Neural Map
Structured Memory for Deep RL
Structured Memory for Deep RL

Background
Neural Map: Location-Aware Memory
Incorporating Prior Knowledge with Memory
Background
Supervised Learning

• Most deep learning problems are posed as supervised learning problems: mapping and input to an output

• Environment is typically **static**

• Typically, outputs are assumed to be **independent** of each other
Environments for RL

- **Environments are dynamic** and change over time

- **Actions can affect the environment** with arbitrary time lags

- **Labels can be expensive** or difficult to obtain
Reinforcement Learning

• Instead of a label, the agent is provided with a **reward signal**:  
  - High reward == good behavior

• RL produces policies  
  - Map observations to actions  
  - Maximize long-term reward

• Allows learning purposeful behaviors in dynamic environments
Deep Reinforcement Learning

• Use a deep network to parameterize the policy
• Adapt parameters to maximize reward using:
  - Q-learning
  - Actor-Critic
  - Evolution Strategies

Learning 3-D game without memory
Chaplot, Lample, AAAI 2017

Reinforcement Learning: an Introduction, Sutton and Barto, 2014
Evolution Strategies, Salimans et al., 2017
Playing FPS games with deep RL, Chaplot & Lample, AAAI 2017
Deep Reinforcement Learning with Memory

• Can we learn an agent with a more advanced external memory?
  - Neural Turing Machines (Graves et al., 2014)
  - Differential Neural Computers (Graves et al., 2016)

• **Challenge:** Memory systems are difficult to train, especially using RL
Why is Memory Challenging?

• Suppose an agent is in a simple random maze:
  - Agent starts at top of map
  - An agent is shown an indicator near its initial state
  - The color of the indicator determines what the correct goal is

Oh et al, Control of Memory, Active Perception, and Action in Minecraft, ICML 2016
Why is Memory Challenging?

• Suppose an agent is in a simple random maze:
  - Agent starts at top of map
  - An agent is shown an indicator near its initial state
  - The color of the indicator determines what the correct goal is

Oh et al, Control of Memory, Active Perception, and Action in Minecraft, ICML 2016
Why is Memory Challenging?

- At the start, **no a priori knowledge** to store color into memory
- The following must hold:
  - Write color to memory at the start of maze
  - Never overwrite memory of the color over $T$ time steps
  - Find and enter the goal
- **Solution**: Write everything into memory
Neural Turing Machines (Graves et al., 2014)

- Basic Idea: Turn neural networks into ‘differentiable computers’ by giving them read-write access to external memory

Differentiable Neural Computer, Graves et al., Nature, 2016;
Neural Turing Machine, Graves et al., 2014
Architecture (Graves et al., 2014)

- The Controller is a neural network (recurrent or feedforward)
- The Heads select portions of the memory and read or write to them
- The Memory is a real-valued matrix

Everything is differentiable

Differentiable Neural Computer, Graves et al., Nature, 2016; Neural Turing Machine, Graves et al., 2014
Selective Attention

• Want to focus on the parts of memory the network will read and write to: need an attention model

• Use the controller outputs to parameterize a distribution (weighting) over the rows (locations) in the memory matrix

• The weighting defines content-based attention mechanism.
Addressing by Content

• A key vector $k$ is emitted by the controller and compared to
  - content of each memory location $M[i]$
  - using a similarity measure $S(.,.)$, e.g. cosine distance
  - then normalized with a softmax

• A ‘sharpness’ parameter is used to narrow the focus:
  - Finds the memories “closest” to the key

\[
w[i] = \frac{\exp(\beta S(k, M[i]))}{\sum_j \exp(\beta S(k, M[j]))}
\]
Addressing by Content

• Once the weightings are defined, each read head returns a read vector $r$ as input to the controller at the next time step

$$r = \sum_i w[i]M[i]$$

• Each write head receives an erase vector $e$ and adds vector $a$ from the controller - and then writes to modify the memory (like LSTM)

$$M[i] \leftarrow M[i](1 - w[i]e) + w[i]a$$
The NTM Copy Algorithm

**Diagram:**
- **Inputs** and **Outputs**
- **Write Vectors** and **Read Vectors**
- **Location** and **Time**
  - Write Weightings
  - Read Weightings

**Algorithm:**
```
initialize: move head to start location
while input delimiter not seen do
  receive input vector
  write input to head location
  increment head location by 1
end while
return head to start location
while true do
  read output vector from head location
  emit output
  increment head location by 1
end while
```
NTM Generalization: length 10 to 120

<table>
<thead>
<tr>
<th>targets</th>
<th>outputs</th>
<th>targets</th>
<th>outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="targets" /></td>
<td><img src="image2" alt="outputs" /></td>
<td><img src="image3" alt="targets" /></td>
<td><img src="image4" alt="outputs" /></td>
</tr>
</tbody>
</table>
Copy N Times

• Learning For Loop using content to jump, iteration to step, and a variable to count to N
Memory Networks

- Store \((key, value)\) representations for the last \(M\) frames

- At each time step:
  - Perform a read operation over their memory database
  - Write the latest percept into memory

Weston et al, Memory Networks, ICLR 2015
Miller et al, Key-Value Memory Networks., EMNLP 2016
Oh et al, Control of Memory, Active Perception, and Action in Minecraft, ICML 2016
Memory Networks

• Easy to learn: Just store as much as possible!
• Can be inefficient:
  - We need $M >$ time horizon of the task (can’t know this *a priori*)
  - We might store a lot of useless/redundant data
• Time/space requirements increase with $M$

Weston et al, Memory Networks, ICLR 2015
Miller et al, Key-Value Memory Networks., EMNLP 2016
Oh et al, Control of Memory, Active Perception, and Action in Minecraft, ICML 2016
Neural Map: Location-Aware Memory
Neural Map (Location-Aware Memory)

• Writable memory with a specific inductive bias:
  - Structure memory into $W \times W$ grid of $K$-dim cells
  - For every $(x, y)$ position, write to $(x', y')$ in the $W \times W$ grid

Parisotto, Salakhutdinov, ICLR 2018
Neural Map (Location-Aware Memory)

• Acts as a map that the agent fills out as it explores

• **Sparse Write:**
  
  - Inductive bias prevents the agent from overwriting its memory too often
  
  - Allow easier credit assignment over time

Parisotto, Salakhutdinov, ICLR 2018
Neural Map (Location-Aware Memory)

$M_t$ write $a_t$

$M_{t+1}$ write $a_{t+1}$

$w_t$ write $s_t$

$w_{t+1}$ write $s_{t+1}$

read with attention

Parisotto, Salakhutdinov, ICLR 2018
Neural Map: Operations

- Two read operations:
  - Global summarization
  - Context-based retrieval
- Sparse write only to agent position
- Both read and write vectors are used to compute policy

\[
\begin{align*}
  r_t &= \text{read}(M_t) \\
  c_t &= \text{context}(M_t, s_t, r_t) \\
  w^{(x_t,y_t)}_{t+1} &= \text{write}(s_t, r_t, c_t, M_t^{(x_t,y_t)}) \\
  M_{t+1} &= \text{update}(M_t, w^{(x_t,y_t)}_{t+1}) \\
  o_t &= [r_t, c_t, w^{(x_t,y_t)}_{t+1}] \\
  \pi_t(a|s) &= \text{Softmax}(f(o_t))
\end{align*}
\]
Neural Map: Global Read

• Reads from the entire neural map using a deep convolutional net
• Produces a vector that provides a global summary

\[
\begin{align*}
    r_t &= \text{read}(M_t) \\
    c_t &= \text{context}(M_t, s_t, r_t) \\
    w_{t+1}^{(x_t,y_t)} &= \text{write}(s_t, r_t, c_t, M_t^{(x_t,y_t)}) \\
    M_{t+1} &= \text{update}(M_t, w_{t+1}^{(x_t,y_t)}) \\
    o_t &= [r_t, c_t, w_{t+1}^{(x_t,y_t)}] \\
    \pi_t(a|s) &= \text{Softmax}(f(o_t))
\end{align*}
\]
Neural Map: Context Read

• Read operation using attention

\[ q_t = W[s_r, r_t] \]
\[ a_t^{(x,y)} = q_t \cdot M_t^{(x,y)} \]
\[ \alpha_t^{(x,y)} = \frac{e^{a_t^{(x,y)}}}{\sum_{(w,z)} e^{a_t^{(w,z)}}} \]
\[ c_t = \sum_{(x,y)} \alpha_t^{(x,y)} M_t^{(x,y)} \]

\[ \pi_t(a|s) = \text{Softmax}(f(o_t)) \]

Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015
Xu et al., Caption Generation with Visual Attention, ICML 2015
Neural Map: Context Read

• Read operation using attention

• Simple 2x2 memory $M_t$
  • Obtain query vector $q_t$ from state $s_t$ and global read $r_t$
Neural Map: Context Read

- Read operation using attention

\[ \alpha_t \]

- Dot product between query vector \( q_t \) and every memory cell

- Produces a similarity \( \alpha_t \)
Neural Map: Context Read

- Read operation using attention
- Dot product between query vector $q_t$ and every memory cell
- Produces a similarity $\alpha_t$
Neural Map: Context Read

- Read operation using attention
- Dot product between query vector $q_t$ and every memory cell
- Produces a similarity $\alpha_t$
Neural Map: Context Read

- Read operation using attention

- Dot product between query vector $q_t$ and every memory cell

- Produces a similarity $\alpha_t$
Neural Map: Context Read

- Read operation using attention
- Element-wise product between query similarities $\alpha_t$ and memory cells $M_t$

\[ q_t \rightarrow \alpha_t \rightarrow (s_t, r_t) \rightarrow M_t \rightarrow \alpha_t \odot M_t \]
Neural Map: Context Read

- Read operation using attention

\[
\alpha_t \odot M_t \sum \rightarrow c_t
\]

- Sum over all positions to obtain context read vector \( c_t \)

**Intuition:** Return vector \( c_t \) in memory \( M_t \) closest to the query \( q_t \)
Neural Map: Write

• Creates a new $k$-dim vector to write to the current position in the map

• Update the neural map at the current position with this new vector

\[
M_{t+1} = \text{update}(M_t, w(x_t, y_t))
\]
\[
\tau_t = \text{read}(M_t)
\]
\[
c_t = \text{context}(M_t, s_t, r_t)
\]
\[
w_{t+1} = \text{write}(s_t, r_t, c_t, M_t(x_t, y_t))
\]
\[
M_{t+1} = \text{update}(M_t, w_{t+1})
\]
\[
o_t = [r_t, c_t, w_{t+1}]
\]
\[
\pi_t(a|s) = \text{Softmax}(f(o_t))
\]
Neural Map: GRU Write Update

- Creates a new $k$-dim vector to write to the current position in the map
- Update the neural map at the current position with this new vector

$$
\begin{align*}
    r_t &= \text{read}(M_t) \\
    c_t &= \text{context}(M_t, s_t, r_t) \\
    w_{t+1}(x_t, y_t) &= \text{write}(s_t, r_t, c_t, M_t(x_t, y_t)) \\
    M_{t+1} &= \text{update}(M_t, w_{t+1}(x_t, y_t)) \\
    o_t &= [r_t, c_t, w_{t+1}(x_t, y_t)] \\
    \pi_t(a|s) &= \text{Softmax}(f(o_t))
\end{align*}
$$

Chung et al., Gated Recurrent Neural Networks, 2014
Neural Map: Output

- Output the read vectors and what we wrote
- Use those features to compute a policy

\[
\begin{align*}
    r_t & = \text{read}(M_t) \\
    c_t & = \text{context}(M_t, s_t, r_t) \\
    w_{t+1}^{(x_t,y_t)} & = \text{write}(s_t, r_t, c_t, M_t^{(x_t,y_t)}) \\
    M_{t+1} & = \text{update}(M_t, w_{t+1}^{(x_t,y_t)}) \\
    o_t & = [r_t, c_t, w_{t+1}^{(x_t,y_t)}] \\
    \pi_t(a|s) & = \text{Softmax}(f(o_t))
\end{align*}
\]
Random Maze with Indicator

- Results are robust with respect to small noise in the \((x, y)\)-position of the agent
Random Maze with Indicator
Egocentric Neural Map

- Problem with Neural Map: it requires mapping from \((x, y)\) to \((x', y')\)
  - We need to have already solved localization
- Obtain a map which is egocentric:
  - The agent always writes to the center of the map
  - When the agent moves, the entire map moves by the opposite amount
Intelligent Agents

Learned External Memory

Knowledge Base

write

read

reason communicate

action $a_t$

reward $r_t$

observation/state $o_t$
Go to the red short torch
Learning to Execute Instructions

Go to the green torch

Natural Language Instruction ($L$)

Image ($I_t$)

GRU Network $g(L; \theta_{GRU})$

Conv Network $f(I_t; \theta_{conv})$

Instruction Representation $x_L = f(L; \theta_{gru})$

Image Representation $x_t = f(I_t; \theta_{conv})$

Multimodal Fusion ($M$)

State Representation $M(x_L, x_t)$

Policy Learning Module

Policy $\Pi(a|I_t, L)$

Chaplot et al., Gated-Attention Architectures for Task-oriented Language Grounding, AAAI 2017

Hermann et al., Grounded Language Learning in a Simulated 3-D world, 2017
Active Neural Localization

$t = 1$

Belief

Ground Truth

Chaplot, Parisotto, Salakhutdinov, ICLR 2018
Discussion

• Can we extend to multi-agent domains?
  - Multiple agents communicating through shared memory.

• Can we train an agent to learn how to simultaneously localize and map its environment using the Neural Map?
  - Solves problem of needing an oracle to supply \((x, y)\) position

• Can we structure neural maps into a multi-scale hierarchy?
  - Each scale will incorporate longer range information
Thank you