Neural Map
Structured Memory for Deep RL

Emilio Parisotto
eparisot@andrew.cmu.edu
PhD Student
Machine Learning Department
Carnegie Mellon University
Supervised Learning

• Most deep learning problems are posed as supervised learning problems.

• The model is trained to map from an input to an action:
  • Describe what is in an image.

• Environment is typically static:
  • It does not change over time.

• Actions are assumed to be independent of another:
  • E.g. labelling one image does not affect the next one.
Environments are not always well-behaved

- Environments are dynamic and change over time:
  - An autonomous agent has to handle new environments.

- Actions can affect the environment with arbitrary time lags:
  - Buying a stock $\rightarrow$ years in the future can lose all money

- Labels can be expensive/difficult to obtain:
  - Optimal actuations of a swimming octopus robot.
Reinforcement Learning: Closing the Loop

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

• Reinforcement Learning produces policies:
  • Behaviors that Map observations to actions
  • Maximize long-term reward

• Allows learning purposeful behaviours in dynamic environments.
Deep Reinforcement Learning

• Use a deep network to do parameterize the policy

• Adapt parameters to maximize reward using:
  - Q-learning (Mnih et al., 2013)
  - Actor-Critic (Mnih et al., 2016)

• How well does it work?
Deep Reinforcement Learning

Learning to play deathmatch

Chaplot, Lample, AAAI 2017
Current Memory for RL agents

- Deep RL does extremely well on reactive tasks.
- But typically has an effective memory horizon of less than 1 second.
- Almost all interesting problems are partially observable:
  - 3D games (with long-term objectives)
  - Self-driving cars (partial occlusion)
- Memory structures will be crucial to scale up to partially-observable tasks.

Diagram:
- Action $a_t$
- Reward $r_t$
- Observation / State $o_t$
- Deep Neural Net $h^1, h^2, h^3$
External Memory?

• Current memory structures are usually simple:
  • Add an LSTM layer to the network

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

• Challenge: memory systems are difficult to train, especially using RL.
Why Memory is Challenging: Write Operations

Suppose an agent is in a simple maze:

• Agent starts at top of map.
• An agent is shown a color near its initial state.
• This color determines what the correct goal is.
Why Memory is Challenging: Write Operations

Suppose an agent is in a simple maze:

• Agent starts at top of map.
• An agent is shown a color near its initial state.
• This color determines what the correct goal is.
Why Memory is Challenging: Write Operations

Suppose an agent is in a simple maze:

- Agent starts at top of map.
- An agent is shown a color near its initial state.
- This color determines what the correct goal is.
Why Writing to Memory is Challenging

At the start, *no a priori* knowledge to store color into memory.

Needs the following to hold:
1. Write color to memory at start of maze.
2. Never overwrite memory of color over ‘T’ time steps.
3. Find and enter the goal.

All conditions must hold or else episode is useless.
- Provides little new information to the agent.

**Solution:** Write everything into memory!
Memory Network

• A class of structures that were recently shown to learn difficult maze-based memory tasks

• These systems just store (key,value) representations for the M last frames

• At each time step, they:
  1. Perform a read operation over their memory database.
  2. Write the latest percept into memory.

Oh et al., 2016
Memory Network Difficulties

Easy to learn → never need to guess on what to store in memory.
• Just store as much as possible!

But 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 $M$
Neural Map (Location-Aware Memory)

• Writeable memory with a specific inductive bias:
  • We structure the memory into a WxW grid of K-dim cells.
  • For every (x,y) in the environment, we write to (x’,y’) in the WxW grid.
Neural Map (Location-Aware Memory)

• Writeable memory with a specific inductive bias:
  • We structure the memory into a WxW grid of K-dim cells.
  • For every (x,y) in the environment, we write to (x’,y’) in the WxW grid.

• 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.
  • Allowing easier credit assignment over time.
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.

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

• Reads from the entire neural map using a deep convolutional network.

• Produces a vector that provides a global summary.

\[ r_t = \text{read}(M_t) \]
Neural Map: Context Read

- Associative read operation.

\[
q_t = W[s_t, r_t]
\]
\[
a_t^{(x,y)} = q_t \cdot M_t^{(x,y)}
\]
\[
\alpha_t^{(x,y)} = \frac{e^{a_t^{(x,y)}}}{\sum_{(w,z)} e^{a_t^{(w,z)}}}
\]
\[
c_t = \sum_{(x,y)} \alpha_t^{(x,y)} M_t^{(x,y)},
\]
\[
r_t = \text{read}(M_t)
\]
\[
c_t = \text{context}(M_t, s_t, r_t)
\]
\[
w_{t+1}^{(x_t,y_t)} = \text{write}(s_t, r_t, c_t, M_t^{(x_t,y_t)})
\]
\[
M_{t+1} = \text{update}(M_t, w_{t+1}^{(x_t,y_t)})
\]
\[
o_t = [r_t, c_t, w_{t+1}^{(x_t,y_t)}]
\]
\[
\pi_t(a|s) = \text{Softmax}(f(o_t)),
\]
Neural Map: Context Read

- Simple 2x2 memory $M_t$
- Color represents memory the agent wrote.
Neural Map: Context Read

- Query vector $q_t$ from state $s_t$ and global read $r_t$
\( \alpha_t \) • Dot product between query \( q_t \) and every memory cell.  
• Produces a similarity \( \alpha_t \)
Neural Map: Context Read

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

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

- Element-wise product between query-similarities $\alpha_t$ and memory cells $M_t$
Neural Map: Context Read

\[ a_t \] \quad \alpha_t \odot M_t \quad \sum \quad c_t \]

- Sum over all 4 positions to get context read vector \( c_t \)
Neural Map: Context Read

Intuitively:
- Return vector $c_t$ in memory $M_t$ closest to query $q_t$
Neural Map: Write

- Creates a new k-dim vector to write to the current position in the neural map.

- Update the neural map at the current position with this new vector.

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

\[ M_{t+1} = \text{update}(M_t, w_{t+1}^{(x_t,y_t)}) \]

\[ w_{t+1}^{(x_t,y_t)} = \text{write}(s_t, r_t, c_t, M_t^{(x_t,y_t)}) \]

\[ c_t = \text{context}(M_t, s_t, r_t) \]

\[ r_t = \text{read}(M_t) \]

\[ o_t = [r_t, c_t, w_{t+1}^{(x_t,y_t)}] \]

\[ \pi_t(a|s) = \text{Softmax}(f(o_t)), \]
Neural Map: GRU Write Update

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

Chung et al., 2014
Neural Map: Output

• Output the read vectors and what we wrote.

• Use those features to calculate a policy.

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

Real Map
(Not Visible)
3xKxK

Input State
(Partially observable)
3x15x3
Random Maze Results

<table>
<thead>
<tr>
<th>Agent</th>
<th>Goal-Search</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7-11 Train</td>
<td>13-15 Train</td>
<td>Total</td>
<td>7-11 Test</td>
<td>13-15 Test</td>
<td>Total</td>
</tr>
<tr>
<td>Random</td>
<td>41.9%</td>
<td>25.7%</td>
<td>38.1%</td>
<td>46.0%</td>
<td>29.6%</td>
<td>38.8%</td>
</tr>
<tr>
<td>LSTM</td>
<td>60.6%</td>
<td>41.8%</td>
<td>59.3%</td>
<td>65.5%</td>
<td>47.5%</td>
<td>57.4%</td>
</tr>
<tr>
<td>MemNN-32</td>
<td>85.1%</td>
<td>58.2%</td>
<td>77.8%</td>
<td>92.6%</td>
<td>69.7%</td>
<td>83.4%</td>
</tr>
<tr>
<td>Neural Map</td>
<td>92.4%</td>
<td>80.5%</td>
<td>89.2%</td>
<td>93.5%</td>
<td>87.9%</td>
<td>91.7%</td>
</tr>
<tr>
<td>Neural Map (GRU)</td>
<td><strong>97.0%</strong></td>
<td><strong>89.2%</strong></td>
<td><strong>94.9%</strong></td>
<td><strong>97.7%</strong></td>
<td><strong>94.0%</strong></td>
<td><strong>96.4%</strong></td>
</tr>
</tbody>
</table>
2D Maze Visualization
Task: Doom Maze

(a) Green Torch → Green Tower

(b) Red Torch → Red Tower

ViZDoom (Kempka et al., 2016)
## Doom Maze Results

<table>
<thead>
<tr>
<th>Agent</th>
<th>Training Map</th>
<th>Unseen Maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>20.9%</td>
<td>22.1%</td>
</tr>
<tr>
<td>MemNN</td>
<td>68.2%</td>
<td>60.3%</td>
</tr>
<tr>
<td>LSTM</td>
<td>69.2%</td>
<td>52.4%</td>
</tr>
<tr>
<td>LSTM+Neural Map (GRU)</td>
<td>78.3%</td>
<td>66.6%</td>
</tr>
</tbody>
</table>
Egocentric Neural Map

• Problem with Neural Map: it requires mapping from \((x,y)\) to \((x',y')\)
  • Means we need to have already solved localization

• Another way might be to get 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.
Conclusion

• Designed a novel memory architecture suited for DRL agents.
  • Spatially structured memory useful for navigation tasks.
  • Sparse write which simplifies credit assignment.

• Demonstrated its ability to store information over long time lags.
  • Surpassed performance of several previous memory-based agents.
Future Directions

• 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

Contact Information:
Emilio Parisotto (eparisot@andrew.cmu.edu)
Extra Slides
What does the Neural Map learn to store?

Observations

True State

Context Read Distribution
Neural Map: Summary

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