Deep Reinforcement Learning with Forward Prediction, Memory, and Hierarchy

Honglak Lee
Google Brain / U. Michigan

Joint work with
Junhyuk Oh, Ruben Villegas, Xiao Xiao Guo, Jimei Yang, Sungryull Sohn, Xunyu Lin, Valliappa Chockalingam, Rick Lewis, Satinder Singh, Pushmeet Kohli
Overview

• Deep RL with Forward Prediction

• Deep RL with Memory

• Deep RL with Hierarchy
Action-conditional video prediction with Deep Architectures

• Motivation:
  – Deep convolutional encoder-decoder architecture modulated by control actions
  – End-to-end multi-step prediction
  – Application to reinforcement learning (e.g., Atari games)

• Results:
  – long-term video prediction (30-500 steps) for atari games
  – Informed exploration: Faster learning and improved performance

• Related work on video prediction
  – [Ranzato et al., 2014], [Srivastava et al., 2015], [Mathieu et al, 2015], [Finn el al., 2016]
Action-conditional video prediction with Deep Architectures

- Convolutional Neural Networks (CNN)
Action-conditional video prediction with Deep Architectures

- CNN combined with Long short-term memory (LSTM)
Freeway: 100 steps Predictions (LSTM)

Video: https://www.youtube.com/watch?v=4e-PqfpS8_4
More at: https://sites.google.com/a/umich.edu/junhyuk-oh/action-conditional-video-prediction
Seaquest: Multi-Step Predictions

https://sites.google.com/a/umich.edu/junhyuk-oh/action-conditional-video-prediction
Space Invaders: Multi-Step Predictions

https://sites.google.com/a/umich.edu/junhyuk-oh/action-conditional-video-prediction
Ms Packman: Multi-Step Predictions

https://sites.google.com/a/umich.edu/junhyuk-oh/action-conditional-video-prediction
Informed Exploration

- **Idea**: choose an exploratory action that leads to a less-frequently-visited frame

- **Method**: estimate visit-frequency by comparing predictive frames with previous frames
  
  - Store the most recent $d$ frames in a *trajectory memory*.
  
  - The predictive model is used to get the next frame ($x^{(a)}$) for every action.
  
  - Estimate visit-frequency using Gaussian kernels.

\[
\begin{align*}
  n_D(x^{(a)}) &= \sum_{i=1}^{d} k(x^{(a)}, x^{(i)});
  
  k(x, y) &= \exp(-\sum_j \min(\max((x_j - y_j)^2 - \delta, 0), 1)/\sigma)
\end{align*}
\]

- Choose an action that leads to the frame with the smallest visit-frequency: 
  \[
  \text{argmin}_a n_D \left( x^{(a)} \right)
  \]
Informed exploration with future predictions

- **Idea:** choose an exploratory action that leads to a less-frequently-visited frame

(Oh et al., NIPS 2015)

<table>
<thead>
<tr>
<th>Model</th>
<th>Seaquest</th>
<th>S. Invaders</th>
<th>Freeway</th>
<th>QBert</th>
<th>Ms Pacman</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN - Random exploration</td>
<td>13119 (538)</td>
<td>698 (20)</td>
<td>30.9 (0.2)</td>
<td>3876 (106)</td>
<td>2281 (53)</td>
</tr>
<tr>
<td>DQN - Informed exploration</td>
<td>13265 (577)</td>
<td>681 (23)</td>
<td>32.2 (0.2)</td>
<td>8238 (498)</td>
<td>2522 (57)</td>
</tr>
</tbody>
</table>

Average Game Score over 100 plays with DQN

Video demo: [https://www.youtube.com/watch?v=DLIWo16r5LA](https://www.youtube.com/watch?v=DLIWo16r5LA)
Using Predictions to Improve Exploration in DQN
(18) Action Representations: Correlations for Seaquest
Emergence of disentangling

Prev. frame

Prediction

Next frame

Action factors

Non-action factors
Improving forward prediction with motion/content decomposition

Decomposing Motion and Content for Natural Video Sequence Prediction.
Experimental results

KTH dataset

Weizmann dataset
Experimental results

UCF-101 dataset
Experimental results

UCF-101 dataset
Experimental results

Figure 4: Quantitative comparison between our model, convolutional LSTM Shi et al. (2015), and Mathieu et al. (2015). Given 4 input frames, the models predict 8 frames recursively, one by one.
Long-term future prediction with structures

Learning to Generate Long-term Future via Hierarchical Prediction.
Ruben Villegas, Jimei Yang, Yuliang Zou, Sungryull Sohn, Xunyu Lin, Honglak Lee. Arxiv (coming soon)
Experimental results

• End to end prediction (Penn Action dataset)
Experimental results

• End to end prediction (Human 3.6M dataset)
Experimental results

• Prediction when provided with GT landmarks
Combining Active Perception (partial observation) and Memory

- **Active Perception**: Can the agent learn to use its perceptual actions to collect useful information in partially observable environments?

- **Memory**: Can the agent remember useful information in partially observable environments?

- **Generalization**: Can the agent generalize to unseen and/or larger environments given the same task?

→ These are hard to examine in the existing benchmarks.

Minecraft domain

- Minecraft provides a rich environment for RL
  - Flexible 3D environment (e.g., moving, collecting, building)
  - We can define many tasks and control the level of tasks
  - Deep partial observability due to the first-person-view observations

Control of Memory, Active Perception, and Action in Minecraft.

Overview of architectures

(a) DQN  (b) DRQN  (c) MQN  (d) RMQN  (e) FRMQN

New memory based architectures

Related work:
I-Maze: Task Description

• The indicator has an equal chance to be green or yellow.
  – Green indicator: Blue gives +1
  – Yellow indicator: Red gives +1

• Fixed sizes of maps (\{5,7,9\}) are given during training.

• Q) Can the agent generalize to unseen sizes of maps?
Why context-dependent memory retrieval?

- The importance of a past event depends on the current context.
- ex) the color of the indicator (yellow) is important only when the agent finds a goal block (blue/red) and decides whether to visit it or not.
Why context-dependent memory retrieval?

• The importance of a past event depends on the current context.
• ex) the color of the indicator (yellow) is important only when the agent finds a goal block (blue/red) and decides whether to visit it or not.
I-Maze: Result

<table>
<thead>
<tr>
<th>SIZE</th>
<th>TRAIN</th>
<th>DQN</th>
<th>DRQN</th>
<th>MQN</th>
<th>RMQN</th>
<th>FRMQN</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>✓</td>
<td>92.1(1.5)</td>
<td>94.8(1.5)</td>
<td>87.2(2.3)</td>
<td>89.2(2.4)</td>
<td>96.9(1.0)</td>
</tr>
<tr>
<td>5</td>
<td>✓</td>
<td><strong>99.3</strong>(0.5)</td>
<td>98.2(1.1)</td>
<td>96.2(1.0)</td>
<td>98.6(0.5)</td>
<td><strong>99.3</strong>(0.7)</td>
</tr>
<tr>
<td>6</td>
<td>✓</td>
<td><strong>99.4</strong>(0.4)</td>
<td>98.2(1.0)</td>
<td>96.0(1.0)</td>
<td>99.0(0.4)</td>
<td><strong>99.7</strong>(0.3)</td>
</tr>
<tr>
<td>7</td>
<td>✓</td>
<td>99.6(0.3)</td>
<td>98.8(0.8)</td>
<td>98.0(0.6)</td>
<td>98.8(0.5)</td>
<td><strong>100.0</strong>(0.0)</td>
</tr>
<tr>
<td>8</td>
<td>✓</td>
<td>99.3(0.4)</td>
<td>98.3(0.8)</td>
<td>98.3(0.5)</td>
<td>98.0(0.8)</td>
<td><strong>100.0</strong>(0.0)</td>
</tr>
<tr>
<td>9</td>
<td>✓</td>
<td>99.0(0.5)</td>
<td>98.4(0.6)</td>
<td>98.0(0.7)</td>
<td>94.6(1.8)</td>
<td><strong>100.0</strong>(0.0)</td>
</tr>
<tr>
<td>10</td>
<td>✓</td>
<td>96.5(0.7)</td>
<td>97.4(1.1)</td>
<td>98.2(0.7)</td>
<td>87.5(2.6)</td>
<td><strong>99.6</strong>(0.3)</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>50.7(0.9)</td>
<td>83.3(3.2)</td>
<td><strong>96.7</strong>(1.3)</td>
<td>89.8(2.4)</td>
<td><strong>97.4</strong>(1.1)</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>48.3(1.0)</td>
<td>63.6(3.7)</td>
<td>97.2(0.9)</td>
<td>96.3(1.2)</td>
<td><strong>98.8</strong>(0.5)</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>48.1(1.0)</td>
<td>57.6(3.7)</td>
<td><strong>98.2</strong>(0.7)</td>
<td>90.3(2.5)</td>
<td><strong>98.4</strong>(0.6)</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>48.6(1.0)</td>
<td>60.5(3.6)</td>
<td><strong>97.9</strong>(0.9)</td>
<td>87.1(2.4)</td>
<td><strong>98.1</strong>(0.6)</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>49.5(1.2)</td>
<td>59.0(3.4)</td>
<td><strong>95.0</strong>(1.1)</td>
<td>84.0(3.2)</td>
<td><strong>94.8</strong>(1.2)</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>46.6(1.2)</td>
<td>59.2(3.6)</td>
<td>77.2(4.2)</td>
<td>71.3(5.0)</td>
<td><strong>89.0</strong>(2.6)</td>
</tr>
</tbody>
</table>

- All architectures perform well on the training set of maps.
- Our architectures generalize better to larger I-mazes than DQN and DRQN architectures.
I-Maze: Memory Retrieval Visualization
I-Maze: Memory Retrieval Visualization

- Our agent (FRMQN) looks at the indicator.

Time

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory attention drawn to past observations
I-Maze: Memory Retrieval Visualization

- Our agent (FRMQN) looks at the indicator.

Memory attention drawn to past observations
I-Maze: Memory Retrieval Visualization

• Our agent goes to the end of the corridor.

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
I-Maze: Memory Retrieval Visualization

• Our agent goes to the end of the corridor.
→ It does not sharply retrieve to the indicator along the way.
I-Maze: Memory Retrieval Visualization

- Our agent sharply retrieves the indicator information only when it has to decide which way to go at the end of the corridor.
I-Maze: Demo
Pattern Matching: Task Description

• There are two 3x3 rooms with color patterns that have an equal chance to be identical or different.
  – If two patterns are identical: Blue gives +1
  – Otherwise: Red gives +1

• A subset of visual patterns is given during training.

• Q) Can the agent generalize to unseen visual patterns?
Pattern Matching: Result

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Unseen</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>62.9% (±3.4%)</td>
<td>60.1% (±2.8%)</td>
</tr>
<tr>
<td>DRQN</td>
<td>49.7% (±0.2%)</td>
<td>49.2% (±0.2%)</td>
</tr>
<tr>
<td>MQN</td>
<td>99.0% (±0.2%)</td>
<td>69.3% (±1.5%)</td>
</tr>
<tr>
<td>RMQN</td>
<td>82.5% (±2.5%)</td>
<td>62.3% (±1.5%)</td>
</tr>
<tr>
<td>FRMQN</td>
<td><strong>100.0%</strong> (±0.0%)</td>
<td><strong>91.8%</strong> (±1.0%)</td>
</tr>
</tbody>
</table>

- DQN/DRQN/RMQN tend to learn a sub-optimal policy that goes to any goal blocks regardless of visual patterns.
- Although MQN performs well on the training maps, it fails to generalize to unseen visual patterns.
- FRMQN generalizes better to unseen maps across different runs.
Pattern Matching: Demo
Random Maze: Task Description

• **Single Goal**
  – Blue gives +1, Red gives -1

• **Sequential Goals**
  – Red → Blue

• **Single Goal with Indicator**
  – Green indicator: Blue gives +1
  – Yellow indicator: Red gives +1

• **Sequential Goals with Indicator**
  – Green indicator: Red → Blue
  – Yellow indicator: Blue → Red
### Random Maze: Result

<table>
<thead>
<tr>
<th>Task</th>
<th>Type</th>
<th>Size</th>
<th>DQN</th>
<th>DRQN</th>
<th>MQN</th>
<th>RMQN</th>
<th>FRMQN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single</strong></td>
<td>Train</td>
<td>4-8</td>
<td>0.31</td>
<td>0.45</td>
<td>0.01</td>
<td>0.49</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>Unseen</td>
<td>4-8</td>
<td>0.22</td>
<td>0.23</td>
<td>0.02</td>
<td>0.30</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td>Unseen-L</td>
<td>9-14</td>
<td><strong>-0.28</strong></td>
<td><strong>-0.40</strong></td>
<td><strong>-0.63</strong></td>
<td><strong>-0.28</strong></td>
<td><strong>-0.28</strong></td>
</tr>
<tr>
<td><strong>Seq</strong></td>
<td>Train</td>
<td>5-7</td>
<td><strong>-0.60</strong></td>
<td><strong>-0.08</strong></td>
<td><strong>-0.48</strong></td>
<td>0.21</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>Unseen</td>
<td>5-7</td>
<td><strong>-0.66</strong></td>
<td><strong>-0.54</strong></td>
<td><strong>-0.59</strong></td>
<td><strong>-0.13</strong></td>
<td><strong>-0.18</strong></td>
</tr>
<tr>
<td></td>
<td>Unseen-L</td>
<td>8-10</td>
<td><strong>-0.82</strong></td>
<td><strong>-0.89</strong></td>
<td><strong>-0.77</strong></td>
<td><strong>-0.43</strong></td>
<td><strong>-0.42</strong></td>
</tr>
<tr>
<td><strong>Single+I</strong></td>
<td>Train</td>
<td>5-7</td>
<td><strong>-0.04</strong></td>
<td>0.23</td>
<td>0.11</td>
<td>0.34</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td>Unseen</td>
<td>5-7</td>
<td><strong>-0.41</strong></td>
<td><strong>-0.46</strong></td>
<td><strong>-0.46</strong></td>
<td><strong>-0.27</strong></td>
<td><strong>-0.23</strong></td>
</tr>
<tr>
<td></td>
<td>Unseen-L</td>
<td>8-10</td>
<td><strong>-0.74</strong></td>
<td><strong>-0.98</strong></td>
<td><strong>-0.66</strong></td>
<td><strong>-0.39</strong></td>
<td><strong>-0.43</strong></td>
</tr>
<tr>
<td><strong>Seq+I</strong></td>
<td>Train</td>
<td>4-6</td>
<td><strong>-0.13</strong></td>
<td>0.25</td>
<td><strong>-0.07</strong></td>
<td>0.37</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>Unseen</td>
<td>4-6</td>
<td><strong>-0.58</strong></td>
<td><strong>-0.65</strong></td>
<td><strong>-0.71</strong></td>
<td><strong>-0.32</strong></td>
<td><strong>-0.28</strong></td>
</tr>
<tr>
<td></td>
<td>Unseen-L</td>
<td>7-9</td>
<td><strong>-0.95</strong></td>
<td><strong>-1.14</strong></td>
<td><strong>-1.04</strong></td>
<td><strong>-0.60</strong></td>
<td><strong>-0.54</strong></td>
</tr>
</tbody>
</table>

- RMQN and FRMQN perform better than the other architectures on most of the tasks and maps.
Random Maze: Result

- RMQN and FRMQN perform better than the other architectures on most of the tasks and maps.
- The performance gap is larger on unseen sets of maps.
Random Maze: Demo

SINGLE GOAL (TRAINING MAPS)
Hierarchical Deep RL for task generalization

- Following unseen sequence composition of instructions
- Executing unseen tasks (action/object combination)
- Needs to deal with interruptions (random events), long delayed reward

Training
1. Visit pig
2. Pick up 3 sheep
3. Transform greenbot
4. Pick up horse

Testing
1. Pick up pig
2. Visit sheep
3. Transform cat
4. Transform 3 sheep
5. Pick up greenbot
6. Pick up 2 pig
7. Transform 2 sheep
8. Transform 2 cat ...

First-person-view (Observation)

Top-down-view (Not available)
Types of generalizations:
1) Unseen instructions

Types of generalizations:
2) Unseen/Longer sequences of instructions

Training set of tasks
- Go to A
- Go to B
- Go to A
- Pick up C
- Go to B
- Pick up B

Unseen tasks
- Go to A
- Go to B
- Pick up B
- Go to A
- Pick up C
- ...
Longer sequences of unseen instructions

Challenges

• Solving unseen instruction itself is a hard problem.
• Deciding when to move on to the next instruction.
  – The agent is not given which instruction to execute.
  – Should keep track of which instruction to solve.
  – Should detect when the current instruction is finished.
• Dealing with random events.
• Dealing with unbounded number of instructions.
• Delayed reward

Related works

• Hierarchical RL
  – Sutton, Precup, and Singh (1999); Dietterich (2000); Parr and Russell (1997); Bacon and Precup (2015); Kulkarni et al. (2016); etc.

• Task generalization
  – Schaul et al. (2015)

• Instruction execution
  – Tellex et al. (2011; 2014); MacMahon et al. (2006); Chen and Mooney (2011); Mei et al. (2015)
Architecture Overview

- **Subtask controller**: 1) execute primitive actions given a subtask and 2) predict whether the current subtask is finished or not.

Architecture Overview

- **Subtask controller**: 1) execute primitive actions given a subtask and 2) predict whether the current subtask is finished or not.
- **Meta controller**: set subtasks given a list of instructions

Subtask Space

• A subtask is decomposed into several arguments.
• This serves as a communication protocol between two controllers.

Architecture Overview

- **Subtask controller**: 1) execute primitive actions given a subtask and 2) predict whether the current subtask is finished or not.

Analogy Making Regularization

• Idea: learn a low-dimensional manifold that captures the correspondences between similar subtasks.
  – Visit A : Visit B :: Pick up A : Pick up B
  – Visit A : Visit C != Pick up A : Pick up B
Analogy Making Regularization

- Constraints

\[
\| \varphi(g_A) - \varphi(g_B) - \varphi(g_C) + \varphi(g_D) \| \approx 0
\]
\[
\| \varphi(g_A) - \varphi(g_B) - (g_C) + \varphi(g_D) \| \geq \tau_{dis}
\]
\[
\| \varphi(g_A) - \varphi(g_B) \| \geq \tau_{diff}
\]

if \( g_A : g_B :: g_C : g_D \)

if \( g_A : g_B \neq g_C : g_D \)

if \( g_A \neq g_B \)
Analogy Making Regularization

- Objective functions (Contrastive loss)

\[ \mathcal{L}_{sim} = \mathbb{E}(g_A, g_B, g_C, g_D) \sim g_{sim} \left[ \| \varphi(g_A) - \varphi(g_B) - (g_C) + \varphi(g_D) \|^2 \right] \]

\[ \mathcal{L}_{dis} = \mathbb{E}(g_A, g_B, g_C, g_D) \sim g_{dis} \left[ \max(0, \tau_{dis} - \| \varphi(g_A) - \varphi(g_B) - (g_C) + \varphi(g_D) \|)^2 \right] \]

\[ \mathcal{L}_{diff} = \mathbb{E}(g_A, g_B) \sim g_{diff} \left[ \max(0, \tau_{diff} - \| \varphi(g_A) - \varphi(g_B) \|)^2 \right] \]
Subtask Controller: Objective function

- **Objective function**
  - RL objective + Analogy making + Termination prediction objective
Experimental Setting

• **Observation**: 3d first person environment with randomly generated objects

• **Actions**: primitive actions
  – Move NSWE
  – Pick up NSWE
  – Transform NSWE
  – No operation

• **Subtasks**: “action” + “target object type”
  – **Visit X**: should be on top of object type X
  – **Pick up X**: perform “pick up” to object type X
  – **Transform X**: perform “transform” to object type X

• **Variations**:
  o **Interact** with X: interaction with objection can vary depending on target X
  o **Repeated actions**: e.g., Pick up 3 X’s

Only a subset of pairs of “action” + “object” is presented during training.
Subtask Controller: Result

• Analogy making is crucial for generalization

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Analogy</th>
<th>Train</th>
<th>Unseen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent</td>
<td>×</td>
<td>0.3 (99.8%)</td>
<td>-3.7 (34.8%)</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>0.3 (99.8%)</td>
<td>0.3 (99.5%)</td>
</tr>
<tr>
<td>Object-dependent</td>
<td>×</td>
<td>0.3 (99.7%)</td>
<td>-5.0 (2.2%)</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>0.3 (99.8%)</td>
<td>0.3 (99.7%)</td>
</tr>
<tr>
<td>Inter/Extrapolation</td>
<td>×</td>
<td>-0.7 (97.5%)</td>
<td>-2.2 (24.9%)</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>-0.7 (97.5%)</td>
<td>-1.7 (94.5%)</td>
</tr>
</tbody>
</table>

Table 1: Performance on parameterized tasks. Each entry shows ‘Average reward (Success rate)’. We assume an episode is successful only if the agent successfully finishes the task and its termination predictions are correct throughout the whole episode.
Demo Video on Parameterized Tasks
Meta Controller Architecture

- **Given**
  - Observation
  - Instructions
  - Subtask termination

- **Do**
  - Set subtask arguments

![Diagram showing the Meta Controller Architecture](image-url)
Meta Controller Architecture

- Retrieve one instruction from the list of instructions
- Choose subtask arguments
- The retrieved instruction and selected arguments are used as input for the next time-step (recurrence)
- Parameter prediction and analogy-making are applied.
Meta Controller: Instruction Memory

• Store all sentence embeddings into the **instruction memory**
• Maintain a pointer to a memory location
• Change the memory pointer through internal action: -1, 0, +1

• Independent of
  – The number of instructions
  – Compositions of instructions
Meta Controller: Differentiable Temporal Abstraction

- Temporal abstraction allows for infrequent updates for subtask controller

![Diagram of Meta Controller]

- **Observation**
- **CNN**
- **Retrieved instruction**
- **Subtask arguments**
- **Subtask termination?**

**Instructions** → **Instruction memory** → **Subtask Updater**

- **Context**
- **Update**

- **Yes** → **gt**
- **No** → **gt ← gt−1**
Updating the sub-tasks dynamically

Figure 5: Unrolled illustration of the meta controller with a learned time-scale. The internal states \((p, r, h)\) and the subtask \((g)\) are updated only when \(c = 1\). If \(c = 0\), the meta controller continues the previous subtask without updating its internal states.
Experimental Setting

- **Instructions**: “action” + “target object type”
  - **Visit X**: should be on top of object type X
  - **Pick up X**: perform “pick up” to object type X
  - **Transform X**: perform “transform” to object type X
  - **Pick up N X’s**: pick up N objects with type X
  - **Transform N X’s**: transform N objects with type X
- Only a subset of pairs of “action” + “object” is presented during training.
Experimental Setting

• **Reward**
  – Default reward: -0.1 (time penalty)
  – Visiting water: -0.3
  – Transforming an enemy: +0.9
  – Finishing all instructions: +1.0
    • No intermediate reward

• **Random event:**
  – a box randomly appears with probability of 0:03
  – transforming a box gives +0:9 reward
Evaluation of Meta Controller

- **Training set**: 4 seen instructions
- **Test set**: 20 seen or unseen instructions

<table>
<thead>
<tr>
<th>Length of instructions</th>
<th>Train 4</th>
<th>Test (Seen) 20</th>
<th>Test (Unseen) 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat</td>
<td>-7.1 (1%)</td>
<td>-63.6 (0%)</td>
<td>-62.0 (0%)</td>
</tr>
<tr>
<td>Hierarchical-Long</td>
<td>-5.8 (31%)</td>
<td>-59.2 (0%)</td>
<td>-59.2 (0%)</td>
</tr>
<tr>
<td>Hierarchical-Short</td>
<td>-3.3 (83%)</td>
<td>-53.4 (23%)</td>
<td>-53.6 (18%)</td>
</tr>
<tr>
<td>Hierarchical-Dynamic</td>
<td>-3.1 (95%)</td>
<td>-30.3 (75%)</td>
<td>-38.0 (56%)</td>
</tr>
</tbody>
</table>

**Baselines:**
- **Flat**: directly chooses primitive actions without using the parameterized skill.
- **Hierarchical-Long**: meta controller can update the subtask only when the current subtask is finished. Similar to (Kulkarni et al., 2016; Tessler et al., 2016).
- **Hierarchical-Short**: meta controller updates the subtask at every time-step.
Summary

• Deep Reinforcement Learning can benefit from building models, memory and hierarchy

• Forward prediction: can be useful for better exploration and possibly planning

• Memory: handle partial observability. Relevant to robotic agents.

• Hierarchical RL: Temporal abstraction and analogy making is beneficial for multi-task generalization and instruction execution