Neural networks with dynamic external memory

Differentiable neural computer

Maciej Żelaszczyk

December 13, 2017

PhD Student in Computer Science
Division of Artificial Intelligence and Computational Methods
Faculty of Mathematics and Information Science

m.zelaszczyk@mini.pw.edu.pl

Warsaw University of Technology

Recurrent neural networks

- Feedforward nets process one input at a time.

Recurrent neural networks

- Feedforward nets process one input at a time.
- Order might be important (e.g. text, sound, video).

Recurrent neural networks

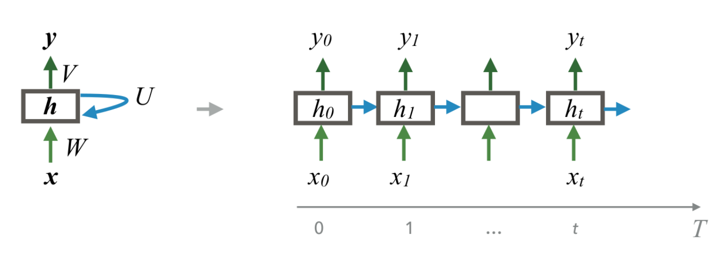
- Feedforward nets process one input at a time.
- Order might be important (e.g. text, sound, video).
- Need to divide data into chunks and process it in sequence.

Recurrent neural networks

- Feedforward nets process one input at a time.
- Order might be important (e.g. text, sound, video).
- Need to divide data into chunks and process it in sequence.
- Adapt feedforward architecture.

Recurrent neural networks

- Feedforward nets process one input at a time.
- Order might be important (e.g. text, sound, video).
- Need to divide data into chunks and process it in sequence.
- Adapt feedforward architecture.

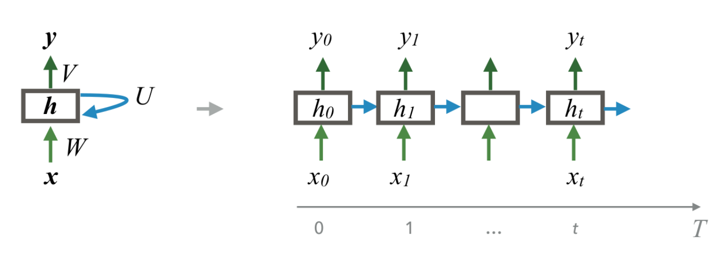


Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Vanilla RNNs

- Classic RNN architecture.

$$\mathbf{h_t} = \tanh\left(\begin{bmatrix}\mathbf{U} & \mathbf{W}\end{bmatrix}\begin{bmatrix}\mathbf{h_{t-1}} \\ \mathbf{x_t}\end{bmatrix}\right)$$



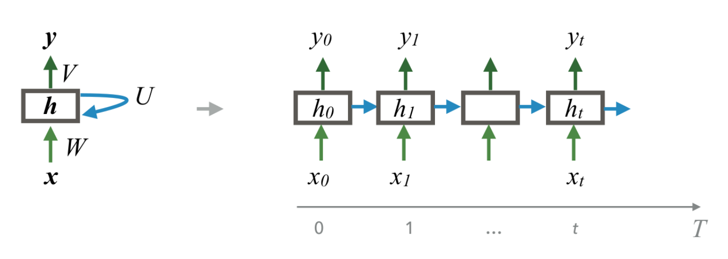
Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Vanilla RNNs

- Classic RNN architecture.

$$\mathbf{h_t} = \tanh\left(\begin{bmatrix} \mathbf{U} & \mathbf{W} \end{bmatrix} \begin{bmatrix} \mathbf{h_{t-1}} \\ \mathbf{x_t} \end{bmatrix}\right)$$

- Possible to think of $\mathbf{h_t}$ as of internal memory.



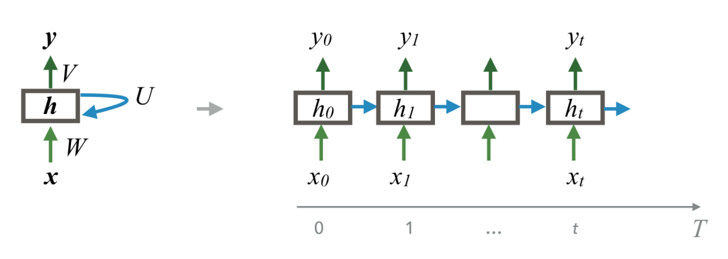
Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Vanilla RNNs

- Classic RNN architecture.

$$\mathbf{h_t} = \tanh\left(\begin{bmatrix}\mathbf{U} & \mathbf{W}\end{bmatrix}\begin{bmatrix}\mathbf{h_{t-1}} \\ \mathbf{x_t}\end{bmatrix}\right)$$

- Possible to think of $\mathbf{h_t}$ as of internal memory.
- In practice, this only works for a couple of steps.



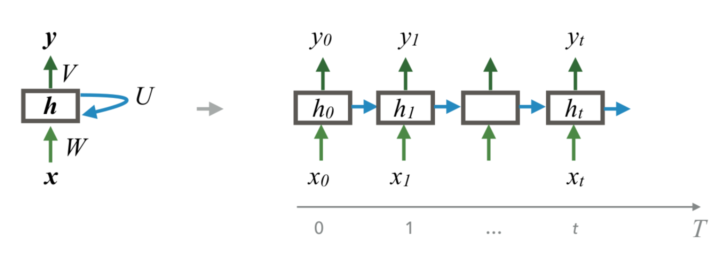
Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Vanilla RNNs

- Classic RNN architecture.

$$\mathbf{h_t} = \tanh\left(\begin{bmatrix}\mathbf{U} & \mathbf{W}\end{bmatrix}\begin{bmatrix}\mathbf{h_{t-1}} \\ \mathbf{x_t}\end{bmatrix}\right)$$

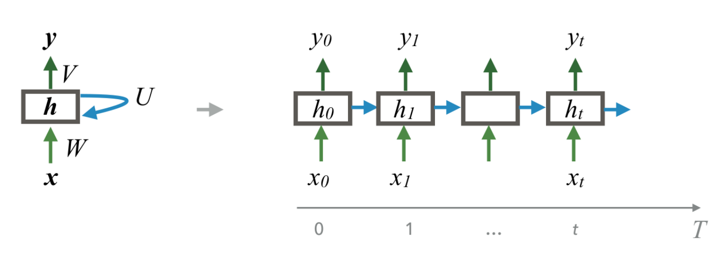
- Possible to think of $\mathbf{h_t}$ as of internal memory.
- In practice, this only works for a couple of steps.
- Gradient either vanishes or explodes during training.



Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Vanishing gradient problem

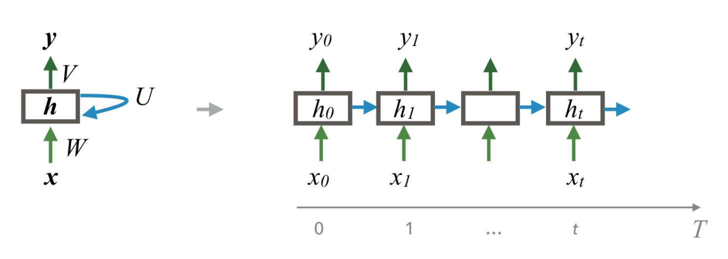
- First explanation of unstable gradients in [Hochreiter, 1991].



Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Vanishing gradient problem

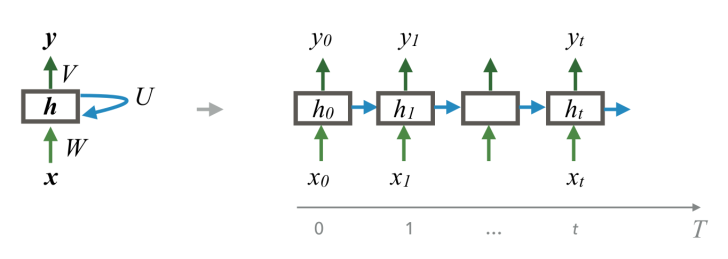
- First explanation of unstable gradients in [Hochreiter, 1991].
- General idea: multiplying by $\frac{d}{dx}\tanh x = 1 - \tanh^2 x \in (0, 1]$.



Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Vanishing gradient problem

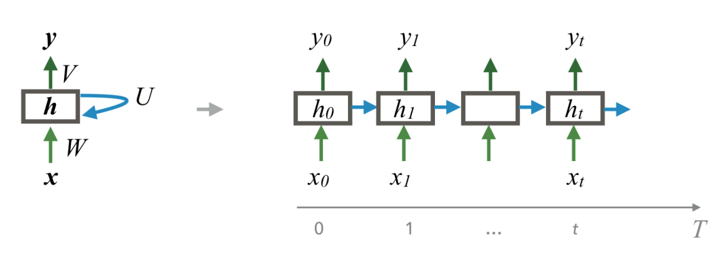
- First explanation of unstable gradients in [Hochreiter, 1991].
- General idea: multiplying by $\frac{d}{dx}\tanh x = 1 - \tanh^2 x \in (0, 1]$.
- Formal argument: based on eigenvalues.



Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Vanishing gradient problem

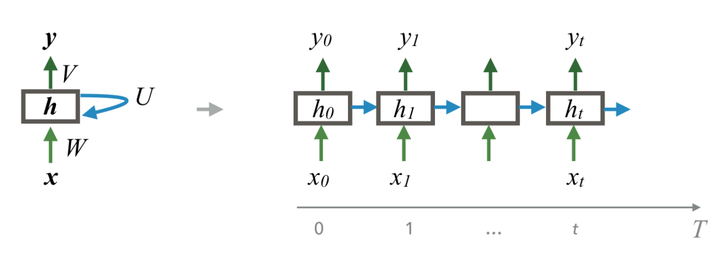
- First explanation of unstable gradients in [Hochreiter, 1991].
- General idea: multiplying by $\frac{d}{dx}\tanh x = 1 - \tanh^2 x \in (0, 1]$.
- Formal argument: based on eigenvalues.
- Vanilla RNNs are inherently unstable in training.



Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Vanishing gradient problem

- First explanation of unstable gradients in [Hochreiter, 1991].
- General idea: multiplying by $\frac{d}{dx}\tanh x = 1 - \tanh^2 x \in (0, 1]$.
- Formal argument: based on eigenvalues.
- Vanilla RNNs are inherently unstable in training.
- Memory is limited to < 10 steps.



Source: Gakhov, A., *Recurrent Neural Networks. Part 1: Theory*

Long-short term memory

- Specifically design an architecture to circumvent vanishing gradients [Hochreiter and Schmidhuber, 1997].

Long-short term memory

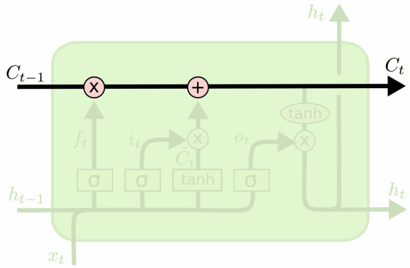
- Specifically design an architecture to circumvent vanishing gradients [Hochreiter and Schmidhuber, 1997].
- General idea: additive interactions transport gradients better.

Long-short term memory

- Specifically design an architecture to circumvent vanishing gradients [Hochreiter and Schmidhuber, 1997].
- General idea: additive interactions transport gradients better.
- Add a separate state cell $\mathbf{c_t}$.

Long-short term memory

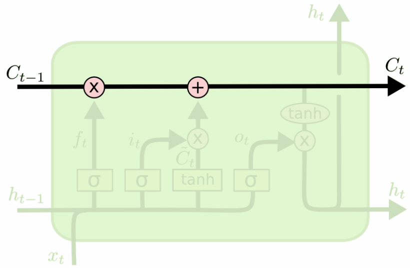
- Specifically design an architecture to circumvent vanishing gradients [Hochreiter and Schmidhuber, 1997].
- General idea: additive interactions transport gradients better.
- Add a separate state cell $\mathbf{c_t}$.



Source: Olah, C., *Understanding LSTM Networks*

Long-short term memory

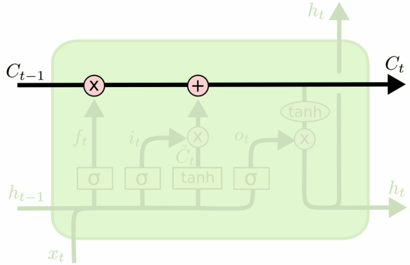
- In practice, this works relatively well (text classification, translation etc.).



Source: Olah, C., *Understanding LSTM Networks*

Long-short term memory

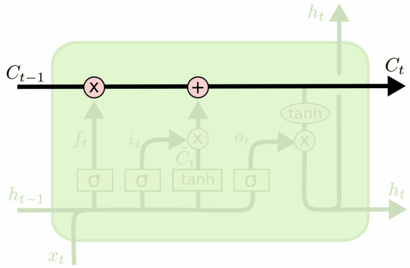
- In practice, this works relatively well (text classification, translation etc.).
- Memory persists for ≈ 100 steps.



Source: Olah, C., *Understanding LSTM Networks*

Long-short term memory

- In practice, this works relatively well (text classification, translation etc.).
- Memory persists for ≈ 100 steps.
- State cell was not designed as memory in traditional sense.



Source: Olah, C., *Understanding LSTM Networks*

Limits of LSTMs

Despite their undeniable success, LSTMs suffer from a number of limitations:

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

Limits of LSTMs

Despite their undeniable success, LSTMs suffer from a number of limitations:

1. 100 steps is not how human memory works.

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

Limits of LSTMs

Despite their undeniable success, LSTMs suffer from a number of limitations:

1. 100 steps is not how human memory works.
2. In practice, hidden state $\mathbf{h_t}$ is modified at each time step.

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

Limits of LSTMs

Despite their undeniable success, LSTMs suffer from a number of limitations:

1. 100 steps is not how human memory works.
2. In practice, hidden state $\mathbf{h_t}$ is modified at each time step.
3. Increasing the size of memory is equivalent to expanding the vector $\mathbf{h_t}$ and the whole network. No. of weights grows at least linearly with required memory.

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

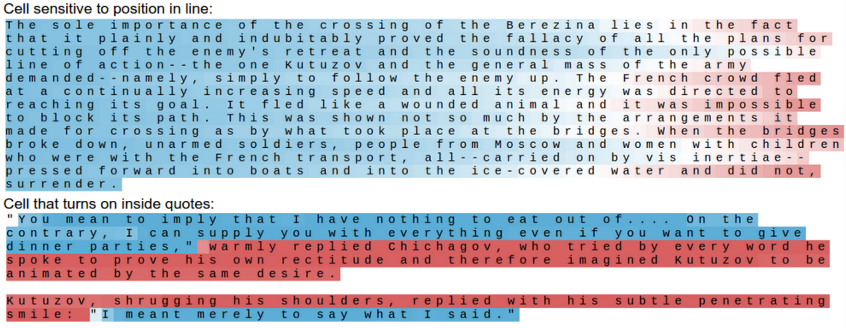
Limits of LSTMs

Despite their undeniable success, LSTMs suffer from a number of limitations:

1. 100 steps is not how human memory works.
2. In practice, hidden state $\mathbf{h_t}$ is modified at each time step.
3. Increasing the size of memory is equivalent to expanding the vector $\mathbf{h_t}$ and the whole network. No. of weights grows at least linearly with required memory.
4. Memory might become "hard-coded." Specific parts of the network might be used to detect given features. Location and content are intertwined.

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

Memory/location entaglement



Source: Karpathy, A., *The Unreasonable Effectiveness of Recurrent Neural Networks*

Networks with external memory

Adding an external memory source mitigates some of the mentioned problems:

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

Networks with external memory

Adding an external memory source mitigates some of the mentioned problems:

1. Not all of the memory is interacted with all the time. Specific parts of memory are accessed at each time step. Memory is "protected."

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

Networks with external memory

Adding an external memory source mitigates some of the mentioned problems:

1. Not all of the memory is interacted with all the time. Specific parts of memory are accessed at each time step. Memory is "protected."
2. Computational cost is not necesarily scaling up with the size of the memory. In theory, memory can be very large. Analogy: increase amount of RAM without changing the CPU.

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

Networks with external memory

Adding an external memory source mitigates some of the mentioned problems:

1. Not all of the memory is interacted with all the time. Specific parts of memory are accessed at each time step. Memory is "protected."
2. Computational cost is not necesarily scaling up with the size of the memory. In theory, memory can be very large. Analogy: increase amount of RAM without changing the CPU.
3. Content is separated out from location. Computation separated from memory.

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

Networks with external memory

Adding an external memory source mitigates some of the mentioned problems:

1. Not all of the memory is interacted with all the time. Specific parts of memory are accessed at each time step. Memory is "protected."
2. Computational cost is not necesarily scaling up with the size of the memory. In theory, memory can be very large. Analogy: increase amount of RAM without changing the CPU.
3. Content is separated out from location. Computation separated from memory.
4. Easier to deal with variables, linked lists, etc. Abstraction comes in handy.

Source: Graves, A., *IJCNN 2017 Plenary Talk: Frontiers in Recurrent Neural Network Research*

Differentiable neural computer

Cast abstract idea in concrete architecture: Differentiable neural computer (DNC) [Graves et al., 2016]. Design principles:

Differentiable neural computer

Cast abstract idea in concrete architecture: Differentiable neural computer (DNC) [Graves et al., 2016]. Design principles:

- Differentiable end-to-end.

Differentiable neural computer

Cast abstract idea in concrete architecture: Differentiable neural computer (DNC) [Graves et al., 2016]. Design principles:

- Differentiable end-to-end.
- Read-write memory.

Differentiable neural computer

Cast abstract idea in concrete architecture: Differentiable neural computer (DNC) [Graves et al., 2016]. Design principles:

- Differentiable end-to-end.
- Read-write memory.

Overview of DNC:

Differentiable neural computer

Cast abstract idea in concrete architecture: Differentiable neural computer (DNC) [Graves et al., 2016]. Design principles:

- Differentiable end-to-end.
- Read-write memory.

Overview of DNC:

(a) Controller - neural network, e.g. deep LSTM.

Differentiable neural computer

Cast abstract idea in concrete architecture: Differentiable neural computer (DNC) [Graves et al., 2016]. Design principles:

- Differentiable end-to-end.
- Read-write memory.

Overview of DNC:

(a) Controller - neural network, e.g. deep LSTM.

(b) Read and write heads.

Differentiable neural computer

Cast abstract idea in concrete architecture: Differentiable neural computer (DNC) [Graves et al., 2016]. Design principles:

- Differentiable end-to-end.
- Read-write memory.

Overview of DNC:

(a) Controller - neural network, e.g. deep LSTM.
(b) Read and write heads.
(c) Memory matrix.

Differentiable neural computer

Cast abstract idea in concrete architecture: Differentiable neural computer (DNC) [Graves et al., 2016]. Design principles:

- Differentiable end-to-end.
- Read-write memory.

Overview of DNC:

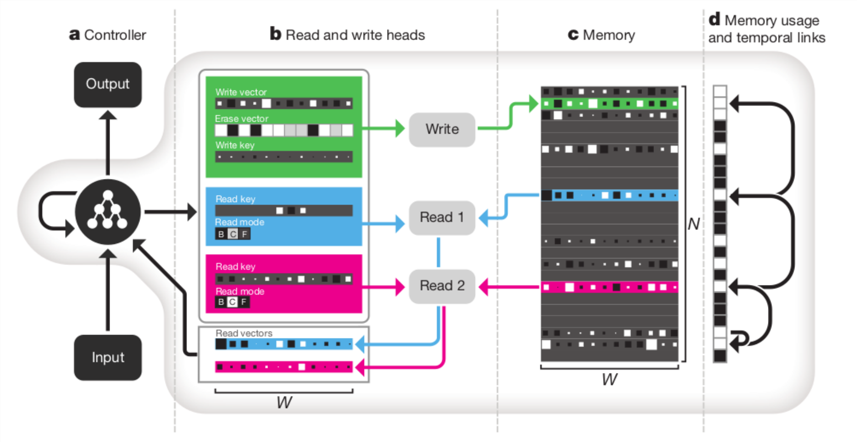
(a) Controller - neural network, e.g. deep LSTM.

(b) Read and write heads.

(c) Memory matrix.

(d) Memory usage vector and temporal link matrix.

Differentiable neural computer



Source: [Graves et al., 2016]

Controller

Neural network $\mathcal{N}$. Let's use a deep LSTM architecture, which carries a hidden state vector $\begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix}$.

Input:

Controller

Neural network $\mathcal{N}$. Let's use a deep LSTM architecture, which carries a hidden state vector $\begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix}$.
Input:

- External input $\mathbf{x_t} \in \mathbb{R}^X$.

Controller

Neural network $\mathcal{N}$. Let's use a deep LSTM architecture, which carries a hidden state vector $\begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix}$.

Input:

- External input $\mathbf{x_t} \in \mathbb{R}^X$.
- R read vectors $\mathbf{r}_{t-1}^1, \dots, \mathbf{r}_{t-1}^R \in M_{t-1} \in \mathbb{R}^{N \times W}$.

Controller

Neural network $\mathcal{N}$. Let's use a deep LSTM architecture, which carries a hidden state vector $\begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix}$.
Input:

- External input $\mathbf{x_t} \in \mathbb{R}^X$.
- R read vectors $\mathbf{r}_{t-1}^1, \dots, \mathbf{r}_{t-1}^R \in M_{t-1} \in \mathbb{R}^{N \times W}$.

Output:

Controller

Neural network $\mathcal{N}$. Let's use a deep LSTM architecture, which carries a hidden state vector $\begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix}$.
Input:

- External input $\mathbf{x_t} \in \mathbb{R}^X$.
- R read vectors $\mathbf{r}_{t-1}^1, \dots, \mathbf{r}_{t-1}^R \in M_{t-1} \in \mathbb{R}^{N \times W}$.

Output:

- Controller output vector $\mathbf{v_t} = W_y \begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix} \in \mathbb{R}^Y$.

Controller

Neural network $\mathcal{N}$. Let's use a deep LSTM architecture, which carries a hidden state vector $\begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix}$.

Input:

- External input $\mathbf{x_t} \in \mathbb{R}^X$.
- R read vectors $\mathbf{r}_{t-1}^1, \dots, \mathbf{r}_{t-1}^R \in M_{t-1} \in \mathbb{R}^{N \times W}$.

Output:

- Controller output vector $\mathbf{v_t} = W_y \begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix} \in \mathbb{R}^Y$.
- Interface vector $\hat{\xi}_\mathbf{t} = W_\xi \begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix} \in \mathbb{R}^{(W \times R) + 3W + 5R + 3}$.

Controller

Neural network $\mathcal{N}$. Let's use a deep LSTM architecture, which carries a hidden state vector $\begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix}$.

Input:

- External input $\mathbf{x_t} \in \mathbb{R}^X$.
- R read vectors $\mathbf{r}_{t-1}^1, \dots, \mathbf{r}_{t-1}^R \in M_{t-1} \in \mathbb{R}^{N \times W}$.

Output:

- Controller output vector $\mathbf{v_t} = W_y \begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix} \in \mathbb{R}^Y$.
- Interface vector $\hat{\xi}_\mathbf{t} = W_\xi \begin{bmatrix} \mathbf{h}_t^1 & \dots & \mathbf{h}_t^L \end{bmatrix} \in \mathbb{R}^{(W \times R) + 3W + 5R + 3}$.
- Memory-augmented output vector $\mathbf{y_t} = \mathbf{v_t} + W_r \begin{bmatrix} \mathbf{r}_t^1 & \dots & \mathbf{r}_t^R \end{bmatrix} \in \mathbb{R}^Y$.

Interface vector

- Interface vector before processing $\hat{\xi}_{\mathbf{t}} = \left[\mathbf{k}_t^{r,1} \dots \mathbf{k}_t^{r,R}; \hat{\beta}_t^{r,1} \dots \hat{\beta}_t^{r,R}; \mathbf{k}_t^{w}; \hat{\beta}_t^{w}; \hat{\mathbf{e}}_t; \mathbf{v}_t; \hat{f}_t^1 \dots \hat{f}_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1 \dots \hat{\pi}_t^R\right]$

Interface vector

- Interface vector before processing $\hat{\xi}_{\mathbf{t}} = \left[\mathbf{k}_t^{r,1} \ldots \mathbf{k}_t^{r,R}; \hat{\beta}_t^{r,1} \ldots \hat{\beta}_t^{r,R}; \mathbf{k}_t^w; \hat{\beta}_t^w; \hat{\mathbf{e}}_t; \mathbf{v}_t; \hat{f}_t^1 \ldots \hat{f}_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1 \ldots \hat{\pi}_t^R\right]$
- Define: $\text{oneplus}(x) = 1 + \ln(1 + e^x) \in [1, \infty)$.

Interface vector

- Interface vector before processing $\hat{\xi}_{\mathbf{t}} = \left[\mathbf{k}_t^{r,1} \dots \mathbf{k}_t^{r,R}; \hat{\beta}_t^{r,1} \dots \hat{\beta}_t^{r,R}; \mathbf{k}_t^{w}; \hat{\beta}_t^{w}; \hat{\mathbf{e}}_t; \mathbf{v}_t; \hat{f}_t^1 \dots \hat{f}_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1 \dots \hat{\pi}_t^R\right]$
- Define: $\text{oneplus}(x) = 1 + \ln(1 + e^x) \in [1, \infty)$.
- Define: $\text{softmax}(\mathbf{x})_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$.

Interface vector

- Interface vector before processing $\hat{\xi}_{\mathbf{t}} = \left[\mathbf{k}_t^{r,1} \dots \mathbf{k}_t^{r,R}; \hat{\beta}_t^{r,1} \dots \hat{\beta}_t^{r,R}; \mathbf{k}_t^{w}; \hat{\beta}_t^{w}; \hat{\mathbf{e}}_t; \mathbf{v}_t; \hat{f}_t^1 \dots \hat{f}_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1 \dots \hat{\pi}_t^R\right]$
- Define: $\text{oneplus}(x) = 1 + \ln(1 + e^x) \in [1, \infty)$.
- Define: $\text{softmax}(\mathbf{x})_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$.
- $\beta_t^{r,i} = \text{oneplus}(\hat{\beta}_t^{r,i}), \beta_t^w = \text{oneplus}(\hat{\beta}_t^w)$

Interface vector

- Interface vector before processing $\hat{\xi}_{\mathbf{t}} = \left[\mathbf{k}_t^{r,1} \dots \mathbf{k}_t^{r,R}; \hat{\beta}_t^{r,1} \dots \hat{\beta}_t^{r,R}; \mathbf{k}_t^{w}; \hat{\beta}_t^{w}; \hat{\mathbf{e}}_t; \mathbf{v}_t; \hat{f}_t^1 \dots \hat{f}_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1 \dots \hat{\pi}_t^R\right]$
- Define: $\text{oneplus}(x) = 1 + \ln\left(1 + e^x\right) \in [1, \infty)$.
- Define: $\text{softmax}(\mathbf{x})_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$.
- $\beta_t^{r,i} = \text{oneplus}(\hat{\beta}_t^{r,i}), \beta_t^w = \text{oneplus}(\hat{\beta}_t^w)$
- $\mathbf{e}_t = \sigma(\hat{\mathbf{e}}_t), f_t^i = \sigma(\hat{f}_t^i), g_t^a = \sigma(\hat{g}_t^a), g_t^w = \sigma(\hat{g}_t^w)$

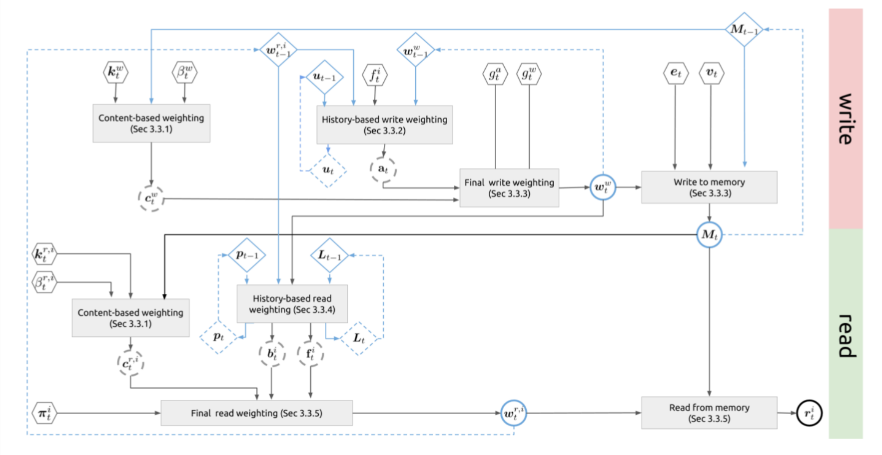
Interface vector

- Interface vector before processing $\hat{\xi}_{\mathbf{t}} = \left[\mathbf{k}_t^{r,1} \dots \mathbf{k}_t^{r,R}; \hat{\beta}_t^{r,1} \dots \hat{\beta}_t^{r,R}; \mathbf{k}_t^{w}; \hat{\beta}_t^{w}; \hat{\mathbf{e}}_t; \mathbf{v}_t; \hat{f}_t^1 \dots \hat{f}_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1 \dots \hat{\pi}_t^R\right]$
- Define: $\text{oneplus}(x) = 1 + \ln(1 + e^x) \in [1, \infty)$.
- Define: $\text{softmax}(\mathbf{x})_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$.
- $\beta_t^{r,i} = \text{oneplus}(\hat{\beta}_t^{r,i}), \beta_t^w = \text{oneplus}(\hat{\beta}_t^w)$
- $\mathbf{e}_t = \sigma(\hat{\mathbf{e}}_t), f_t^i = \sigma(\hat{f}_t^i), g_t^a = \sigma(\hat{g}_t^a), g_t^w = \sigma(\hat{g}_t^w)$
- $\pi_t^i = \text{softmax}(\hat{\pi}_t^i)$

Interface vector

- Interface vector before processing $\hat{\xi}_{\mathbf{t}} = \left[\mathbf{k}_t^{r,1} \dots \mathbf{k}_t^{r,R}; \hat{\beta}_t^{r,1} \dots \hat{\beta}_t^{r,R}; \mathbf{k}_t^{w}; \hat{\beta}_t^{w}; \hat{\mathbf{e}}_t; \mathbf{v}_t; \hat{f}_t^1 \dots \hat{f}_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1 \dots \hat{\pi}_t^R\right]$
- Define: $\text{oneplus}(x) = 1 + \ln(1 + e^x) \in [1, \infty)$.
- Define: $\text{softmax}(\mathbf{x})_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$.
- $\beta_t^{r,i} = \text{oneplus}(\hat{\beta}_t^{r,i}), \beta_t^w = \text{oneplus}(\hat{\beta}_t^w)$
- $\mathbf{e}_t = \sigma(\hat{\mathbf{e}}_t), f_t^i = \sigma(\hat{f}_t^i), g_t^a = \sigma(\hat{g}_t^a), g_t^w = \sigma(\hat{g}_t^w)$
- $\pi_t^i = \text{softmax}(\hat{\pi}_t^i)$
- Interface vector after processing $\xi_{\mathbf{t}} = \left[\mathbf{k}_t^{r,1} \dots \mathbf{k}_t^{r,R}; \beta_t^{r,1} \dots \beta_t^{r,R}; \mathbf{k}_t^{w}; \beta_t^{w}; \mathbf{e}_t; \mathbf{v}_t; f_t^1 \dots f_t^R; g_t^a; g_t^w; \pi_t^1 \dots \pi_t^R\right]$

Interacting with memory



Source: Hsin, C., *Implementation and Optimization of Differentiable Neural Computers*

Writing to memory

1. Content-based addressing:

Writing to memory

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

Writing to memory

1. Content-based addressing:
   - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i,\cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j,\cdot])\beta\}}$
   - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$

Writing to memory

1. Content-based addressing:
   - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i,\cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j,\cdot])\beta\}}$
   - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$
   - $\mathbf{c}_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w) \in \mathcal{S}_N$

Writing to memory

1. Content-based addressing:
   - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i,\cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j,\cdot])\beta\}}$
   - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$
   - $\mathbf{c}_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w) \in \mathcal{S}_N$
2. Dynamic memory allocation:

Writing to memory

1. Content-based addressing:
   - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i, \cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j, \cdot])\beta\}}$
   - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$
   - $\mathbf{c}_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w) \in \mathcal{S}_N$
2. Dynamic memory allocation:
   - memory retention vector $\psi_t = \prod_{i=1}^{R} \left(\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{r,i}\right) \in [0, 1]^N$

Writing to memory

1. Content-based addressing:
    - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i,\cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j,\cdot])\beta\}}$
    - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$
    - $\mathbf{c}_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w) \in \mathcal{S}_N$
2. Dynamic memory allocation:
    - memory retention vector $\psi_t = \prod_{i=1}^{R} \left(\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{r,i}\right) \in [0, 1]^N$
    - memory usage vector
      $\mathbf{u}_t = \left(\mathbf{u}_{t-1} + \mathbf{w}_{t-1}^w - \mathbf{u}_{t-1} \circ \mathbf{w}_{t-1}^w\right) \circ \psi_t \in [0, 1]^N$

Writing to memory

1. Content-based addressing:
   - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i,\cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j,\cdot])\beta\}}$
   - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$
   - $\mathbf{c}_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w) \in \mathcal{S}_N$
2. Dynamic memory allocation:
   - memory retention vector $\psi_t = \prod_{i=1}^{R} \left(\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{r,i}\right) \in [0, 1]^N$
   - memory usage vector
     $\mathbf{u}_t = \left(\mathbf{u}_{t-1} + \mathbf{w}_{t-1}^w - \mathbf{u}_{t-1} \circ \mathbf{w}_{t-1}^w\right) \circ \psi_t \in [0, 1]^N$
   - sort indices of memory locations in ascending order of usage, $\phi_t \in \mathbb{N}^+, \phi_t[1]$ is the least used location

Writing to memory

1. Content-based addressing:
   - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i,\cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j,\cdot])\beta\}}$
   - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$
   - $\mathbf{c}_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w) \in \mathcal{S}_N$
2. Dynamic memory allocation:
   - memory retention vector $\psi_t = \prod_{i=1}^{R} \left(\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{r,i}\right) \in [0, 1]^N$
   - memory usage vector
     $\mathbf{u}_t = \left(\mathbf{u}_{t-1} + \mathbf{w}_{t-1}^w - \mathbf{u}_{t-1} \circ \mathbf{w}_{t-1}^w\right) \circ \psi_t \in [0, 1]^N$
   - sort indices of memory locations in ascending order of usage,
     $\phi_t \in \mathbb{N}^+, \phi_t[1]$ is the least used location
   - allocation weighting
     $\mathbf{a}_t[\phi_t[j]] = (1 - \mathbf{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \mathbf{u}_t[\phi_t[i]] \in \Delta_N$

Writing to memory

1. Content-based addressing:
   - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i,\cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j,\cdot])\beta\}}$
   - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$
   - $\mathbf{c}_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w) \in \mathcal{S}_N$
2. Dynamic memory allocation:
   - memory retention vector $\psi_t = \prod_{i=1}^{R} \left(\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{r,i}\right) \in [0, 1]^N$
   - memory usage vector
     $\mathbf{u}_t = \left(\mathbf{u}_{t-1} + \mathbf{w}_{t-1}^w - \mathbf{u}_{t-1} \circ \mathbf{w}_{t-1}^w\right) \circ \psi_t \in [0, 1]^N$
   - sort indices of memory locations in ascending order of usage, $\phi_t \in \mathbb{N}^+, \phi_t[1]$ is the least used location
   - allocation weighting
     $\mathbf{a}_t[\phi_t[j]] = (1 - \mathbf{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \mathbf{u}_t[\phi_t[i]] \in \Delta_N$
3. Write weighting:

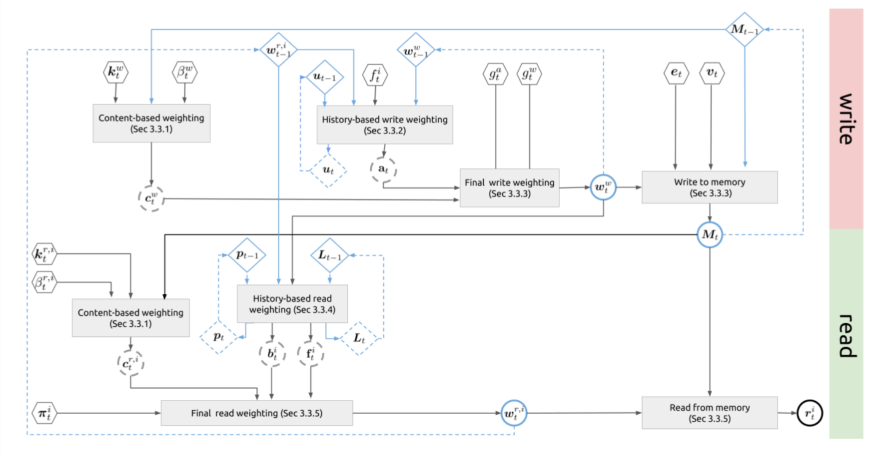
Writing to memory

1. Content-based addressing:
   - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i,\cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j,\cdot])\beta\}}$
   - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$
   - $\mathbf{c}_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w) \in \mathcal{S}_N$
2. Dynamic memory allocation:
   - memory retention vector $\psi_t = \prod_{i=1}^{R} \left(\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{r,i}\right) \in [0, 1]^N$
   - memory usage vector
     $\mathbf{u}_t = \left(\mathbf{u}_{t-1} + \mathbf{w}_{t-1}^w - \mathbf{u}_{t-1} \circ \mathbf{w}_{t-1}^w\right) \circ \psi_t \in [0, 1]^N$
   - sort indices of memory locations in ascending order of usage,
     $\phi_t \in \mathbb{N}^+, \phi_t[1]$ is the least used location
   - allocation weighting
     $\mathbf{a}_t[\phi_t[j]] = (1 - \mathbf{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \mathbf{u}_t[\phi_t[i]] \in \Delta_N$
3. Write weighting:
   - $\mathbf{w}_t^w = g_t^w \left[g_t^a \mathbf{a}_t + (1 - g_t^a)\mathbf{c}_t^w\right] \in \Delta_N$

Writing to memory

1. Content-based addressing:
   - $\mathcal{C}(M, \mathbf{k}, \beta)[i] = \frac{\exp\{\mathcal{D}(\mathbf{k}, M[i,\cdot])\beta\}}{\sum_j \exp\{\mathcal{D}(\mathbf{k}, M[j,\cdot])\beta\}}$
   - cosine similarity $\mathcal{D}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}||\mathbf{v}|} \in [-1, 1]$
   - $\mathbf{c}_t^w = \mathcal{C}(M_{t-1}, \mathbf{k}_t^w, \beta_t^w) \in \mathcal{S}_N$
2. Dynamic memory allocation:
   - memory retention vector $\psi_t = \prod_{i=1}^{R} \left(\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{r,i}\right) \in [0, 1]^N$
   - memory usage vector
     $\mathbf{u}_t = \left(\mathbf{u}_{t-1} + \mathbf{w}_{t-1}^w - \mathbf{u}_{t-1} \circ \mathbf{w}_{t-1}^w\right) \circ \psi_t \in [0, 1]^N$
   - sort indices of memory locations in ascending order of usage,
     $\phi_t \in \mathbb{N}^+, \phi_t[1]$ is the least used location
   - allocation weighting
     $\mathbf{a}_t[\phi_t[j]] = (1 - \mathbf{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \mathbf{u}_t[\phi_t[i]] \in \Delta_N$
3. Write weighting:
   - $\mathbf{w}_t^w = g_t^w \left[g_t^a \mathbf{a}_t + (1 - g_t^a)\mathbf{c}_t^w\right] \in \Delta_N$
4. Actual write operation: $M_t = M_{t-1} \circ (E - \mathbf{w}_t^w \mathbf{e}_t^T) + \mathbf{w}_t^w \mathbf{v}_t^T$

Interacting with memory



Source: Hsin, C., *Implementation and Optimization of Differentiable Neural Computers*

Reading from memory

1. Content-based addressing:

Reading from memory

1. Content-based addressing:
   - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$

Reading from memory

1. Content-based addressing:
   - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$
2. Temporal memory linkage:

Reading from memory

1. Content-based addressing:
   - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$
2. Temporal memory linkage:
   - temporal link matrix $L_t \in [0,1]^{N \times N}, L_t[i,\cdot] \in \Delta_N, L_t[\cdot,j] \in \Delta_N$

Reading from memory

1. Content-based addressing:
    - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$
2. Temporal memory linkage:
    - temporal link matrix $L_t \in [0,1]^{N \times N}, L_t[i,\cdot] \in \Delta_N, L_t[\cdot, j] \in \Delta_N$
    - precedence weighting $\mathbf{p}_t = \left(1 - \sum_i \mathbf{w}_t^w[i]\right) \mathbf{p}_{t-1} + \mathbf{w}_t^w \in \Delta_N$

Reading from memory

1. Content-based addressing:
   - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$
2. Temporal memory linkage:
   - temporal link matrix $L_t \in [0,1]^{N \times N}, L_t[i, \cdot] \in \Delta_N, L_t[\cdot, j] \in \Delta_N$
   - precedence weighting $\mathbf{p}_t = \left(1 - \sum_i \mathbf{w}_t^w[i]\right) \mathbf{p}_{t-1} + \mathbf{w}_t^w \in \Delta_N$
   - linkage logic: $\forall i : L_t[i,i] = 0, L_t[i,j] = \left(1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j]\right) L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$

Reading from memory

1. Content-based addressing:
   - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$
2. Temporal memory linkage:
   - temporal link matrix $L_t \in [0,1]^{N \times N}, L_t[i,\cdot] \in \Delta_N, L_t[\cdot, j] \in \Delta_N$
   - precedence weighting $\mathbf{p}_t = (1 - \sum_i \mathbf{w}_t^w[i])\, \mathbf{p}_{t-1} + \mathbf{w}_t^w \in \Delta_N$
   - linkage logic: $\forall i : L_t[i,i] = 0, L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])\, L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$
   - forward weighting: $\mathbf{f}_t^i = L_t \mathbf{w}_{t-1}^{r,i} \in \Delta_N$

Reading from memory

1. Content-based addressing:
   - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$
2. Temporal memory linkage:
   - temporal link matrix $L_t \in [0,1]^{N \times N}, L_t[i,\cdot] \in \Delta_N, L_t[\cdot, j] \in \Delta_N$
   - precedence weighting $\mathbf{p}_t = \left(1 - \sum_i \mathbf{w}_t^w[i]\right) \mathbf{p}_{t-1} + \mathbf{w}_t^w \in \Delta_N$
   - linkage logic: $\forall i : L_t[i,i] = 0, L_t[i,j] = \left(1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j]\right) L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$
   - forward weighting: $\mathbf{f}_t^i = L_t \mathbf{w}_{t-1}^{r,i} \in \Delta_N$
   - backward weighting: $\mathbf{b}_t^i = L_t^T \mathbf{w}_{t-1}^{r,i} \in \Delta_N$

Reading from memory

1. Content-based addressing:
   - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$
2. Temporal memory linkage:
   - temporal link matrix $L_t \in [0,1]^{N \times N}, L_t[i,\cdot] \in \Delta_N, L_t[\cdot, j] \in \Delta_N$
   - precedence weighting $\mathbf{p}_t = (1 - \sum_i \mathbf{w}_t^w[i])\, \mathbf{p}_{t-1} + \mathbf{w}_t^w \in \Delta_N$
   - linkage logic: $\forall i : L_t[i,i] = 0, L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])\, L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$
   - forward weighting: $\mathbf{f}_t^i = L_t \mathbf{w}_{t-1}^{r,i} \in \Delta_N$
   - backward weighting: $\mathbf{b}_t^i = L_t^T \mathbf{w}_{t-1}^{r,i} \in \Delta_N$
3. Read weighting:

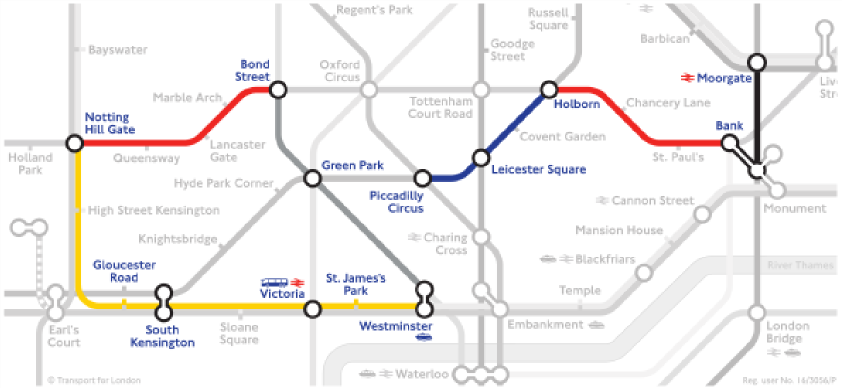
Reading from memory

1. Content-based addressing:
   - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$
2. Temporal memory linkage:
   - temporal link matrix $L_t \in [0,1]^{N\times N}, L_t[i,\cdot] \in \Delta_N, L_t[\cdot, j] \in \Delta_N$
   - precedence weighting $\mathbf{p}_t = (1 - \sum_i \mathbf{w}_t^w[i])\, \mathbf{p}_{t-1} + \mathbf{w}_t^w \in \Delta_N$
   - linkage logic: $\forall i : L_t[i,i] = 0, L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])\, L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$
   - forward weighting: $\mathbf{f}_t^i = L_t \mathbf{w}_{t-1}^{r,i} \in \Delta_N$
   - backward weighting: $\mathbf{b}_t^i = L_t^T \mathbf{w}_{t-1}^{r,i} \in \Delta_N$
3. Read weighting:
   - $\mathbf{w}_t^{r,i} = \pi_t^i[1]\mathbf{b}_t^i + \pi_t^i[2]\mathbf{c}_t^{r,i} + \pi_t^i[3]\mathbf{f}_t^i \in \Delta_N$

Reading from memory

1. Content-based addressing:
   - $\mathbf{c}_t^{r,i} = \mathcal{C}(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}) \in \mathcal{S}_N$
2. Temporal memory linkage:
   - temporal link matrix $L_t \in [0,1]^{N\times N}, L_t[i,\cdot] \in \Delta_N, L_t[\cdot,j] \in \Delta_N$
   - precedence weighting $\mathbf{p}_t = (1 - \sum_i \mathbf{w}_t^w[i])\,\mathbf{p}_{t-1} + \mathbf{w}_t^w \in \Delta_N$
   - linkage logic: $\forall i : L_t[i,i] = 0, L_t[i,j] =$
     $(1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])\,L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$
   - forward weighting: $\mathbf{f}_t^i = L_t \mathbf{w}_{t-1}^{r,i} \in \Delta_N$
   - backward weighting: $\mathbf{b}_t^i = L_t^T \mathbf{w}_{t-1}^{r,i} \in \Delta_N$
3. Read weighting:
   - $\mathbf{w}_t^{r,i} = \pi_t^i[1]\mathbf{b}_t^i + \pi_t^i[2]\mathbf{c}_t^{r,i} + \pi_t^i[3]\mathbf{f}_t^i \in \Delta_N$
4. Actual read operation: $\mathbf{r}_t^i = M_t^T \mathbf{w}_t^{r,i}$.

Traversing London Underground



Source: [Graves et al., 2016]

Traversing London Underground

- London Underground as a graph.

Traversing London Underground

- London Underground as a graph.
- Explicit vector representation of an edge:
  $\begin{bmatrix} \text{station}_1 & \text{station}_2 & \text{line} \end{bmatrix}$

Traversing London Underground

- London Underground as a graph.
- Explicit vector representation of an edge:
  $\begin{bmatrix} \text{station}_1 & \text{station}_2 & \text{line} \end{bmatrix}$
- Queries: traversal, shortest path.

Traversing London Underground

- London Underground as a graph.
- Explicit vector representation of an edge:
  $\begin{bmatrix} \text{station}_1 & \text{station}_2 & \text{line} \end{bmatrix}$
- Queries: traversal, shortest path.
- Training: graphs with random nodes and connections.

Traversing London Underground

- London Underground as a graph.
- Explicit vector representation of an edge:
  $\begin{bmatrix} \text{station}_1 & \text{station}_2 & \text{line} \end{bmatrix}$
- Queries: traversal, shortest path.
- Training: graphs with random nodes and connections.
- Curriculum learning with increasing complexity of graphs and queries.

Traversing London Underground

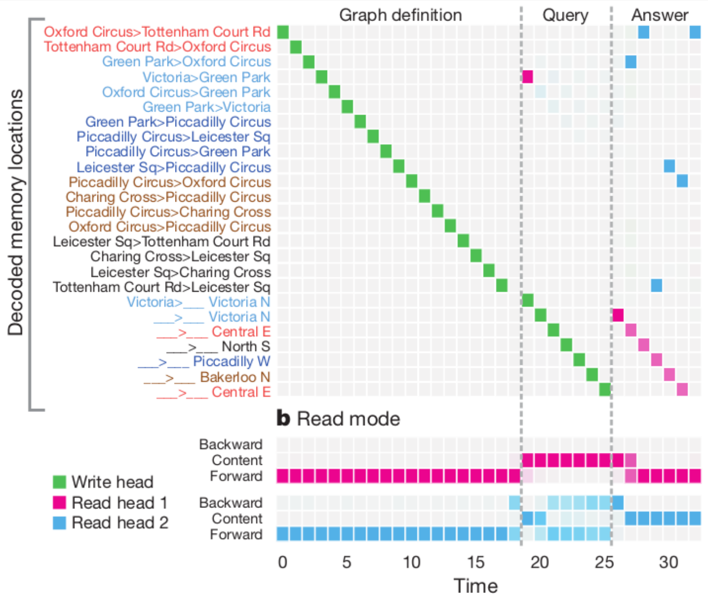
- London Underground as a graph.
- Explicit vector representation of an edge:
  $\begin{bmatrix} \text{station}_1 & \text{station}_2 & \text{line} \end{bmatrix}$
- Queries: traversal, shortest path.
- Training: graphs with random nodes and connections.
- Curriculum learning with increasing complexity of graphs and queries.
- Tested without re-training on the London Underground graph.

Traversal/shortest path

Traversal	Shortest-path
Traversal question: (BondSt, _, Central), (_, _, Circle), (_, _, Circle), (_, _, Circle), (_, _, Circle), (_, _, Jubilee), (_, _, Jubilee),	**Shortest-path question:** (Moorgate, PiccadillyCircus, _)
Answer: (BondSt, NottingHillGate, Central) (NottingHillGate, GloucesterRd, Circle) ⋮ (Westminster, GreenPark, Jubilee) (GreenPark, BondSt, Jubilee)	**Answer:** (Moorgate, Bank, Northern) (Bank, Holborn, Central) (Holborn, LeicesterSq, Piccadilly) (LeicesterSq, PiccadillyCircus, Piccadilly)

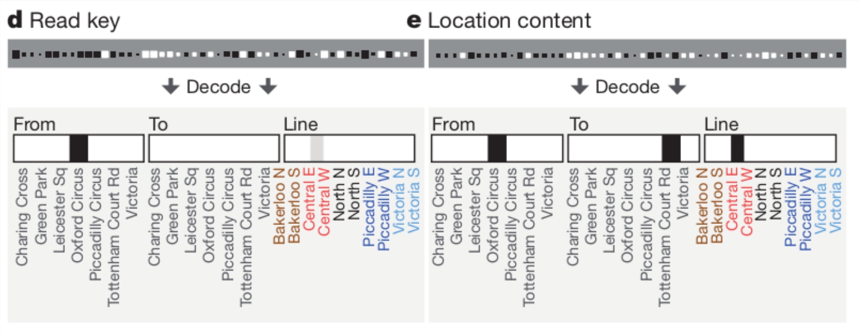
Source: [Graves et al., 2016]

Traversal



Graph definition
Query
Answer
Decoded memory locations
Oxford Circus>Tottenham Court Rd
Tottenham Court Rd>Oxford Circus
Green Park>Oxford Circus
Victoria>Green Park
Oxford Circus>Green Park
Green Park>Victoria
Green Park>Piccadilly Circus
Piccadilly Circus>Leicester Sq
Piccadilly Circus>Green Park
Leicester Sq>Piccadilly Circus
Piccadilly Circus>Oxford Circus
Charing Cross>Piccadilly Circus
Piccadilly Circus>Charing Cross
Oxford Circus>Piccadilly Circus
Leicester Sq>Tottenham Court Rd
Charing Cross>Leicester Sq
Leicester Sq>Charing Cross
Tottenham Court Rd>Leicester Sq
Victoria>___ Victoria N
___>___ Victoria N
___>___ Central E
___>___ North S
___>___ Piccadilly W
___>___ Bakerloo N
___>___ Central E
b Read mode
Write head
Read head 1
Read head 2
Backward
Content
Forward
Backward
Content
Forward
0
5
10
15
20
25
30
Time

Traversal



Source: [Graves et al., 2016]

Further research

- Synthetic gradients [Jaderberg et al., 2016].

Further research

- Synthetic gradients [Jaderberg et al., 2016].
- Speed up training.

Further research

- Synthetic gradients [Jaderberg et al., 2016].
- Speed up training.
- DNC with other types of neural networks.

Further research

- Synthetic gradients [Jaderberg et al., 2016].
- Speed up training.
- DNC with other types of neural networks.
- Scale up.

Further research

- Synthetic gradients [Jaderberg et al., 2016].
- Speed up training.
- DNC with other types of neural networks.
- Scale up.
- Tasks beyond graphs.

Graves, A., Wayne, G., et al. (2016).
Hybrid computing using a neural network with dynamic external memory.
Nature, 538:471–476.

Hochreiter, S. (1991).
Untersuchungen zu dynamischen neuronalen netzen.
Diploma thesis, Technical University Munich.

Hochreiter, S. and Schmidhuber, J. (1997).
Long short-term memory.
Neural Computation, 9(8):1735–1780.

Jaderberg, M., Czarnecki, W. M., et al. (2016).
Decoupled neural interfaces using synthetic gradients.
arXiv.