivation	1.5	0	0	0	0	0
Failed Tasks (> 5% error):	16	9	3	2	1	**0**
Mean Error:	20.1	12.8	4.2	3.8	2.8	**0.5**

Figure : Comparison of different architectures on the bAbI dataset with 10k training examples. (Table taken from [5].)

Extensions

Extensions

- Train the model on sequences of varying length and see if generalization improves.
- Use LSTM controller for copy task and compare with feed-forward model.
- Use other metric than cosine for content-based lookup (Euclidean, Manhattan etc).
- Implement differentiable version of allocation part:

$$\mathbf{a}_t[i] = \frac{1 - \mathbf{u}_t[i]}{n - \sum_{j=1}^{n} \mathbf{u}_t[j]}$$

  Alternatively:

$$\mathbf{a}_t[i] = \frac{\exp\{(\mathbf{u}_t[i] - 1)\beta^a\}}{\sum_{j=1} \exp\{(\mathbf{u}_t[j] - 1)\beta^a\}}$$

References I

Mike DeHaan.
The turing machine: A brief introduction.
https://www.decodedscience.org/the-turing-machine-a-brief-introduction/10852/2.

Alex Graves, Greg Wayne, and Ivo Danihelka.
Neural turing machines.
CoRR, abs/1410.5401, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis.
Hybrid computing using a neural network with dynamic external memory.
Nature, 538(7626):471–476, October 2016.

Çaglar Gülçehre, Sarath Chandar, Kyunghyun Cho, and Yoshua Bengio.
Dynamic neural turing machine with soft and hard addressing schemes.
CoRR, abs/1607.00036, 2016.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun.
Tracking the world state with recurrent entity networks.
CoRR, abs/1612.03969, 2016.

Carol Hsin.
Implementation and optimization of differentiable neural computers.
https://web.stanford.edu/class/cs224n/reports/2753780.pdf.

References II

Alexander G. Huth, Shinji Nishimoto, An T. Vu, and Jack L. Gallant.
A continuous semantic space describes the representation of thousands of object and action categories across the human brain.
Neuron, 76(6):1210 – 1224, 2012.

Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus.
End-to-end memory networks.
In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2440–2448. Curran Associates, Inc., 2015.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly.
Pointer networks.
In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015.

Caiming Xiong, Stephen Merity, and Richard Socher.
Dynamic memory networks for visual and textual question answering.
In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2397–2406. JMLR.org, 2016.