10703 Deep Reinforcement Learning and Control
Russ Salakhutdinov
Machine Learning Department
rsalakhu@cs.cmu.edu

Memory-Augmented RL
Used Materials

• **Disclaimer**: Much of the material and slides for this lecture were provided by Alex Graves and Emilio Parisotto
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
  - Translate a sentence from English to French

- Environment is typically static:
  - It does not change over time

- Actions are assumed to be independent of another:
  - E.g. labeling one image does not affect the next one
Challenges

‣ Environment is dynamic and changes 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 (RL)

- 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

- RL allows learning purposeful behaviors in dynamic environments.

![Diagram of Reinforcement Learning Process]
External Memory

- Deep RL does extremely well on reactive tasks
- But typically has a short effective memory horizon
- Can we learn an agent with an external memory?
Current Memory for RL agents

- Current memory structures for RL are usually simple:
  - Just add an LSTM layer to the network

- Almost every kind of interesting problem is partially observable:
  - 3D games (with long-term objectives)
  - Autonomous robots (partial occlusion)

- Memory structures will be crucial to scaling up deep RL agents to partially-observable and non-reflexive tasks

- **Challenge**: complicated memory systems are difficult to train, especially using RL objectives.
Random Maze with Indicator

- **Indicator**: Either blue or pink
  - If blue, find the green block
  - If yellow, find the red block
- **Negative reward** if agent does not find correct block in N steps or goes to wrong block.

Parisotto, Salakhutdinov, 2017
Neural Turing Machines
Graves et al., 2014

- **Basic idea**: turn neural networks into ‘differentiable computers’ by giving them read-write access to external memory.

\[ \text{‘CPU’} \quad + \quad \text{Memory} \quad = \quad \text{computer that learns programs from examples} \]

(neural net that separates computation from memory)
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
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.

Differentiable Neural Computer, Graves et al., Nature, 2016;
Neural Turing Machine, Graves et al., 2014
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’ $\beta$ is used to narrow the focus:
  - Finds the memories “closest” to the key

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

Differentiable Neural Computer, Graves et al., Nature, 2016; Neural Turing Machine, Graves et al., 2014
Reading and Writing

› 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 an add vector \( a \) from the controller
  - and then writes to modify the memory (like LSTM)

\[
M[i] \leftarrow M[i](1 - w[i]e) + w[i]a
\]

Differentiable Neural Computer, Graves et al., Nature, 2016; Neural Turing Machine, Graves et al., 2014
The NTM Copy 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
  while true do
    read output vector from head location
    emit output
    increment head location by 1
  end while
end while
NTM Generalization: length 10 to 120
Copy N Times

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

- 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

Oh et al., 2016, Weston et al. 2014
Memory Networks

- At each time step, they:
  - Predict a “context” vector from the current state
  - Match the context vector to the M last (key) features to get a probability distribution over the M last (value) features
  - Use the context vector and a weighted average of the value features to predict Q-values

Oh et al., 2016, Weston et al. 2014
Challenges

‣ Memory networks are easier to learn because the agent never needs to make the initial guess on what to store in memory.
  - No need to learn “what to write” to memory, just store as much as possible!

‣ The state at each time step is a temporal convolution of the last $M$ frames, with the weights being defined by the context vector.

‣ This is 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)

- Neural Map: NTM with a specific inductive bias:
  - We structure the memory into a $K \times K$ grid instead of $M$ cells.
  - For every $(x,y)$ position in the environment, we write to a corresponding position $(x',y')$ in the $K \times K$ grid.

- Acts as a map that the agent fills out as it explores.

- **Sparse Write**: The inductive bias prevents the agent from overwriting its memory too often, allowing easier credit assignment over time.

Parisotto, Salakhutdinov, 2017
Random Maze with Indicator

$M_t$  
\[ \text{Write} \]
$w_t$  
\[ \text{Read with Attention} \]
$O_t$  

$M_{t+1}$  
\[ \text{Write} \]
$w_{t+1}$  
\[ \text{Read with Attention} \]
$O_{t+1}$  

$a_t$  

$a_{t+1}$  

Parisotto, Salakhutdinov, 2017
Neural Map: Operations

- $M_t$ is a k channel WxW image representing the environment.

- **Two read operations:**
  - **Global** summarization
  - **Context-based** retrieval

- **Sparse write** only to map position where agent current is.

- **Outputs** of both reads and the write vector are passed to next layer.

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

Parisotto, Salakhutdinov, 2017
Neural Map: Global Read

- $M_t$ is a $k$ channel $W \times W$ image representing the environment.

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

- 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

- $M_t$ is a $k$ channel $W \times W$ image representing the environment.
- **Context 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: Write

- $M_t$ is a $k$ channel $W \times W$ image representing the environment.

- Create a new vector to write to the current position in the neural map.

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

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

- $M_t$ is a $k$ channel $W \times W$ image representing the environment.

- Output the read vectors and what we wrote.

- Use those features to calculate a policy.

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

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

\[
\begin{align*}
    r_{t+1}^{(x_t,y_t)} &= \sigma(W_r[s_t, r_t, c_t, M_t^{(x_t,y_t)})]
    \\
    \hat{w}_{t+1}^{(x_t,y_t)} &= \tanh(W_{\hat{h}}[s_t, r_t, c_t] + U_{\hat{h}}(r_{t+1}^{(x_t,y_t)} \odot M_t^{(x_t,y_t)}))
    \\
    \hat{z}_{t+1}^{(x_t,y_t)} &= \sigma(W_{\hat{z}}[s_t, r_t, c_t, M_t^{(x_t,y_t)}])
    \\
    w_{t+1}^{(x_t,y_t)} &= (1 - \hat{z}_{t+1}^{(x_t,y_t)}) \odot M_t^{(x_t,y_t)} + \hat{z}_{t+1}^{(x_t,y_t)} \odot \hat{w}_{t+1}^{(x_t,y_t)},
\end{align*}
\]

- Instead of a hard write each time step, we can use gating.
- Used GRU-like update equations.
- Greatly improved learning speed and stability.
Start in a specific position in a random maze.

Near the start position there is an indicator.
- Either green or yellow

If green, find the blue block.

If yellow, find the red block.

Failure (negative reward) if agent doesn’t find correct block in N steps or goes to wrong block.
Random Maze with Indicator

Real Map
(Not Visible)
3xKxK

Input State
(Partially observable)
3x5x3
Random Maze with Indicator
**Results: Random Maze**

<table>
<thead>
<tr>
<th>Agent</th>
<th>Goal-Search</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>13-15</td>
<td>Total</td>
<td>Test</td>
<td>13-15</td>
<td>Total</td>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
</tbody>
</table>

![Graph showing performance over epochs](image)
Neural Map: Learning to Store

Observations

True State

Context Read Distribution
## 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><strong>78.3%</strong></td>
<td><strong>66.6%</strong></td>
</tr>
</tbody>
</table>
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