Memory Architectures in Deep (Reinforcement) Learning

Rylan Schaeffer

March 15th, 2019

Deep Learning: Classics and Trends

Roadmap

- Motivation
- History of Memory Architectures in Deep Learning
- Neural Turing Machine (NTM)
- Differentiable Neural Computer (DNC)
- Memory, Reinforcement Learning and Inference Network (MERLIN)

Motivation

- Recurrent neural networks are theoretically Turing-complete [9], but practical problems proliferate

Motivation

- Recurrent neural networks are theoretically Turing-complete [9], but practical problems proliferate
- Vanilla LSTMs struggle on simple tasks requiring memory: copying sequences, adding numbers presented digit by digit, memorizing key-value pairs, etc

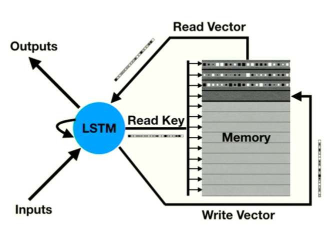
Motivation

- Recurrent neural networks are theoretically Turing-complete [9], but practical problems proliferate
- Vanilla LSTMs struggle on simple tasks requiring memory: copying sequences, adding numbers presented digit by digit, memorizing key-value pairs, etc
- Approach: Use introspective attention mechanism to manipulate, store, retrieve specific information (memory)

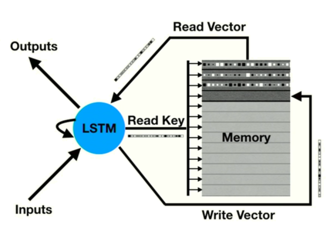
History of Memory Architectures in Deep Learning

- Memory Networks (2014) [14]
- **Neural Turing Machines (2014)** [4]
- Pointer Networks (2015) [11]
- End-to-End Memory Networks (2015) [10]
- **Differentiable Neural Computer (2016)** [5]
- Associative Long Short-Term Memory (2016) [2]
- Lie Access Neural Turing Machine (2016) [17]
- **Memory, RL and Inference Network (2018)** [12]
- Kanerva Machine (2018) [15]
- Relational Memory Core (2018) [8]
- Reconstructive Memory Agent (2018) [6]
- Dynamic Kanerva Machine (2018) [16]
- Improvements to DNC (2019) [1]

Neural Turing Machine

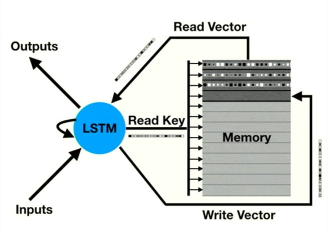


Neural Turing Machine



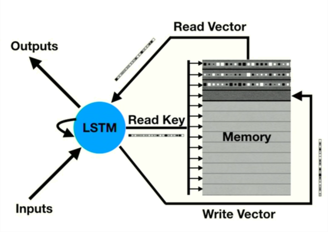
- Couple a neural network to an external 2D matrix

Neural Turing Machine



- Couple a neural network to an external 2D matrix
- Enable network to learn reading/writing by defining interactions in differentiable manner

Neural Turing Machine



- Couple a neural network to an external 2D matrix
- Enable network to learn reading/writing by defining interactions in differentiable manner
- Specifically, read & write are defined as soft attention mechanism over entire matrix

Terminology

- **Weights**: Parameters of network

Terminology

- **Weights**: Parameters of network
- **Weighting**: ”Probability” vector **w** used to determine weighted combinations of memory contents

Terminology

- **Weights**: Parameters of network
- **Weighting**: "Probability" vector $\mathbf{w}$ used to determine weighted combinations of memory contents
- The set of N-dimensional weightings Δ_N is defined as follows:

$$\Delta_N \stackrel{\text{def}}{=} \{\mathbf{w} \in \mathbb{R}^N : w_i \in [0,1], \sum_{i=1}^{N} w_i \leq 1\}$$

Terminology

- **Weights**: Parameters of network
- **Weighting**: ”Probability” vector $\mathbf{w}$ used to determine weighted combinations of memory contents
- The set of N-dimensional weightings Δ_N is defined as follows:

$$\Delta_N \stackrel{\text{def}}{=} \{\mathbf{w} \in \mathbb{R}^N : w_i \in [0,1], \sum_{i=1}^{N} w_i \leq 1\}$$

- Weighting with sum < 1 will be subtly influential for DNC

Neural Turing Machine [4] - Reading

- Notation: Memory matrix has R rows and C columns, denoted M_t

Neural Turing Machine [4] - Reading

- Notation: *Memory matrix* has R rows and C columns, denoted M_t
- To read from memory, network uses **read heads**

Neural Turing Machine [4] - Reading

- Notation: *Memory matrix* has R rows and C columns, denoted M_t
- To read from memory, network uses **read heads**
- Each read head emits a weighting $\mathbf{w_t} \in \Delta_R$

Neural Turing Machine [4] - Reading

- Notation: *Memory matrix has R rows and C columns, denoted M_t*
- To read from memory, network uses **read heads**
- Each read head emits a weighting $\mathsf{w_t} \in \Delta_R$
- Each read head returns to network a weighted combination of memory rows called a **read vector**, $\mathsf{r_t} \in \mathbb{R}^C$

$$\mathsf{r_t} \leftarrow \mathsf{M_t}^T \mathsf{w_t}$$

Neural Turing Machine [4] - Reading

- Notation: Memory matrix has R rows and C columns, denoted M_t
- To read from memory, network uses **read heads**
- Each read head emits a weighting $\mathbf{w_t} \in \Delta_R$
- Each read head returns to network a weighted combination of memory rows called a **read vector**, $\mathbf{r_t} \in \mathbb{R}^C$

$$\mathbf{r_t} \leftarrow \mathbf{M_t}^T \mathbf{w_t}$$

- Network makes parallel reads, one per read head

Neural Turing Machine [4] - Reading

- Notation: Memory matrix has R rows and C columns, denoted M_t
- To read from memory, network uses **read heads**
- Each read head emits a weighting $\mathbf{w_t} \in \Delta_R$
- Each read head returns to network a weighted combination of memory rows called a **read vector**, $\mathbf{r_t} \in \mathbb{R}^C$

$$\mathbf{r_t} \leftarrow \mathbf{M_t}^T \mathbf{w_t}$$

- Network makes parallel reads, one per read head
- DNC differs only in how weighting w_t is generated

Neural Turing Machine [4] - Writing

- To write to memory, network has **write heads**

Neural Turing Machine [4] - Writing

- To write to memory, network has **write heads**
- Each write head emits three vectors:

Neural Turing Machine [4] - Writing

- To write to memory, network has **write heads**
- Each write head emits three vectors:
  - Weighting $\mathbf{w_t} \in \Delta_R$

Neural Turing Machine [4] - Writing

- To write to memory, network has **write heads**
- Each write head emits three vectors:
  - Weighting $\mathbf{w_t} \in \Delta_R$
  - Erase vector $\mathbf{e_t} \in \mathbb{R}^C$ with values $\in (0, 1)$

Neural Turing Machine [4] - Writing

- To write to memory, network has **write heads**
- Each write head emits three vectors:
  - Weighting $\mathbf{w_t} \in \Delta_R$
  - Erase vector $\mathbf{e_t} \in \mathbb{R}^C$ with values $\in (0, 1)$
  - New content vector $\mathbf{v_t} \in \mathbb{R}^C$

Neural Turing Machine [4] - Writing

- To write to memory, network has **write heads**
- Each write head emits three vectors:
    - Weighting $\mathbf{w_t} \in \Delta_R$
    - Erase vector $\mathbf{e_t} \in \mathbb{R}^C$ with values $\in (0, 1)$
    - New content vector $\mathbf{v_t} \in \mathbb{R}^C$
- Each write head modifies every row in memory by (partially) erasing old values and adding new values

$$\mathbf{M_t} \leftarrow \mathbf{M_{t+1}} \circ (\mathbf{1} - \mathbf{w_t}\mathbf{e_t}^T) + \mathbf{w_t}\mathbf{v_t}^T$$

Neural Turing Machine [4] - Writing

- To write to memory, network has **write heads**
- Each write head emits three vectors:
  - Weighting $\mathbf{w_t} \in \Delta_R$
  - Erase vector $\mathbf{e_t} \in \mathbb{R}^C$ with values $\in (0, 1)$
  - New content vector $\mathbf{v_t} \in \mathbb{R}^C$
- Each write head modifies every row in memory by (partially) erasing old values and adding new values

$$\mathbf{M_t} \leftarrow \mathbf{M_{t+1}} \circ (\mathbf{1} - \mathbf{w_t}\mathbf{e_t}^T) + \mathbf{w_t}\mathbf{v_t}^T$$

- Here, $\circ$ denotes element-wise multiplication

Neural Turing Machine [4] - Writing

- To write to memory, network has **write heads**
- Each write head emits three vectors:
  - Weighting $\mathbf{w_t} \in \Delta_R$
  - Erase vector $\mathbf{e_t} \in \mathbb{R}^C$ with values $\in (0, 1)$
  - New content vector $\mathbf{v_t} \in \mathbb{R}^C$
- Each write head modifies every row in memory by (partially) erasing old values and adding new values

$$\mathbf{M_t} \leftarrow \mathbf{M_{t+1}} \circ (\mathbf{1} - \mathbf{w_t}\mathbf{e_t}^T) + \mathbf{w_t}\mathbf{v_t}^T$$

- Here, $\circ$ denotes element-wise multiplication
- Writes are performed in two sequential steps (erase, then add) to permit parallel writes

Neural Turing Machine [4] - Writing

- To write to memory, network has **write heads**
- Each write head emits three vectors:
  - Weighting $\mathbf{w_t} \in \Delta_R$
  - Erase vector $\mathbf{e_t} \in \mathbb{R}^C$ with values $\in (0, 1)$
  - New content vector $\mathbf{v_t} \in \mathbb{R}^C$
- Each write head modifies every row in memory by (partially) erasing old values and adding new values

$$\mathbf{M_t} \leftarrow \mathbf{M_{t+1}} \circ (\mathbf{1} - \mathbf{w_t}\mathbf{e_t}^T) + \mathbf{w_t}\mathbf{v_t}^T$$

- Here, $\circ$ denotes element-wise multiplication
- Writes are performed in two sequential steps (erase, then add) to permit parallel writes
- DNC differs only in how weighting w_t is generated

Neural Turing Machine [4] - Attention Weightings

- Four steps to generate each read/write head's weightings w_t:

Neural Turing Machine [4] - Attention Weightings

- Four steps to generate each read/write head's weightings $\mathbf{w_t}$:
- Content: Network emits search key $\mathbf{k_t} \in \mathbb{R}^C$ and search key strength $\beta \in [1, \infty)$

$$w_t^c[i] \leftarrow Softmax(\beta Similarity(\mathbf{k_t}, \mathbf{M_t}[\mathbf{i}]))$$

Neural Turing Machine [4] - Attention Weightings

- Four steps to generate each read/write head's weightings $\mathbf{w_t}$:
- Content: Network emits search key $\mathbf{k_t} \in \mathbb{R}^C$ and search key strength $\beta \in [1, \infty)$

$$w_t^c[i] \leftarrow Softmax(\beta Similarity(\mathbf{k_t}, \mathbf{M_t}[\mathbf{i}]))$$

- Interpolation: Network emits scalar g_t to blend content-based weighting with previous weighting

$$\mathbf{w_t^g} \leftarrow g_t \mathbf{w_t}^c + (1 - g_t)\mathbf{w_{t-1}}$$

Neural Turing Machine [4] - Attention Weightings

- Four steps to generate each read/write head's weightings $\mathbf{w_t}$:
- Content: Network emits search key $\mathbf{k_t} \in \mathbb{R}^C$ and search key strength $\beta \in [1, \infty)$

$$w_t^c[i] \leftarrow Softmax(\beta Similarity(\mathbf{k_t}, \mathbf{M_t}[\mathbf{i}]))$$

- Interpolation: Network emits scalar g_t to blend content-based weighting with previous weighting

$$\mathbf{w_t^g} \leftarrow g_t \mathbf{w_t}^c + (1 - g_t)\mathbf{w_{t-1}}$$

- Location: Network emits distribution over permitted shift values (e.g. -1, 0, 1) s_t to rotationally shift weighting (mod num of rows)

$$w_t^l[i] \leftarrow \sum_{j=0}^{R-1} w_t^g[j] s_t[i-j]$$

Neural Turing Machine [4] - Attention Weightings

- Four steps to generate each read/write head's weightings $\mathbf{w_t}$:
- Content: Network emits search key $\mathbf{k_t} \in \mathbb{R}^C$ and search key strength $\beta \in [1, \infty)$

$$w_t^c[i] \leftarrow Softmax(\beta Similarity(\mathbf{k_t}, \mathbf{M_t[i]}))$$

- Interpolation: Network emits scalar g_t to blend content-based weighting with previous weighting

$$\mathbf{w_t^g} \leftarrow g_t \mathbf{w_t}^c + (1 - g_t)\mathbf{w_{t-1}}$$

- Location: Network emits distribution over permitted shift values (e.g. -1, 0, 1) s_t to rotationally shift weighting (mod num of rows)

$$w_t^l[i] \leftarrow \sum_{j=0}^{R-1} w_t^g[j] s_t[i-j]$$

- Sharpen: Network emits scalar $\gamma_t \geq 1$ to sharpen weighting

$$w_t[i] = Softmax(w_t^l[i]^{\gamma_t})$$

Differentiable Neural Computer [5] - Motivation

- Problem: NTM has no mechanism to read sequential writes if a write head jumps

Differentiable Neural Computer [5] - Motivation

- Problem: NTM has no mechanism to read sequential writes if a write head jumps
- Approach: Use a **temporal link matrix** that represents degree to which row i was written to after row j

Differentiable Neural Computer [5] - Motivation

- Problem: NTM has no mechanism to read sequential writes if a write head jumps
- Approach: Use a **temporal link matrix** that represents degree to which row *i* was written to after row *j*
- Problem: NTM has no mechanism to prevent memory blocks from overlapping/interfering

Differentiable Neural Computer [5] - Motivation

- Problem: NTM has no mechanism to read sequential writes if a write head jumps
- Approach: Use a **temporal link matrix** that represents degree to which row i was written to after row j
- Problem: NTM has no mechanism to prevent memory blocks from overlapping/interfering
- Problem: NTM has no mechanism to indicate memory blocks are no longer needed

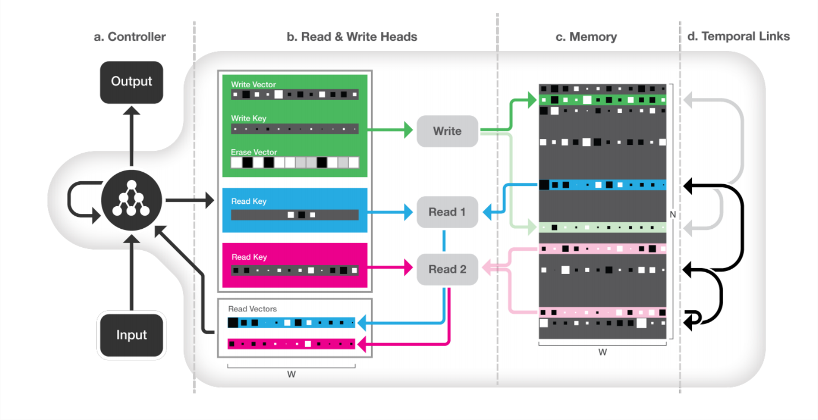
Differentiable Neural Computer [5] - Motivation

- Problem: NTM has no mechanism to read sequential writes if a write head jumps
- Approach: Use a **temporal link matrix** that represents degree to which row *i* was written to after row *j*
- Problem: NTM has no mechanism to prevent memory blocks from overlapping/interfering
- Problem: NTM has no mechanism to indicate memory blocks are no longer needed
- Approach: Enable network to learn **dynamic memory management**

Differentiable Neural Computer [5] - Motivation

- Problem: NTM has no mechanism to read sequential writes if a write head jumps
- Approach: Use a **temporal link matrix** that represents degree to which row i was written to after row j
- Problem: NTM has no mechanism to prevent memory blocks from overlapping/interfering
- Problem: NTM has no mechanism to indicate memory blocks are no longer needed
- Approach: Enable network to learn **dynamic memory management**
- Proposal: Use different attention mechanisms for reading and for writing

Differentiable Neural Computer [5] - Picture



Differentiable Neural Computer [5] - Reading

- Goal: Generate weighting based on content and location-based previous reads

Differentiable Neural Computer [5] - Reading

- Goal: Generate weighting based on content and location-based previous reads
- Network emits search key $\mathsf{k_t} \in \mathbb{R}^C$ and key strength $\beta \in [1, \infty)$ and computes **content (read) weighting $\mathsf{c_t}$**

$$c_t[i] \leftarrow \mathit{Softmax}(\beta \mathit{Similarity}(\mathsf{k_t}, \mathsf{M_t}[\mathsf{i}]))$$

Differentiable Neural Computer [5] - Reading

- Goal: Generate weighting based on content and location-based previous reads
- Network emits search key $\mathsf{k_t} \in \mathbb{R}^C$ and key strength $\beta \in [1, \infty)$ and computes **content (read) weighting $\mathsf{c_t}$**

$$c_t[i] \leftarrow \mathit{Softmax}(\beta \mathit{Similarity}(\mathsf{k_t}, \mathsf{M_t}[\mathsf{i}]))$$

- Assume a temporal link matrix $\mathsf{L_t} \in [0, 1]^{R \times R}$ exists that represents degree to which row i was written to after row j

Differentiable Neural Computer [5] - Reading

- Goal: Generate weighting based on content and location-based previous reads
- Network emits search key $\mathbf{k_t} \in \mathbb{R}^C$ and key strength $\beta \in [1, \infty)$ and computes **content (read) weighting** $\mathbf{c_t}$

$$c_t[i] \leftarrow Softmax(\beta Similarity(\mathbf{k_t}, \mathbf{M_t}[\mathbf{i}]))$$

- Assume a temporal link matrix $\mathbf{L_t} \in [0, 1]^{R \times R}$ exists that represents degree to which row i was written to after row j
- Define **forward weighting** and **backward weighting**: $\mathbf{f_t}, \mathbf{b_t} \in \Delta_R$:

$$\mathbf{f_t} \leftarrow \mathbf{L_t} \mathbf{w}^{\text{read}}_{\mathbf{t-1}}$$
$$\mathbf{b_t} \leftarrow \mathbf{L_t}^T \mathbf{w}^{\text{read}}_{\mathbf{t-1}}$$

Differentiable Neural Computer [5] - Reading

- Goal: Generate weighting based on content and location-based previous reads
- Network emits search key $\mathbf{k_t} \in \mathbb{R}^C$ and key strength $\beta \in [1, \infty)$ and computes **content (read) weighting $\mathbf{c_t}$**

$$c_t[i] \leftarrow Softmax(\beta Similarity(\mathbf{k_t}, \mathbf{M_t}[\mathbf{i}]))$$

- Assume a temporal link matrix $\mathbf{L_t} \in [0, 1]^{R \times R}$ exists that represents degree to which row i was written to after row j
- Define **forward weighting** and **backward weighting**: $\mathbf{f_t}, \mathbf{b_t} \in \Delta_R$:

$$\mathbf{f_t} \leftarrow \mathbf{L_t} \mathbf{w_{t-1}^{read}}$$

$$\mathbf{b_t} \leftarrow \mathbf{L_t}^T \mathbf{w_{t-1}^{read}}$$

- Network emits a **read mode weighting $\mathbf{m_t} \in \Delta_3$** to adjudicate between the backward, content and forward weightings

$$\mathbf{w_t^{read}} \leftarrow m_t[1]\mathbf{b_t} + m_t[2]\mathbf{c_t} + m_t[3]\mathbf{f_t}$$

Differentiable Neural Computer [5] - Temporal Link Matrix

- Goal: Track the degree that a row i was written to after row j using a **temporal link matrix** $\mathbf{L_t} \in [0, 1]^{R \times R}$

Differentiable Neural Computer [5] - Temporal Link Matrix

- Goal: Track the degree that a row i was written to after row j using a **temporal link matrix** $\mathbf{L_t} \in [0, 1]^{R \times R}$
- Define a precedence weighting $\mathbf{p_t} \in \Delta_R$, where $p_t[i]$ represents degree to which row i was last row written to

$$\mathbf{p_0} \leftarrow \mathbf{0}$$

$$\mathbf{p_t} \leftarrow (1 - \sum_{r=1}^{R} w_t^{\text{write}}[r])\mathbf{p_{t+1}} + \mathbf{w_t^{\text{write}}}$$

Differentiable Neural Computer [5] - Temporal Link Matrix

- Goal: Track the degree that a row i was written to after row j using a **temporal link matrix** $\mathbf{L_t} \in [0,1]^{R \times R}$
- Define a precedence weighting $\mathbf{p_t} \in \Delta_R$, where $p_t[i]$ represents degree to which row i was last row written to

$$\mathbf{p_0} \leftarrow \mathbf{0}$$

$$\mathbf{p_t} \leftarrow (1 - \sum_{r=1}^{R} w_t^{\text{write}}[r])\mathbf{p_{t+1}} + \mathbf{w_t^{\text{write}}}$$

- Then, use the precedent weighting to construct the temporal link matrix

$$\mathbf{L_0} \leftarrow \mathbf{0}$$

$$L_t[i,i] \leftarrow 0$$

$$L_t[i,j] \leftarrow (1 - w_t^{\text{write}}[i] - w_t^{\text{write}}[j])L_{t-1}[i,j] + w_t^{\text{write}}[i]p_{t-1}[j]$$

Differentiable Neural Computer [5] - Writing

- Goal: Write using content lookup, constrained by memory management system

Differentiable Neural Computer [5] - Writing

- Goal: Write using content lookup, constrained by memory management system
- Network emits search key $\mathbf{k_t} \in \mathbb{R}^C$ and key strength $\beta \in [1, \infty)$ and computes **content (write) weighting**

$$c_t[i] \leftarrow Softmax(\beta Similarity(\mathbf{k_t}, \mathbf{M_t}[\mathbf{i}]))$$

Differentiable Neural Computer [5] - Writing

- Goal: Write using content lookup, constrained by memory management system
- Network emits search key $\mathsf{k_t} \in \mathbb{R}^C$ and key strength $\beta \in [1, \infty)$ and computes **content (write) weighting**

$$c_t[i] \leftarrow Softmax(\beta Similarity(\mathsf{k_t}, \mathsf{M_t}[\mathsf{i}]))$$

- Assume learned dynamic memory management **allocation weighting** $\mathsf{a_t} \in \Delta_R$ exists that represents degree to which each row can be written to

Differentiable Neural Computer [5] - Writing

- Goal: Write using content lookup, constrained by memory management system
- Network emits search key $\mathbf{k_t} \in \mathbb{R}^C$ and key strength $\beta \in [1, \infty)$ and computes **content (write) weighting**

$$c_t[i] \leftarrow Softmax(\beta Similarity(\mathbf{k_t}, \mathbf{M_t[i]}))$$

- Assume learned dynamic memory management **allocation weighting** $\mathbf{a_t} \in \Delta_R$ exists that represents degree to which each row can be written to
- Network emits two gates, write and allocation $g_t^w, g_t^a \in [0, 1]$, to interpolate content weighting with allocation weighting

$$\mathbf{w_t^{\text{write}}} \leftarrow g_t^w(g_t^a \mathbf{a_t} + (1 - g_t^a)\mathbf{c_t^w})$$

Differentiable Neural Computer [5] - Writing

- Goal: Write using content lookup, constrained by memory management system
- Network emits search key $\mathbf{k_t} \in \mathbb{R}^C$ and key strength $\beta \in [1, \infty)$ and computes **content (write) weighting**

$$c_t[i] \leftarrow Softmax(\beta Similarity(\mathbf{k_t}, \mathbf{M_t}[\mathbf{i}]))$$

- Assume learned dynamic memory management **allocation weighting $\mathbf{a_t} \in \Delta_R$** exists that represents degree to which each row can be written to
- Network emits two gates, write and allocation $g_t^w, g_t^a \in [0, 1]$, to interpolate content weighting with allocation weighting

$$\mathbf{w_t^{write}} \leftarrow g_t^w(g_t^a \mathbf{a_t} + (1 - g_t^a)\mathbf{c_t^w})$$

- Like NTM, network also emits erase vector $e_t \in (0, 1)^C$ and new content vector $v_t \in \mathbb{R}^C$, and updates the memory:

$$\mathbf{M_t} \leftarrow \mathbf{M_{t+1}} \circ (1 - \mathbf{w_t^{write}} \mathbf{e_t}^T) + \mathbf{w_t^{write}} \mathbf{v_t}^T$$

Differentiable Neural Computer [5] - Memory Management

- Goal: Specify which memory rows can be written to

Differentiable Neural Computer [5] - Memory Management

- Goal: Specify which memory rows can be written to
- For each read head, network indicates whether previously read contents are still needed using a free gates $f_t^h \in [0, 1]$

Differentiable Neural Computer [5] - Memory Management

- Goal: Specify which memory rows can be written to
- For each read head, network indicates whether previously read contents are still needed using a free gates $f_t^h \in [0, 1]$
- Indicate which rows are still needed by creating usage weighting $u_t \in [0, 1]^R$

$$\psi_t \leftarrow \prod_{h=1}^{\text{read heads}} (1 - f_t^h \mathbf{w}_{t-1}^h)$$

$$\mathbf{u}_t \leftarrow (\mathbf{u}_{t-1} + (1 - \mathbf{u}_{t-1}) \circ \mathbf{w}_{t-1}^{\text{write}}) \circ \psi_t$$

Differentiable Neural Computer [5] - Memory Management

- Goal: Specify which memory rows can be written to
- For each read head, network indicates whether previously read contents are still needed using a free gates $f_t^h \in [0, 1]$
- Indicate which rows are still needed by creating usage weighting $u_t \in [0, 1]^R$

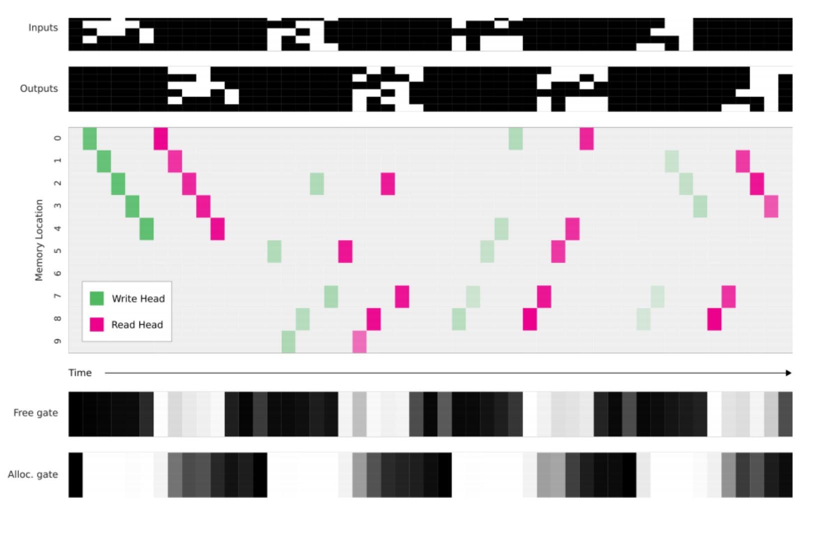
$$\psi_t \leftarrow \prod_{h=1}^{\text{read heads}} (1 - f_t^h \mathbf{w}_{t-1}^{h})$$

$$\mathbf{u}_t \leftarrow (\mathbf{u}_{t-1} + (1 - \mathbf{u}_{t-1}) \circ \mathbf{w}_{t-1}^{\text{write}}) \circ \psi_t$$

- Create the **allocation weighting** $\mathbf{a}_t \in \Delta_R$ by sorting the usages. Let $\phi_t[i]$ be the index of the i-th least used location,

$$a_t[\phi_t[j]] \leftarrow (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]]$$

DNC [5] - Dynamic Memory Management Experiment



DNC [5] - Testing Graph Experiments

- How well does DNC perform at reasoning in graph structures, compared against NTM and LSTM?

DNC [5] - Testing Graph Experiments

- How well does DNC perform at reasoning in graph structures, compared against NTM and LSTM?
- 2 graph datasets

DNC [5] - Testing Graph Experiments

- How well does DNC perform at reasoning in graph structures, compared against NTM and LSTM?
- 2 graph datasets
    - bAbI [13]: programmatically generated natural language questions for textual reasoning

DNC [5] - Testing Graph Experiments

- How well does DNC perform at reasoning in graph structures, compared against NTM and LSTM?
- 2 graph datasets
  - bAbI [13]: programmatically generated natural language questions for textual reasoning
  - Randomly generated planar graphs consisting of 3-tuples: (source, destination, edge label)

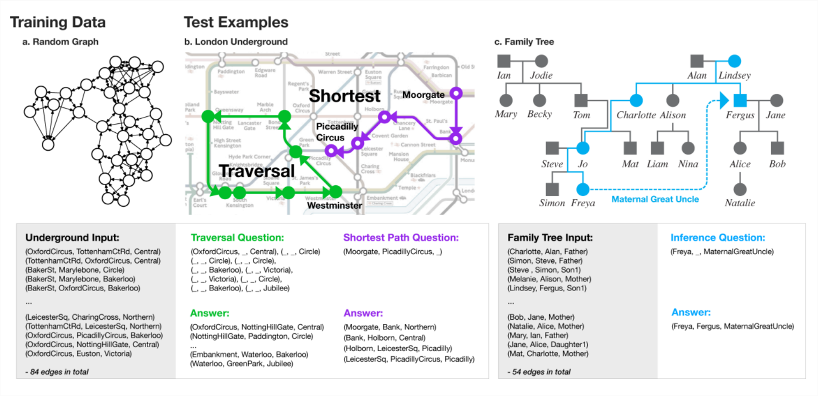
DNC [5] - Testing Graph Experiments

- How well does DNC perform at reasoning in graph structures, compared against NTM and LSTM?
- 2 graph datasets
  - bAbI [13]: programmatically generated natural language questions for textual reasoning
  - Randomly generated planar graphs consisting of 3-tuples: (source, destination, edge label)
- 3 types of queries: path traversal, shortest path, inferred relations

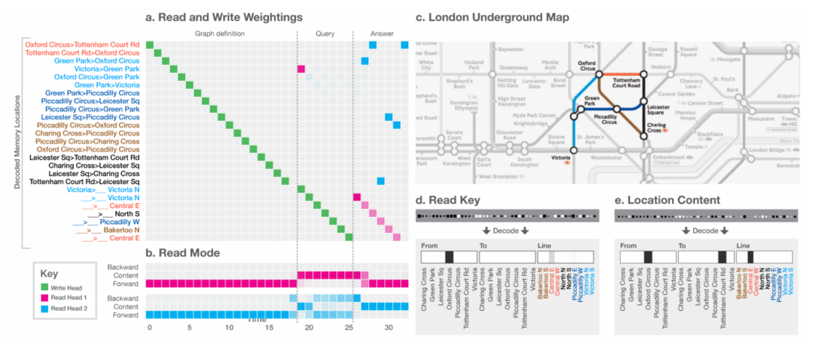
DNC [5] - bAbI Results

Task	LSTM	NTM	DNC1	DNC2
1: 1 supporting fact	28.4±1.5	40.6±6.7	**9.0 ± 12.6**	16.2±13.7
2: 2 supporting facts	56.0±1.5	56.3±1.5	**39.2 ± 20.5**	47.5±17.3
3: 3 supporting facts	51.3±1.4	47.8±1.7	**39.6 ± 16.4**	44.3±14.5
4: 2 argument relations	0.8±0.5	0.9±0.7	**0.4 ± 0.7**	**0.4 ± 0.3**
5: 3 argument relations	3.2±0.5	1.9±0.8	**1.5 ± 1.0**	1.9±0.6
6: yes/no questions	15.2±1.5	18.4±1.6	**6.9 ± 7.5**	11.1±7.1
7: counting	16.4±1.4	19.9±2.5	**9.8 ± 7.0**	15.4±7.1
8: lists/sets	17.7±1.2	18.5±4.9	**5.5 ± 5.9**	10.0±6.6
9: simple negation	15.4±1.5	17.9±2.0	**7.7 ± 8.3**	11.7±7.4
10: indefinite knowledge	28.7±1.7	25.7±7.3	**9.6 ± 11.4**	14.7±10.8
11: basic coreference	12.2±3.5	24.4±7.0	**3.3 ± 5.7**	7.2±8.1
12: conjunction	5.4±0.6	21.9±6.6	**5.0 ± 6.3**	10.1±8.1
13: compound coreference	7.2±2.3	8.2±0.8	**3.1 ± 3.6**	5.5±3.4
14: time reasoning	55.9±1.2	44.9±13.0	**11.0 ± 7.5**	15.0±7.4
15: basic deduction	47.0±1.7	46.5±1.6	**27.2 ± 20.1**	40.2±11.1
16: basic induction	**53.3 ± 1.3**	53.8±1.4	53.6±1.9	54.7±1.3
17: positional reasoning	34.8±4.1	**29.9 ± 5.2**	32.4±8.0	30.9±10.1
18: size reasoning	5.0±1.4	4.5±1.3	**4.2 ± 1.8**	4.3±2.1
19: path finding	90.9±1.1	86.5±19.4	**64.6 ± 37.4**	75.8±30.4
20: agents motivations	1.3±0.4	1.4±0.6	**0.0 ± 0.1**	**0.0 ± 0.0**
Mean Error (%)	27.3±0.8	28.5±2.9	**16.7 ± 7.6**	20.8±7.1
Failed Tasks (err. > 5%)	17.1±1.0	17.3±0.7	**11.2 ± 5.4**	14.0±5.0

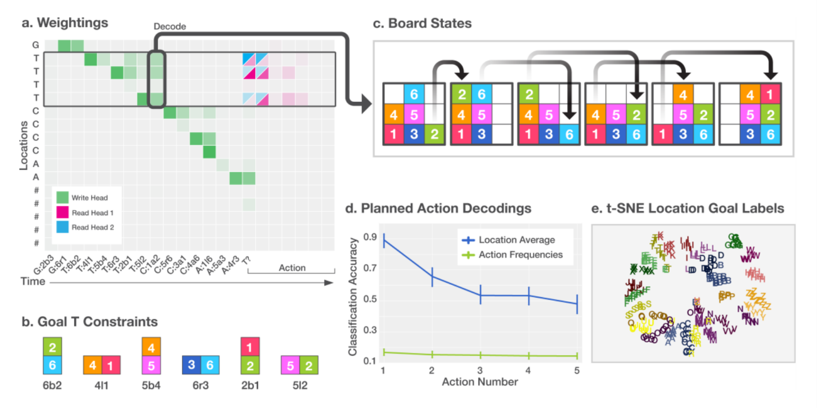
DNC [5] - Custom Graphs Experiment



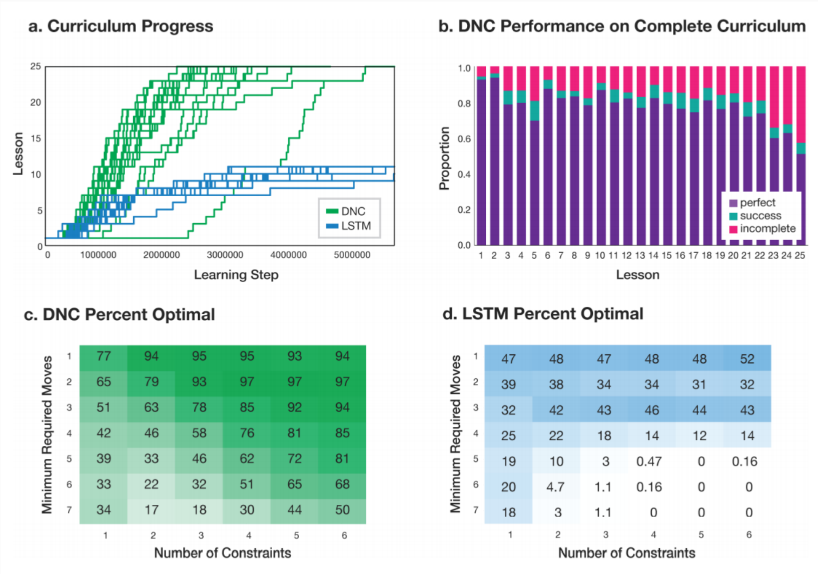
DNC [5] - Custom Graphs Experiment



DNC [5] - Reinforcement Learning Experiment



DNC [5] - Reinforcement Learning Experiment



Memory, RL and Inference Network [12] - Motivation

- Problem: On simple RL task, DNC required curriculum training to learn how to use its memory

Memory, RL and Inference Network [12] - Motivation

- Problem: On simple RL task, DNC required curriculum training to learn how to use its memory
- Greg Wayne [7] spurns end-to-end learning, instead arguing:

Memory, RL and Inference Network [12] - Motivation

- Problem: On simple RL task, DNC required curriculum training to learn how to use its memory
- Greg Wayne [7] spurns end-to-end learning, instead arguing:
    - Cost functions are diverse across areas and change over development

Memory, RL and Inference Network [12] - Motivation

- Problem: On simple RL task, DNC required curriculum training to learn how to use its memory
- Greg Wayne [7] spurns end-to-end learning, instead arguing:
    - Cost functions are diverse across areas and change over development
    - Specialized systems allow efficient solution of key computational sub-problems

Memory, RL and Inference Network [12] - Motivation

- Problem: On simple RL task, DNC required curriculum training to learn how to use its memory
- Greg Wayne [7] spurns end-to-end learning, instead arguing:
  - Cost functions are diverse across areas and change over development
  - Specialized systems allow efficient solution of key computational sub-problems
- Approach: Use separate objective functions and networks for memory, action selection.

Memory, RL and Inference Network [12] - Motivation

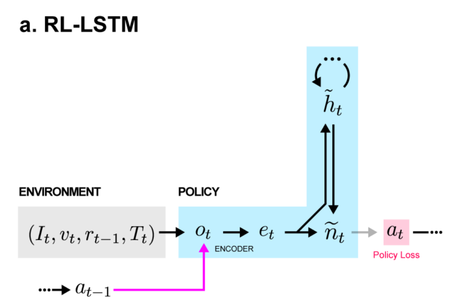
- Problem: On simple RL task, DNC required curriculum training to learn how to use its memory
- Greg Wayne [7] spurns end-to-end learning, instead arguing:
  - Cost functions are diverse across areas and change over development
  - Specialized systems allow efficient solution of key computational sub-problems
- Approach: Use separate objective functions and networks for memory, action selection.
- Train memory to learn predictive (generative) model of world in unsupervised manner, and train actions via reinforcement learning, granting agent access to memory

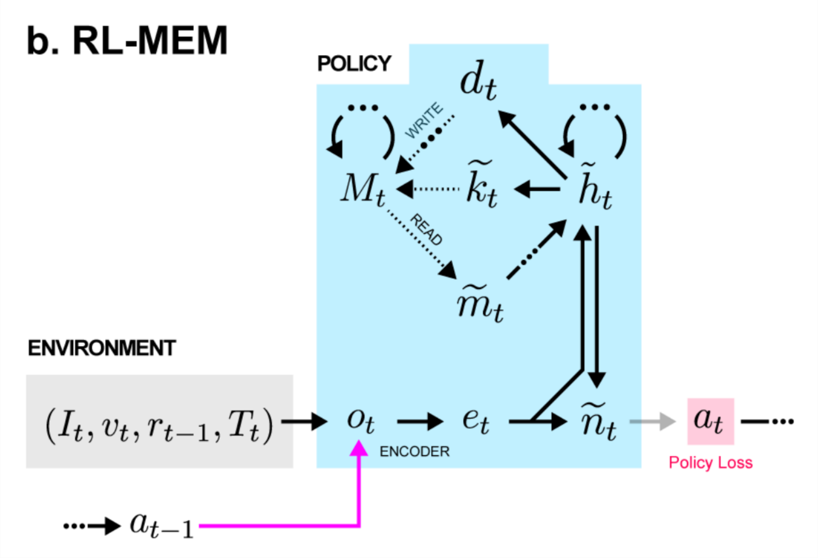
Memory, RL and Inference Network [12] - Motivation

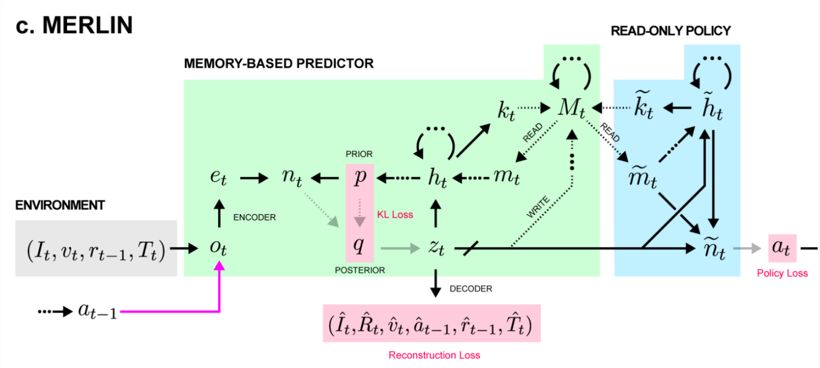
- Problem: On simple RL task, DNC required curriculum training to learn how to use its memory
- Greg Wayne [7] spurns end-to-end learning, instead arguing:
  - Cost functions are diverse across areas and change over development
  - Specialized systems allow efficient solution of key computational sub-problems
- Approach: Use separate objective functions and networks for memory, action selection.
- Train memory to learn predictive (generative) model of world in unsupervised manner, and train actions via reinforcement learning, granting agent access to memory
- Compared three agent architectures (LSTM, MEM, MERLIN) across a variety of tasks requiring memory

MERLIN [12] - LSTM

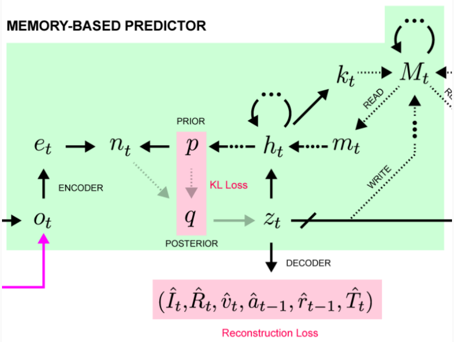
- I_t: Image
- v_t: Egocentric translational and rotational velocity
- r_{t-1}: Previous reward
- a_{t-1}: Previous action
- T: (Optional) Text instruction
- $\tilde{h}_t$: LSTM
- $\tilde{n}_t$: Action probabilities




b. RL-MEM
POLICY
d_t
WRITE
M_t
$\tilde{k}_t$
$\tilde{h}_t$
READ
$\tilde{m}_t$
ENVIRONMENT
(I_t, v_t, r_{t-1}, T_t)
o_t
e_t
$\tilde{n}_t$
a_t
ENCODER
Policy Loss
a_{t-1}

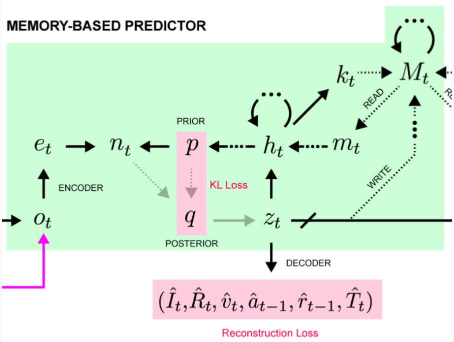

c. MERLIN
MEMORY-BASED PREDICTOR
READ-ONLY POLICY
ENVIRONMENT
(I_t, v_t, r_{t-1}, T_t)
a_{t-1}
o_t
ENCODER
e_t
n_t
PRIOR
p
KL Loss
q
POSTERIOR
h_t
m_t
z_t
k_t
M_t
READ
WRITE
DECODER
$(\hat{I}_t, \hat{R}_t, \hat{v}_t, \hat{a}_{t-1}, \hat{r}_{t-1}, \hat{T}_t)$
Reconstruction Loss
$\tilde{k}_t$
$\tilde{h}_t$
$\tilde{m}_t$
$\tilde{n}_t$
a_t
Policy Loss

MERLIN [12] - Memory-Based Predictor



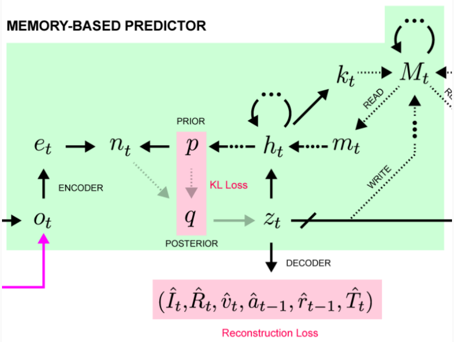
MERLIN [12] - Memory-Based Predictor

- LSTM h_{t-1} outputs a prior p_{t-1} for the next state variable z_t

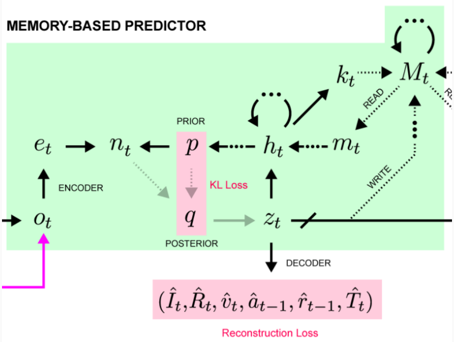


MERLIN [12] - Memory-Based Predictor

- LSTM h_{t-1} outputs a prior p_{t-1} for the next state variable z_t
- p_{t-1} concatenated with e_t, fed through network to produce n_t

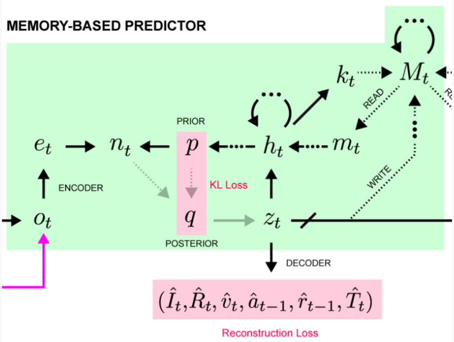


- LSTM h_{t-1} outputs a prior p_{t-1} for the next state variable z_t
- p_{t-1} concatenated with e_t, fed through network to produce n_t
- State posterior $q_t \leftarrow p_{t-1} + n_t$

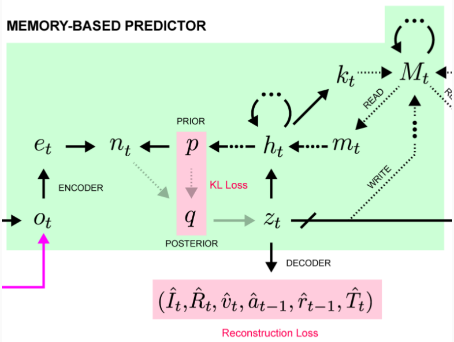


MERLIN [12] - Memory-Based Predictor

- LSTM h_{t-1} outputs a prior p_{t-1} for the next state variable z_t
- p_{t-1} concatenated with e_t, fed through network to produce n_t
- State posterior $q_t \leftarrow p_{t-1} + n_t$
- z_t sampled from q_t, decoded to reconstruct observations and then appended as new row in M_t

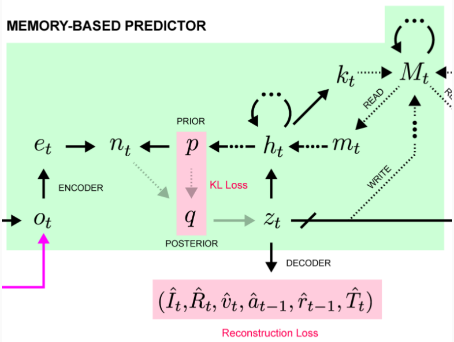


MERLIN [12] - Memory-Based Predictor

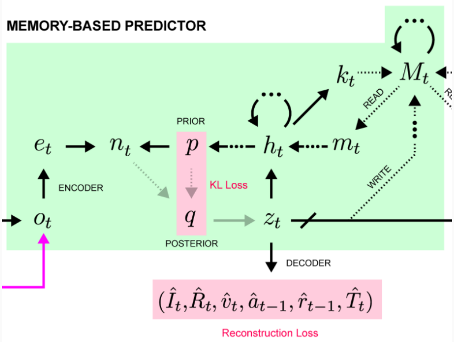


MERLIN [12] - Memory-Based Predictor

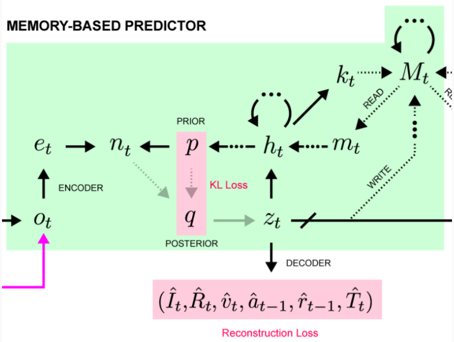
- Problem: reconstructing inputs alone can result in loss of small, but critical information (“bullet problem”)



- Problem: reconstructing inputs alone can result in loss of small, but critical information ("bullet problem")
- Approach: Also reconstruct the return prediction $\hat{R}_t$ [3]

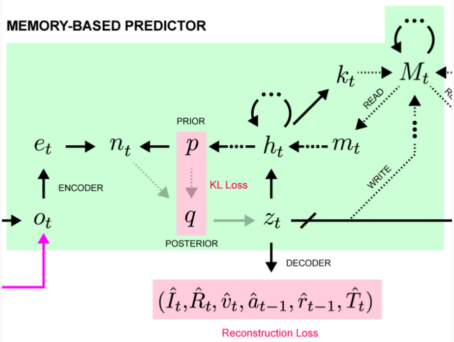


MERLIN [12] - Memory-Based Predictor



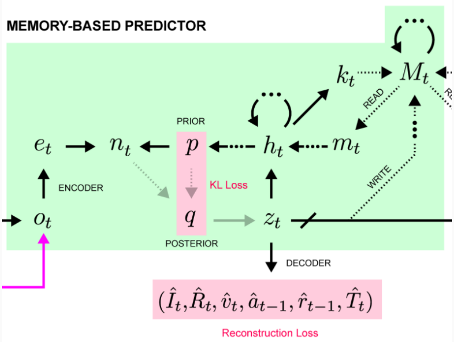
MERLIN [12] - Memory-Based Predictor

- Problem: Sampled state variables z_t have no knowledge of subsequent events

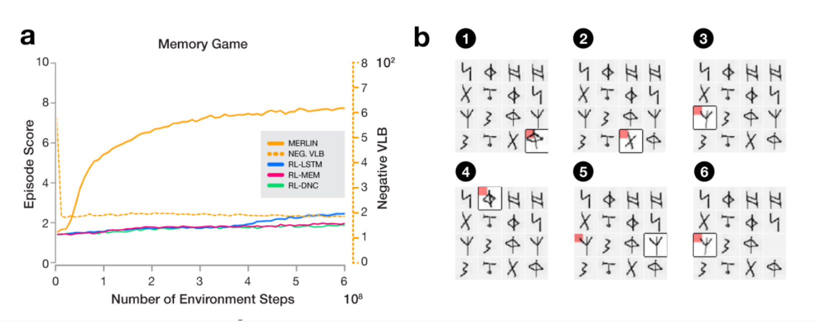


MERLIN [12] - Memory-Based Predictor

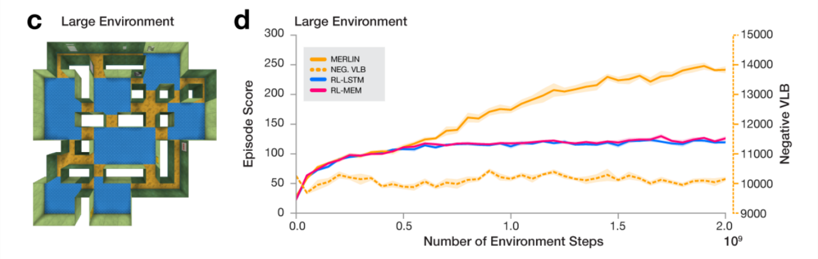
- Problem: Sampled state variables z_t have no knowledge of subsequent events
- Approach: Concatenate z_t with filtered sum of subsequent state variables $(1-\gamma)\sum_{t'>t}\gamma^{t'-t}z_{t'}$ in memory

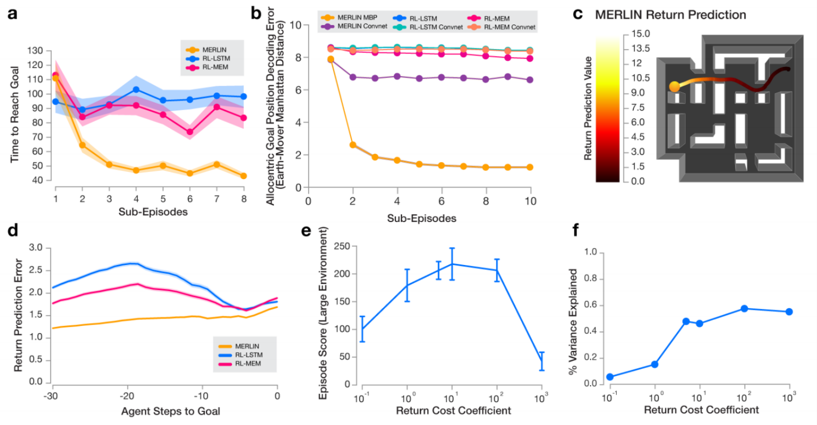


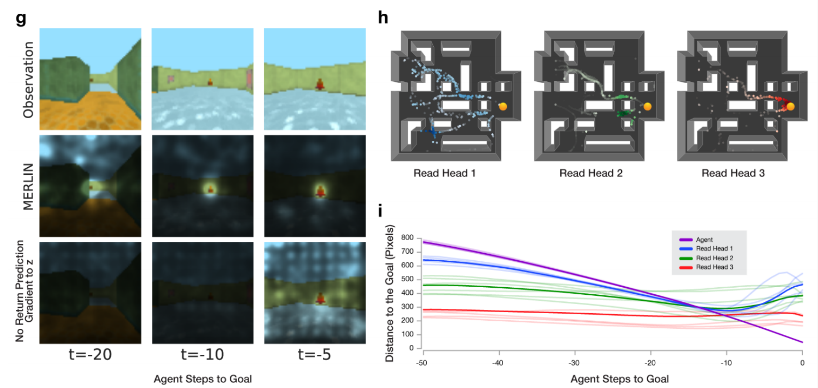
Pairs of 8 Omniglot images are obscured. Agent looks at one image at a time, trying to find pairs.



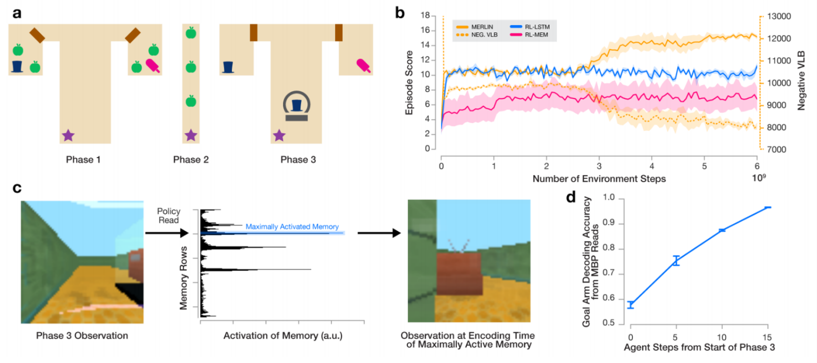
MERLIN [12] - One-Shot Navigation

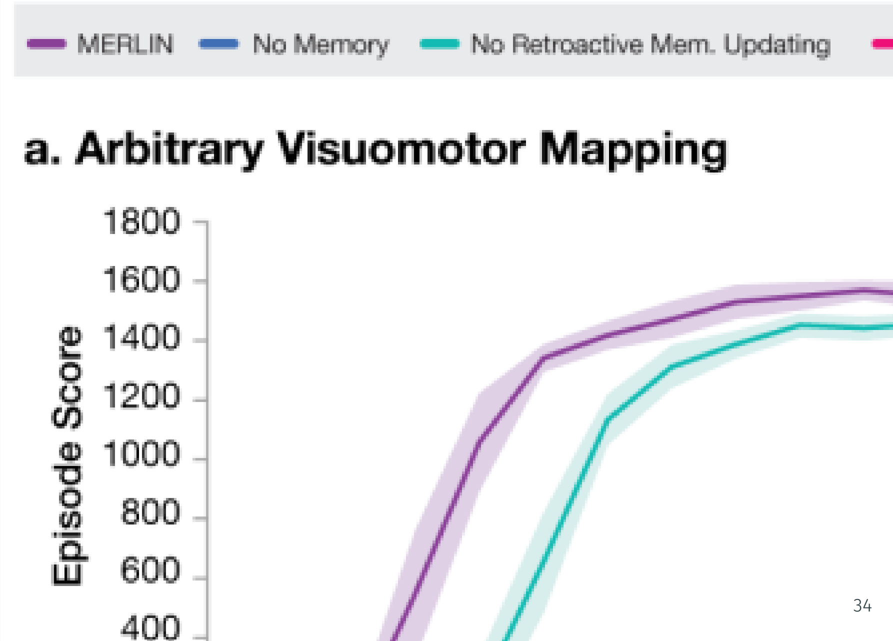


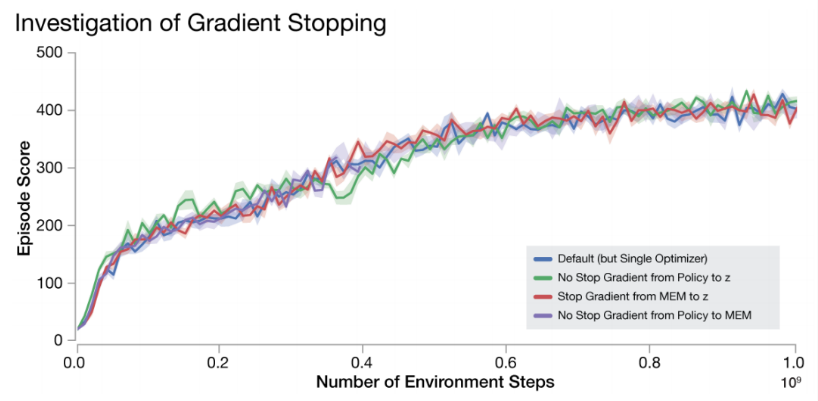

a
MERLIN
RL-LSTM
RL-MEM
Time to Reach Goal
Sub-Episodes
b
MERLIN MBP
MERLIN Convnet
RL-LSTM
RL-LSTM Convnet
RL-MEM
RL-MEM Convnet
Allocentric Goal Position Decoding Error
(Earth-Mover Manhattan Distance)
Sub-Episodes
c
MERLIN Return Prediction
Return Prediction Value
d
Return Prediction Error
MERLIN
RL-LSTM
RL-MEM
Agent Steps to Goal
e
Episode Score (Large Environment)
Return Cost Coefficient
f
% Variance Explained
Return Cost Coefficient


g
Observation
MERLIN
No Return Prediction Gradient to z
t=-20
t=-10
t=-5
Agent Steps to Goal
h
Read Head 1
Read Head 2
Read Head 3
i
Distance to the Goal (Pixels)
800
700
600
500
400
300
200
100
0
-50
-40
-30
-20
-10
0
Agent
Read Head 1
Read Head 2
Read Head 3
Agent Steps to Goal

MERLIN [12] - Latent Learning




MERLIN
No Memory
No Retroactive Mem. Updating
a. Arbitrary Visuomotor Mapping
Episode Score
1800
1600
1400
1200
1000
800
600
400


Investigation of Gradient Stopping
Episode Score
0
100
200
300
400
500
0.0
0.2
0.4
0.6
0.8
1.0
10^9
Number of Environment Steps
Default (but Single Optimizer)
No Stop Gradient from Policy to z
Stop Gradient from MEM to z
No Stop Gradient from Policy to MEM

R. Csordas and J. Schmidhuber.
Improving differentiable neural computers through memory masking, de-allocation, and link distribution sharpness control.
In *International Conference on Learning Representations*, 2019.

I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves.
Associative long short-term memory.
arXiv preprint arXiv:1602.03032, 2016.

M. A. Gluck and C. E. Myers.
Hippocampal mediation of stimulus representation: A computational theory.
Hippocampus, 3(4):491–516, 1993.

A. Graves, G. Wayne, and I. Danihelka.
Neural turing machines.
arXiv preprint arXiv:1410.5401, 2014.

A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al.
Hybrid computing using a neural network with dynamic external memory.
Nature, 538(7626):471, 2016.

C.-C. Hung, T. Lillicrap, J. Abramson, Y. Wu, M. Mirza, F. Carnevale, A. Ahuja, and G. Wayne.
Optimizing agent behavior over long time scales by transporting value.
arXiv preprint arXiv:1810.06721, 2018.

A. H. Marblestone, G. Wayne, and K. P. Kording.
Towards an integration of deep learning and neuroscience.
Frontiers in computational neuroscience, 10:94, 2016.

A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, and T. Lillicrap.
Relational recurrent neural networks.
In *Advances in Neural Information Processing Systems*, pages 7310–7321, 2018.

H. T. Siegelmann and E. D. Sontag.
On the computational power of neural nets.
Journal of computer and system sciences, 50(1):132–150, 1995.

S. Sukhbaatar, J. Weston, R. Fergus, et al.
End-to-end memory networks.
In *Advances in neural information processing systems*, pages 2440–2448, 2015.

O. Vinyals, M. Fortunato, and N. Jaitly.
Pointer networks.
In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

G. Wayne, C.-C. Hung, D. Amos, et al.
Unsupervised predictive memory in a goal-directed agent.
arXiv preprint arXiv:1803.10760, 2018.

J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov.
Towards ai-complete question answering: A set of prerequisite toy tasks.
arXiv preprint arXiv:1502.05698, 2015.

J. Weston, S. Chopgra, and A. Bordes.
Memory networks.
arXiv preprint arXiv:1410.3916, 2015.

Y. Wu, G. Wayne, A. Graves, and T. Lillicrap.
The kanerva machine: A generative distributed memory.
arXiv preprint arXiv:1804.01756, 2018.

Y. Wu, G. Wayne, K. Gregor, and T. Lillicrap.
Learning attractor dynamics for generative memory.
In *Advances in Neural Information Processing Systems*, pages 9401–9410, 2018.

G. Yang.
Lie access neural turing machine.
arXiv preprint arXiv:1602.08671, 2016.

Theme Credit

Get the source of this theme and the demo presentation from

`github.com/matze/mtheme`