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).
Rec current 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.

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

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

- Classic RNN architecture.

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

- Possible to think of \( h_t \) as internal memory.

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

- Classic RNN architecture.

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

- Possible to think of \( h_t \) as 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.

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

- Possible to think of \( h_t \) 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
• Specifically design an architecture to circumvent vanishing gradients [Hochreiter and Schmidhuber, 1997].
• 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 $c_t$. 
- Specifically design an architecture to circumvent vanishing gradients [Hochreiter and Schmidhuber, 1997].
- General idea: additive interactions transport gradients better.
- Add a separate state cell $c_t$.

Source: Olah, C., *Understanding LSTM Networks*
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 \( \approx 100 \) steps.

Source: Olah, C., *Understanding LSTM Networks*
Long-short term memory

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

Source: Olah, C., *Understanding LSTM Networks*
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 $h_t$ is modified at each time step.
3. Increasing the size of memory is equivalent to expanding the vector $h_t$ 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*
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 $h_t$ 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 $h_t$ is modified at each time step.
3. Increasing the size of memory is equivalent to expanding the vector $h_t$ 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
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 $h_t$ is modified at each time step.
3. Increasing the size of memory is equivalent to expanding the vector $h_t$ 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
Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action—the one Kutuzov and the general mass of the army demanded—namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all—carried on by vis inertiae—pressed forward into boats and into the ice-covered water and did not surrender.

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Source: Karpathy, A., *The Unreasonable Effectiveness of Recurrent Neural Networks*
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 necessarily 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*
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."
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 necessarily 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*
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 necessarily 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*
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 necessarily 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
Cast abstract idea in concrete architecture: Differentiable neural computer (DNC) [Graves et al., 2016]. Design principles:
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.
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.
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]
Neural network $\mathcal{N}$. Let’s use a deep LSTM architecture, which carries a hidden state vector $[h^1_t \ldots h^L_t]$.

Input:
Controller

Neural network $\mathcal{N}$. Let’s use a deep LSTM architecture, which carries a hidden state vector $[h^1_t \ldots h^L_t]$.

Input:

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

Output:

- Controller output vector $v_t = W_y [h^1_t \ldots h^L_t] \in \mathbb{R}^Y$.
- Interface vector $\hat{\xi}_t = W_\xi [h^1_t \ldots h^L_t] \in \mathbb{R}^{(W \times R) + 3W + 5R + 3}$.
- Memory-augmented output vector $y_t = v_t + W_r [r_1^t \ldots r_R^t] \in \mathbb{R}^Y$. 
Controller

Neural network $\mathcal{N}$. Let’s use a deep LSTM architecture, which carries a hidden state vector $[h_1^t \ldots h_L^t]$.

Input:

- External input $x_t \in \mathbb{R}^X$.
- $R$ read vectors $r_1^{t-1}, \ldots, r_R^{t-1} \in M_{t-1} \in \mathbb{R}^{N \times W}$. 

Output:

- Controller output vector $v_t = W_y [h_1^t \ldots h_L^t] \in \mathbb{R}^Y$.
- Interface vector $\hat{\xi}_t = W_\xi [h_1^t \ldots h_L^t] \in \mathbb{R}^{(W \times R) + 3W + 5R + 3}$.
- Memory-augmented output vector $y_t = v_t + W_r [r_1^{t-1} \ldots r_R^{t-1}] \in \mathbb{R}^Y$. 

Controller

Neural network $\mathcal{N}$. Let’s use a deep LSTM architecture, which carries a hidden state vector $[h^1_t \ldots h^L_t]$.

Input:

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

Output:
Controller

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

Input:

- External input $x_t \in \mathbb{R}^X$.
- $R$ read vectors $r^1_{t-1}, \ldots, r^R_{t-1} \in \mathcal{M}_{t-1} \in \mathbb{R}^{N \times W}$.

Output:

- Controller output vector $v_t = W_y \begin{bmatrix} h^1_t & \ldots & h^L_t \end{bmatrix} \in \mathbb{R}^Y$. 
Neural network $\mathcal{N}$. Let’s use a deep LSTM architecture, which carries a hidden state vector $\begin{bmatrix} h^1_t & \ldots & h^L_t \end{bmatrix}$.

**Input:**

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

**Output:**

- Controller output vector $v_t = W_y \begin{bmatrix} h^1_t & \ldots & h^L_t \end{bmatrix} \in \mathbb{R}^Y$.
- Interface vector $\hat{\xi}_t = W_\xi \begin{bmatrix} h^1_t & \ldots & h^L_t \end{bmatrix} \in \mathbb{R}^{(W \times R) + 3W + 5R + 3}$. 
Neural network $\mathcal{N}$. Let’s use a deep LSTM architecture, which carries a hidden state vector $[h^1_t \ldots h^L_t]$.

Input:

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

Output:

- Controller output vector $v_t = W_y [h^1_t \ldots h^L_t] \in \mathbb{R}^Y$.
- Interface vector $\hat{\xi}_t = W_\xi [h^1_t \ldots h^L_t] \in \mathbb{R}^{(W \times R) + 3W + 5R + 3}$.
- Memory-augmented output vector $y_t = v_t + W_r [r^1_t \ldots r^R_t] \in \mathbb{R}^Y$. 
Interface vector

- Interface vector before processing \( \hat{\xi}_t = \)
\[
\begin{bmatrix}
k_{t}^{r,1} & \ldots & k_{t}^{r,R} & \hat{\beta}_{t}^{r,1} & \ldots & \hat{\beta}_{t}^{r,R} & k_{t}^{w} & \hat{\beta}_{t}^{w} & \hat{e}_t & \hat{f}_t & \ldots & \hat{f}_R & \hat{g}_t^{a} & \hat{g}_t^{w} & \hat{\pi}_t^{1} & \ldots & \hat{\pi}_t^{R}
\end{bmatrix}
\]
Interface vector

• Interface vector before processing $\hat{\xi}_t = \left[ k_t^r, 1 \ldots k_t^r, R; \hat{\beta}_t^r, 1 \ldots \hat{\beta}_t^r, R; k_t^w; \hat{\beta}_t^w; \hat{e}_t; v_t; f_t^1 \ldots f_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1 \ldots \hat{\pi}_t^R \right]$

• Define: $\text{oneplus}(x) = 1 + \ln (1 + e^x) \in [1, \infty)$. 
Interface vector

- Interface vector before processing: 
  \[ \hat{\xi}_t = [k_t^{r,1} \ldots k_t^{r,R}; \hat{\beta}_t^{r,1} \ldots \hat{\beta}_t^{r,R}; k_t^w; \hat{\beta}_t^w; e_t; v_t; f_t^1 \ldots f_t^R; g_t^a; g_t^w; \hat{\pi}_t^1 \ldots \hat{\pi}_t^R] \]

- Define: \( \text{oneplus}(x) = 1 + \ln (1 + e^x) \in [1, \infty) \).

- Define: \( \text{softmax}(x)_j = \frac{e^{x_j}}{\sum_i e^{x_i}} \).
Interface vector

- Interface vector before processing $\hat{\xi}_t = \begin{bmatrix} k_{t,1} \ldots k_{t,R}; \hat{\beta}_{t,1} \ldots \hat{\beta}_{t,R}; k_t; \hat{\beta}_t; e_t; v_t; f_{t,1} \ldots f_{t,R}; g_t^a; g_t^w; \hat{\pi}_{t,1} \ldots \hat{\pi}_t \end{bmatrix}$

- Define: $\text{oneplus}(x) = 1 + \ln (1 + e^x) \in [1, \infty)$.

- Define: $\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$.

- $\beta_{t,i}^r = \text{oneplus}(\hat{\beta}_{t,i}^r), \beta_t^w = \text{oneplus}(\hat{\beta}_t^w)$
Interface vector

- Interface vector before processing \( \hat{\xi}_t = \begin{bmatrix} k_{t,1}^r \ldots k_{t,R}^r; \hat{\beta}_{t,1}^r \ldots \hat{\beta}_{t,R}^r; k_t^w; \hat{\beta}_t^w; \hat{e}_t; \hat{v}_t; \hat{f}_{t,1}^1 \ldots \hat{f}_{t,R}^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_{t,1}^1 \ldots \hat{\pi}_t^R \end{bmatrix} \)

- Define: \( \text{oneplus}(x) = 1 + \ln (1 + e^x) \in [1, \infty) \).

- Define: \( \text{softmax}(x)_j = \frac{e^{x_j}}{\sum_i e^{x_i}} \).

- \( \beta_{t,i}^r = \text{oneplus}(\hat{\beta}_{t,i}^r), \beta_{t}^w = \text{oneplus}(\hat{\beta}_{t}^w) \)

- \( e_t = \sigma(\hat{e}_t), f_t^i = \sigma(\hat{f}_t^i), g_t^a = \sigma(\hat{g}_t^a), g_t^w = \sigma(\hat{g}_t^w) \)
Interface vector

• Interface vector before processing $\hat{\xi}_t = \left[ k^r_1 \ldots k^R_t; \hat{\beta}^r_1 \ldots \hat{\beta}^R_t; k^w_t; \hat{\beta}^w_t; e_t; \mathbf{v}; f^1_t \ldots f^R_t; \hat{g}^a_t; \hat{g}^w_t; \hat{\pi}^1_t \ldots \hat{\pi}^R_t \right]$

• Define: $\text{oneplus}(x) = 1 + \ln (1 + e^x) \in [1, \infty)$.

• Define: $\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$.

• $\beta^r_{t,i} = \text{oneplus}(\hat{\beta}^r_t)_i$, $\beta^w_t = \text{oneplus}(\hat{\beta}^w_t)$

• $e_t = \sigma(\hat{e}_t)$, $f^i_t = \sigma(\hat{f}^i_t)$, $g^a_t = \sigma(\hat{g}^a_t)$, $g^w_t = \sigma(\hat{g}^w_t)$

• $\pi^i_t = \text{softmax}(\hat{\pi}^i_t)$
• Interface vector before processing $\hat{\xi}_t = \begin{bmatrix} k^r_{t,1} \ldots k^r_{t,R}; \hat{\beta}^r_{t,1} \ldots \hat{\beta}^r_{t,R}; k^w_t; \hat{\beta}^w_t; \hat{e}_t; v_t; f^1_t \ldots f^R_t; \hat{g}^a_t; \hat{g}^w_t; \hat{\pi}^1_t \ldots \hat{\pi}^R_t \end{bmatrix}$

• Define: oneplus($x$) = $1 + \ln (1 + e^x) \in [1, \infty)$.

• Define: softmax($\mathbf{x}$)$_j = \frac{e^{x_j}}{\sum_i e^{x_i}}$.

• $\beta^r_{t,i} = \text{oneplus}(\hat{\beta}^r_{t,i}), \beta^w_t = \text{oneplus}(\hat{\beta}^w_t)$

• $\mathbf{e}_t = \sigma(\hat{\mathbf{e}}_t), f^i_t = \sigma(\hat{f}^i_t), g^a_t = \sigma(\hat{g}^a_t), g^w_t = \sigma(\hat{g}^w_t)$

• $\pi^i_t = \text{softmax}(\hat{\pi}^i_t)$

• Interface vector after processing $\xi_t = \begin{bmatrix} k^r_{t,1} \ldots k^r_{t,R}; \beta^r_{t,1} \ldots \beta^r_{t,R}; k^w_t; \beta^w_t; \mathbf{e}_t; v_t; f^1_t \ldots f^R_t; g^a_t; g^w_t; \pi^1_t \ldots \pi^R_t \end{bmatrix}$
Interacting with memory

Source: Hsin, C., *Implementation and Optimization of Differentiable Neural Computers*
1. Content-based addressing:

- $C(M, k, \beta)[i] = \exp\{D(k, M[i, \cdot])\beta\} \sum_{} \exp\{D(k, M[j, \cdot])\beta\}$

- Cosine similarity $D(u, v) = \frac{u \cdot v}{||u|| \cdot ||v||}$, $u, v \in [-1, 1]$

- $c_{wt} = C(M_{t-1}, k_{wt}, \beta_{wt}) \in \mathbb{S}^N$

2. Dynamic memory allocation:

- Memory retention vector $\psi_t = \prod_{R_i=1}^R (1 - f_{i_t \cdot w_{i_t - 1}}) \in [0, 1]$

- Memory usage vector $u_t = (u_{t-1} + w_{w_t - 1} - u_{t-1} \circ w_{w_t - 1}) \circ \psi_t \in [0, 1]^N$

- 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 $a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]] \in \Delta \mathbb{N}$

3. Write weighting:

- $w_w \in (g_{w_t}[g_{a_t} + (1 - g_{a_t})c_{wt}]) \in \Delta \mathbb{N}$

4. Actual write operation:

- $M_t = M_{t-1} \circ (E - w_w v_T t) + w_w v_T t$
1. Content-based addressing:
   - \( C(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, \cdot])\beta\}}{\sum_j \exp\{D(k, M[j, \cdot])\beta\}} \)
Writing to memory

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

1. Content-based addressing:
   - \( C(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, \cdot])\beta\}}{\sum_j \exp\{D(k, M[j, \cdot])\beta\} } \)
   - cosine similarity \( D(u, v) = \frac{u \cdot v}{|u||v|} \in [-1, 1] \)
   - \( c_t^w = C(M_{t-1}, k_t^w, \beta_t^w) \in S_N \)
Writing to memory

1. Content-based addressing:
   - \( C(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, \cdot])\beta\}}{\sum_j \exp\{D(k, M[j, \cdot])\beta\} } \)
   - cosine similarity \( D(u, v) = \frac{u \cdot v}{|u||v|} \in [-1, 1] \)
   - \( c_t^w = C(M_{t-1}, k_t^w, \beta_t^w) \in S_N \)

2. Dynamic memory allocation:

- memory retention vector \( \psi_t = \prod_{R_i=1} \left( 1 - f_{it}\right) \in [0, 1] \)
- memory usage vector \( u_t = \left( u_t - 1 + w_{w_t-1} - u_{t-1} \cdot w_{w_t-1} \right) \cdot \psi_t \in [0, 1] \)
- 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 \( a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{j=1}^{\phi_t[j]} u_t[\phi_t[i]] \in \Delta \mathbb{N} \)
1. Content-based addressing:
   - $C(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, \cdot])\beta\}}{\sum_j \exp\{D(k, M[j, \cdot])\beta\}}$
   - cosine similarity $D(u, v) = \frac{u \cdot v}{|u||v|} \in [-1, 1]$
   - $c^w_t = C(M_{t-1}, k^w_t, \beta^w_t) \in S_N$

2. Dynamic memory allocation:
   - memory retention vector $\psi_t = \prod_{i=1}^R (1 - f_t^{i} w^{r,i}_{t-1}) \in [0, 1]^N$
Writing to memory

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

2. Dynamic memory allocation:
   - memory retention vector \( \psi_t = \prod_{i=1}^R (1 - f^i_t w^r_{t-1}^i) \in [0, 1]^N \)
   - memory usage vector
     \[ u_t = (u_{t-1} + w^w_{t-1} - u_{t-1} \circ w^w_{t-1}) \circ \psi_t \in [0, 1]^N \]
Writing to memory

1. Content-based addressing:
   - \( C(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, \cdot])\beta\}}{\sum_j \exp\{D(k, M[j, \cdot])\beta\} \} \)
   - cosine similarity \( D(u, v) = \frac{u \cdot v}{|u||v|} \in [-1, 1] \)
   - \( c^w_t = C(M_{t-1}, k^w_t, \beta^w_t) \in S_N \)

2. Dynamic memory allocation:
   - memory retention vector \( \psi_t = \prod_{i=1}^{R} (1 - f_t^{i} w^r_{t-1}^i) \in [0, 1]^N \)
   - memory usage vector
     \( u_t = (u_{t-1} + w^w_{t-1} - u_{t-1} \circ w^w_{t-1}) \circ \psi_t \in [0, 1]^N \)
   - 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:
   - $C(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, \cdot])\beta\}}{\sum_j \exp\{D(k, M[j, \cdot])\beta\}}$
   - cosine similarity $D(u, v) = \frac{u \cdot v}{|u||v|} \in [-1, 1]$
   - $c^w_t = C(M_{t-1}, k^w_t, \beta^w_t) \in \mathcal{S}_N$

2. Dynamic memory allocation:
   - memory retention vector $\psi_t = \prod_{i=1}^{R} \left(1 - f^i_t w^r_{t-1}\right) \in [0, 1]^N$
   - memory usage vector $u_t = (u_{t-1} + w^w_{t-1} - u_{t-1} \circ w^w_{t-1}) \circ \psi_t \in [0, 1]^N$
   - 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 $a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]] \in \Delta_N$
Writing to memory

1. Content-based addressing:
   - \( C(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, \cdot])\beta\}}{\sum_j \exp\{D(k, M[j, \cdot])\beta\} \cdot \text{cosine similarity} \ D(u, v) = \frac{u \cdot v}{\|u\| \|v\|} \in [-1, 1] \)
   - \( c^w_t = C(M_{t-1}, k^w_t, \beta^w_t) \in S_N \)

2. Dynamic memory allocation:
   - memory retention vector \( \psi_t = \prod_{i=1}^R (1 - f^i_t w_{t-1}^r, i) \in [0, 1]^N \)
   - memory usage vector
     \( u_t = (u_{t-1} + w_{t-1}^w - u_{t-1} \circ w_{t-1}^w) \circ \psi_t \in [0, 1]^N \)
   - 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
     \( a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]] \in \Delta_N \)

3. Write weighting:
1. Content-based addressing:
   - \( \mathcal{C}(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, \cdot])\beta\}}{\sum_j \exp\{D(k, M[j, \cdot])\beta\} } \)
   - cosine similarity \( D(u, v) = \frac{u \cdot v}{|u||v|} \in [-1, 1] \)
   - \( c^w_t = \mathcal{C}(M_{t-1}, k^w_t, \beta^w_t) \in S_N \)

2. Dynamic memory allocation:
   - memory retention vector \( \psi_t = \prod_{i=1}^{R} \left( 1 - f^i_t w^i_{t-1} \right) \in [0, 1]^N \)
   - memory usage vector
     \[ u_t = (u_{t-1} + w^w_{t-1} - u_{t-1} \circ w^w_{t-1}) \circ \psi_t \in [0, 1]^N \]
   - 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
     \[ a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]] \in \Delta_N \]

3. Write weighting:
   - \( w^w_t = g^w_t \left[ g^a_t a_t + (1 - g^a_t) c^w_t \right] \in \Delta_N \)
Writing to memory

1. Content-based addressing:
   - \( C(M, k, \beta)[i] = \frac{\exp\{D(k, M[i, :])\beta\}}{\sum_j \exp\{D(k, M[j, :])\beta\} } \)
   - cosine similarity \( D(u, v) = \frac{u \cdot v}{||u|| \cdot ||v||} \in [-1, 1] \)
   - \( c^w_t = C(M_{t-1}, k^w_t, \beta^w_t) \in S_N \)

2. Dynamic memory allocation:
   - memory retention vector \( \psi_t = \prod_{i=1}^{R} (1 - f^r_i w^r_{t-1}) \in [0, 1]^N \)
   - memory usage vector \( u_t = (u_{t-1} + w^w_{t-1} - u_{t-1} \circ w^w_{t-1}) \circ \psi_t \in [0, 1]^N \)
   - 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 \( a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]] \in \Delta_N \)

3. Write weighting:
   - \( w^w_t = g^w_t [g^a_t a_t + (1 - g^a_t)c^w_t] \in \Delta_N \)

4. Actual write operation: \( M_t = M_{t-1} \circ (E - w^w_t e_t^T) + w^w_t v_t^T \)
Interacting with memory

Source: Hsin, C., *Implementation and Optimization of Differentiable Neural Computers*
1. Content-based addressing:

\[ c_{\text{r}, \text{i}t} = C(M_{\text{t}}, \text{w}_{\text{r}, \text{i}t}, \beta_{\text{r}, \text{i}t}) \in \mathbb{S} \]

2. Temporal memory linkage:

- temporal link matrix
  \[ L_{\text{t}} \in [0, 1]^{N \times N}, L_{\text{t}}[i, \cdot] \in \Delta N, L_{\text{t}}[\cdot, j] \in \Delta N \]
- precedence weighting
  \[ p_{\text{t}} = (1 - \sum_{i} \text{w}_{\text{w}, \text{r}, \text{i}t}[i]) p_{\text{t}} - 1 + \text{w}_{\text{w}, \text{r}, \text{i}t}[i] \in \Delta N \]
- linkage logic:
  \[ \forall i: L_{\text{t}}[i, i] = 0, L_{\text{t}}[i, j] = (1 - \text{w}_{\text{w}, \text{r}, \text{i}t}[i] - \text{w}_{\text{w}, \text{r}, \text{i}t}[j]) L_{\text{t}} - 1[i, j] + \text{w}_{\text{w}, \text{r}, \text{i}t}[i] p_{\text{t}} - 1[j] \]
- forward weighting
  \[ f_{\text{i}t} = L_{\text{t}} \text{w}_{\text{r}, \text{i}t} - 1 \in \Delta N \]
- backward weighting
  \[ b_{\text{i}t} = L_{\text{t}} \text{w}_{\text{r}, \text{i}t} - 1 \in \Delta N \]

3. Read weighting:

\[ \text{w}_{\text{r}, \text{i}t} = \pi_{\text{i}t}[1] b_{\text{i}t} + \pi_{\text{i}t}[2] c_{\text{r}, \text{i}t} + \pi_{\text{i}t}[3] f_{\text{i}t} \in \Delta N \]

4. Actual read operation:

\[ r_{\text{i}t} = M_{\text{t}} \text{w}_{\text{r}, \text{i}t}. \]
1. Content-based addressing:
   \[ c^{r,i}_t = C(M_t, k^{r,i}_t, \beta^{r,i}_t) \in S_N \]
1. Content-based addressing:
   - \( c_t^r,i = C(M_t, k_t^r,i, \beta_t^r,i) \in S_N \)

2. Temporal memory linkage:
Reading from memory

1. Content-based addressing:
   - $c^r_i = C(M_t, k^r_i, \beta^r_i) \in S_N$

2. Temporal memory linkage:
   - temporal link matrix $L_t \in [0, 1]^{N \times N}, L_t[i, :] \in \Delta_N, L_t[:, j] \in \Delta_N$
Reading from memory

1. Content-based addressing:
   - \( c_t^r,i = C(M_t, k_t^r,i, \beta_t^r,i) \in S_N \)

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 \( p_t = (1 - \sum_i w_t^w[i]) p_{t-1} + w_t^w \in \Delta_N \)

3. Read weighting:
   - \( w_r,i_t = \pi_{i_t}[1] b_{i_t} + \pi_{i_t}[2] c_{r,i_t} + \pi_{i_t}[3] f_{i_t} \in \Delta_N \)

4. Actual read operation:
   - \( r_{i_t} = M_T w_r, i_t \).
1. Content-based addressing:
   - \( c_t^r, i = C(M_t, k_t^r, i, \beta_t^r, i) \in S_N \)

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 \( p_t = (1 - \sum_i w_t^w[i]) \) \( p_{t-1} + w_t^w \in \Delta_N \)
   - linkage logic: \( \forall i : L_t[i, i] = 0, L_t[i, j] = (1 - w_t^w[i] - w_t^w[j]) L_{t-1}[i, j] + w_t^w[i] p_{t-1}[j] \)
1. Content-based addressing:
   - \( c_t^r,i = C(M_t, k_t^r,i, \beta_t^r,i) \in S_N \)

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 \( p_t = (1 - \sum_i w_t^w[i]) p_{t-1} + w_t^w \in \Delta_N \)
   - linkage logic: \( \forall i : L_t[i, i] = 0, L_t[i, j] = \)
     \( (1 - w_t^w[i] - w_t^w[j]) L_{t-1}[i, j] + w_t^w[i] p_{t-1}[j] \)
   - forward weighting: \( f_t^i = L_t w_{t-1}^r,i \in \Delta_N \)
1. Content-based addressing:
   - \( c_t^r,i = C(M_t, k_t^r,i, \beta_t^r,i) \in S_N \)
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 \( p_t = (1 - \sum_i w^w_t[i]) p_{t-1} + w^w_t \in \Delta_N \)
   - linkage logic: \( \forall i : L_t[i, i] = 0, L_t[i, j] = (1 - w^w_t[i] - w^w_t[j]) L_{t-1}[i, j] + w^w_t[i] p_{t-1}[j] \)
   - forward weighting: \( f^i_t = L_t w^r,t-1 \in \Delta_N \)
   - backward weighting: \( b^i_t = L_t^T w^r,t-1 \in \Delta_N \)
1. Content-based addressing:
   - $c_t^r,i = C(M_t, k_t^r,i, \beta_t^r,i) \in S_N$

2. Temporal memory linkage:
   - temporal link matrix $L_t \in [0, 1]^{N \times N}$, $L_t[i, :] \in \Delta_N$, $L_t[:, j] \in \Delta_N$
   - precedence weighting $p_t = (1 - \sum_i w_t^w[i]) p_{t-1} + w_t^w \in \Delta_N$
   - linkage logic: $\forall i : L_t[i, i] = 0, L_t[i, j] = (1 - w_t^w[i] - w_t^w[j]) L_{t-1}[i, j] + w_t^w[i] p_{t-1}[j]$
   - forward weighting: $f_t^i = L_t w_{t-1}^r,i \in \Delta_N$
   - backward weighting: $b_t^i = L_t^T w_{t-1}^r,i \in \Delta_N$

3. Read weighting:
1. Content-based addressing:
   - $c_t^{r,i} = C(M_t, k_t^{r,i}, \beta_t^{r,i}) \in S_N$

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 $p_t = (1 - \sum_i w_t^w[i]) p_{t-1} + w_t^w \in \Delta_N$
   - linkage logic: $\forall i : L_t[i, i] = 0, L_t[i, j] = (1 - w_t^w[i] - w_t^w[j]) L_{t-1}[i, j] + w_t^w[i] p_{t-1}[j]$
   - forward weighting: $f_t^i = L_t w_{t-1}^{r,i} \in \Delta_N$
   - backward weighting: $b_t^i = L_t^T w_{t-1}^{r,i} \in \Delta_N$

3. Read weighting:
   - $w_t^{r,i} = \pi_t^i[1] b_t^i + \pi_t^i[2] c_t^{r,i} + \pi_t^i[3] f_t^i \in \Delta_N$
Reading from memory

1. Content-based addressing:
   - $c_{t}^{r,i} = C(M_t, k_{t}^{r,i}, \beta_{t}^{r,i}) \in S_N$

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 $p_t = (1 - \sum_i w_t^w[i]) p_{t-1} + w_t^w \in \Delta_N$
   - linkage logic: $\forall i : L_t[i, i] = 0, L_t[i, j] = (1 - w_t^w[i] - w_t^w[j]) L_{t-1}[i, j] + w_t^w[i] p_{t-1}[j]$
   - forward weighting: $f_t^i = L_t w_{t-1}^{r,i} \in \Delta_N$
   - backward weighting: $b_t^i = L_t^T w_{t-1}^{r,i} \in \Delta_N$

3. Read weighting:
   - $w_t^{r,i} = \pi_t^i[1] b_t^i + \pi_t^i[2] c_t^{r,i} + \pi_t^i[3] f_t^i \in \Delta_N$

4. Actual read operation: $r_t^i = M_t^T w_t^{r,i}$. 
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.
<table>
<thead>
<tr>
<th>Traversal</th>
<th>Shortest-path</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Traversal question:</strong></td>
<td><strong>Shortest-path question:</strong></td>
</tr>
<tr>
<td>(BondSt, _, Central),</td>
<td>(Moorgate, PiccadillyCircus, _)</td>
</tr>
<tr>
<td>(_, <em>, Circle), (</em>, _, Circle),</td>
<td></td>
</tr>
<tr>
<td>(_, <em>, Circle), (</em>, _, Circle),</td>
<td></td>
</tr>
<tr>
<td>(_, <em>, Jubilee), (</em>, _, Jubilee),</td>
<td></td>
</tr>
<tr>
<td><strong>Answer:</strong></td>
<td><strong>Answer:</strong></td>
</tr>
<tr>
<td>(BondSt, NottingHillGate, Central)</td>
<td>(Moorgate, Bank, Northern)</td>
</tr>
<tr>
<td>(NottingHillGate, GloucesterRd, Circle)</td>
<td>(Bank, Holborn, Central)</td>
</tr>
<tr>
<td>:</td>
<td>(Holborn, LeicesterSq, Piccadilly)</td>
</tr>
<tr>
<td>(Westminster, GreenPark, Jubilee)</td>
<td>(LeicesterSq, PiccadillyCircus, Piccadilly)</td>
</tr>
<tr>
<td>(GreenPark, BondSt, Jubilee)</td>
<td></td>
</tr>
</tbody>
</table>

Source: [Graves et al., 2016]
Traversing

Graph definition

Source: [Graves et al., 2016]
Traversals are used to navigate through a graph or a network. They can be seen as a sequence of edges that start from a source and end at a target. In the context of urban transportation, they represent the routes that can be taken from one location to another. The diagram illustrates how these traverals can be represented as sequences of edges and how they can be decoded to understand the location content.

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.

