Model Based Reinforcement Learning

Oriol Vinyals (DeepMind) @OriolVinyalsML

May 2018
Stanford University
The Reinforcement Learning Paradigm

GOAL -> OBSERVATIONS

Agent

OBSERVATIONS

Environment

 ACTIONS

DeepMind
The Reinforcement Learning Paradigm

Maximize Return - long term reward:
\[ R_t = \sum_{t' \geq t} \gamma^{t'-t} r_{t'} = r_t + \gamma R_{t+1} \]
\[ \gamma \in [0,1] \]

With Policy - action distribution:
\[ \pi = P(a_t | x_t, ...) \]

Measure success with Value Function:
\[ V_{\pi}(x_t) = E_{\pi}(R_t) \]
A Classic Dilemma

“Old school” AI Researcher

Deep Learning Researcher

LEARN
ALL THE THINGS
A Classic Dilemma

\[ O \left( \sqrt{\frac{d}{n}} \right) \]

Model Complexity (VC Dimension)

Training set size
A Classic Dilemma

Model Based RL

Deep RL

LEARN

ALL THE THINGS
(Deep) Model Based RL

Deep Generative Model

Imagination Augmented Agents

Deep RL

Learning Model Based Planning from Scratch
Imagination Augmented Agents (NIPS17)

Joint work with:
We have good environment models ⇒ can we use them to solve tasks?

How do we do model-based RL and deal with imperfect simulators?

In this particular approach, we treat the generative model as an oracle of possible futures. ⇒ How do we interpret those ‘warnings’?
Imagination Augmented Agents (I2A)

a) Imagination core

Policy Net  Env. Model

$\hat{\pi}$  $\hat{a}_t$  EM

$O_t$  $\hat{O}_t$

$\hat{r}_{t+1}$

internal state

fixed input
Imagination Planning Networks (IPNs)

a) Imagination core

- Policy Net
- Env. Model
- \( \hat{\pi} \rightarrow \hat{a}_t \)
- \( \hat{O}_t \) or \( \hat{O}_t + 1 \)
- \( \hat{r}_t + 1 \)

b) Single imagination rollout

1. imagine future
- Imag. core
- \( \hat{O}_{t+2} \)
- \( \hat{r}_{t+2} \)

2. encode
- Imag. core
- \( \hat{O}_{t+1} \)
- \( \hat{r}_{t+1} \)
- Encoder

Rollout Encoding
Imagination Planning Networks (IPNs)

a) Imagination core
- Policy Net
- Env. Model
- Imag. core

b) Single imagination rollout
1. Imagine future
   - $\hat{o}_t$
   - $\hat{r}_{t+1}$
2. Encode
   - $\hat{o}_{t+1}$
   - $\hat{r}_{t+1}$

- Model-based path
- Model-free path

- Rollout encoder
- Aggregator

- $\pi, V$

- Internal state
- Fixed input
Sokoban environment

- Procedurally generated
- Irreversible decisions
Sokoban environment
What happens if our model is bad?
Mental retries with I2A
Mental retries with I2A
Mental retries with I2A

Solves 95% of levels!
Imagination efficiency

Imagination is expensive ⇒ can we limit the number of times we ask the agent to imagine a transition in order to solve a levels?

In other words, can we guide the search more efficiently than current methods?

| Method             | Value  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I2A @ 87</td>
<td>~ 1400</td>
</tr>
<tr>
<td>I2A MC search @ 95</td>
<td>~ 4000</td>
</tr>
<tr>
<td>MCTS @ 87</td>
<td>~ 25000</td>
</tr>
<tr>
<td>MCTS @ 95</td>
<td>~ 100000</td>
</tr>
<tr>
<td>Random search</td>
<td>~ millions</td>
</tr>
</tbody>
</table>

Table 1
One model, many tasks
Five events:

- Do nothing
- Eat a small pill
- Eat a power pill
- Eat a ghost
- Be eaten by a ghost

We assign to each event a different reward, and create five different games:

- ‘Regular’
- ‘Rush’ (eat big pills as fast as possible)
- ‘Hunt’ (eat ghosts, pills are ok i guess)
- ‘Ambush’ (eat ghosts, avoid everything else)
- ‘Avoid’ (everything hurts)
Results

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Standard model-free</th>
<th>Copy-model</th>
<th>I2A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>192</td>
<td>919</td>
<td>859</td>
</tr>
<tr>
<td>Avoid</td>
<td>-16</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>Hunt</td>
<td>-35</td>
<td>33</td>
<td>334</td>
</tr>
<tr>
<td>Ambush</td>
<td>-40</td>
<td>-30</td>
<td>294</td>
</tr>
<tr>
<td>Rush</td>
<td>1.3</td>
<td>178</td>
<td>214</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boxes</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2A (%)</td>
<td>99.5</td>
<td>97</td>
<td>92</td>
<td>87</td>
<td>77</td>
<td>66</td>
<td>53</td>
</tr>
<tr>
<td>Standard (%)</td>
<td>97</td>
<td>87</td>
<td>72</td>
<td>60</td>
<td>47</td>
<td>32</td>
<td>23</td>
</tr>
</tbody>
</table>
Learning model-based planning from scratch

Joint work with:
Prior work: Spaceship Task v1.0

Hamrick, Ballard, Pascanu, Vinyals, Heess, Battaglia (2017)

- Propel spaceship to home planet (white) by choosing thruster force and magnitude
- Other planets’ (grey) gravitational fields influence the trajectory
- Continuous, context bandit problem
Prior work: Imagination-based metacontroller

- Restricted to bandit problems
This paper:

Imagination-based Planner (IBP)
Spaceship Task v2.0: Multiple actions

- Use thruster multiple times
- Increase difficult than Spaceship Task v1.0:
  1. Pay for fuel
  2. Multiplicative control noise
- Opens up new strategies, such as:
  1. Move away from challenging gravity wells
  2. Apply thruster toward target
Spaceship Task v2.0: Multiple actions

- Use thruster multiple times
- Increase difficult than Spaceship Task v1.0:
  1. Pay for fuel
  2. Multiplicative control noise
- Opens up new strategies, such as:
  1. Move away from challenging gravity wells
  2. Apply thruster toward target
Spaceship Task v2.0: Multiple actions

- Use thruster multiple times
- Increase difficult than Spaceship Task v1.0:
  1. Pay for fuel
  2. Multiplicative control noise
- Opens up new strategies, such as:
  1. Move away from challenging gravity wells
  2. Apply thruster toward target
Spaceship Task v2.0: Multiple actions

- Use thruster multiple times
- Increase difficult than Spaceship Task v1.0:
  1. Pay for fuel
  2. Multiplicative control noise
- Opens up new strategies, such as:
  1. Move away from challenging gravity wells
  2. Apply thruster toward target
Spaceship Task v2.0: Multiple actions

- Use thruster multiple times
- Increase difficult than Spaceship Task v1.0:
  1. Pay for fuel
  2. Multiplicative control noise
- Opens up new strategies, such as:
  1. Move away from challenging gravity wells
  2. Apply thruster toward target
Imagination-based Planner

- Imagination can be:
  - *Current step only*: imagine only from the current state

**Simple Breath First planner**

- step world
- imagination

Root State
Imagination-based Planner

- Imagination can be:
  - *Current step only*: imagine only from the current state
  - *Chained steps only*: imagine a sequence of actions

Simple Depth First planner

- step world
- imagination

Root State
Imagination-based Planner

- Imagination can be:
  - *Current step only*: imagine only from the current state
  - *Chained steps only*: imagine a sequence of actions
  - *Imagination tree*: manager chooses whether to use current (root) state, or chain imagined states together

### Simple Imagination tree

- **step world**
- **imagination**

Root State
Imagination-based Planner

Manager

Imagine

Controller

Imagination

Memory

Act

Controller

World

Memory

Imagination tree

1-step

n-step

k = 0

k = 1

k = 2
Imagination-based Planner
Real trials: 3 actions

More complex plans:
1. Moves away from complex gravity
2. Slows its velocity
3. Moves to target
Different strategies for exploration

1 step  n step  Imagination trees
Results
Results

A. Performance v imagination steps
   - 1 action
   - 2 action
   - 3 action

B. Imagination use v resource cost

C. Performance v resource cost
   - Task loss
   - Total cost

D. Imagination strategy performance
   - Fuel cost 0.0002
     - 1-step
     - n-step
     - Tree

E. Fuel cost 0.0004
   - Maximum imaginations per action
   - Task loss (log scale)
Figure 6: **Top row:** Single maze, multiple goals, P represents the position of the player, and G is the goal. The first three configurations were from training, and the last tested the agent’s ability to generalize. The imagination steps are the shaded blue areas, for different imagination budgets 5, 9, 13 and 15 from left to right. The imagination depth corresponds to the color saturation of the grid cells. The actual action the agent took is shown as a solid black arrow. **Bottom row:** Multiple mazes and one example run. Left two: two example mazes, one with more candidate goal locations than the other. Middle and right four: one example run of a learned agent on one maze. In this run, the agent performed four imagination steps, then took its first action, then performed one more imagination step. Both the imagination steps and the rollout updates to c are shown. In the rollout updates, the original Q + c values before the rollout are colored gray, and the updates are blue.
Figure 5: Learned imagination trees for the maze task. See supplementary material for more details.
Imagination-based Planner

How does it work? (learnable components are **bold**)

1. On each step, inputs:
   - State, \( s_t \): the planet and ship positions, etc.
   - Imagined state, \( s'_t \): internal state belief
   - History, \( h_t \): summary of planning steps so far
Imagination-based Planner

How does it work? (learnable components are **bold**)

1. On each step, inputs:
   - State, \( s_t \): the planet and ship positions, etc.
   - Imagined state, \( s'_t \): internal state belief
   - History, \( h_t \): summary of planning steps so far

2. **Controller** policy returns action, \( a_t \)

3. **Manager** routes actions to world or imagination, \( r_t \)
Imagination-based Planner

How does it work? (learnable components are **bold**)

1. On each step, inputs:
   - State, $s_t$: the planet and ship positions, etc.
   - Imagined state, $s'_t$: internal state belief
   - History, $h_t$: summary of planning steps so far

2. **Controller** policy returns action, $a_t$

3. **Manager** routes actions to world or imagination, $r_t$

4. If route, $r_t$, indicates:
   a. “**Imagination**”, predicts imagined state, $s'_{t+1}$
Imagination-based Planner

How does it work? (learnable components are bold)

1. On each step, inputs:
   - State, $s_t$: the planet and ship positions, etc.
   - Imagined state, $s'_t$: internal state belief
   - History, $h_t$: summary of planning steps so far

2. **Controller** policy returns action, $a_t$

3. **Manager** routes actions to world or imagination, $r_t$

4. If route, $r_t$, indicates:
   a. “Imagination”, predicts imagined state, $s'_{t+1}$
   b. “World”, model predicts new state, $s_{t+1}$
Imagination-based Planner

How does it work? (learnable components are bold)

1. On each step, inputs:
   - State, $s_t$: the planet and ship positions, etc.
   - Imagined state, $s'_t$: internal state belief
   - History, $h_t$: summary of planning steps so far

2. Controller policy returns action, $a_t$

3. Manager routes actions to world or imagination, $r_t$

4. If route, $r_t$, indicates:
   a. “Imagination”, predicts imagined state, $s'_{t+1}$
   b. “World”, model predicts new state, $s_{t+1}$

5. Memory aggregates new info into updated history, $h_{t+1}$
Imagination-based Planner

How is it trained?
Three distinct, concurrent, on-policy training loops
Imagination-based Planner

How is it trained?

Three distinct, concurrent, on-policy training loops

1. **Model/Imagination** (interaction network)
   
   Supervised: $s_t, a_t \rightarrow s_{t+1}$
Imagination-based Planner

How is it trained?

Three distinct, concurrent, on-policy training loops

1. **Model/Imagination** (interaction network)
   
   Supervised: \( s_t, a_t \rightarrow s_{t+1} \)

2. **Controller/Memory** (MLP/LSTM)
   
   SVG: Reward, \( u_t \), is assumed to be \( |s_{t+1} - s^*|^2 \). Model, imagination, memory, and controller are differentiable. Manager’s discrete \( r_t \) choices are assumed to be constants.
Imagination-based Planner

How is it trained?

Three distinct, concurrent, on-policy training loops

1. **Model/Imagination** (interaction network)
   
   Supervised: $s_t, a_t \rightarrow s_{t+1}$

2. **Controller/Memory** (MLP/LSTM)
   
   SVG: Reward, $u_t$, is assumed to be $|s_{t+1} - s^*|_2$. Model, imagination, memory, and controller are differentiable. Manager’s discrete $r_t$ choices are assumed to be constants.

3. **Manager**: finite horizon MDP (MLP q-net, stochastic)

   REINFORCE: Return = (reward + comp. costs), $(u_t + c_t)$
Bonu Paper: MCTMnet

Joint work with:
Arthur Guez*, Theo Weber*, Ioannis Antonoglou, Karen Simonyan,
Daan Wierstra, Remi Munos, David Silver
Vanilla MCTS: a single simulation (tree-policy phase)

Q, \{N\}  
UCB-type rule.  
*Fixed* function of Q, visits, (prior net)

Tree after some sims
MCTS: a single simulation (tree-policy phase)

(Using true model for each transition)
MCTS: a single simulation

Value network (pretrained)
MCTS: a single simulation (backup phase)
MCTS: a single simulation (backup phase)

\[ Q \leftarrow V, N_{a+1} \]

**MCTS output:** After many simulations, take \( \max Q \) (or \( \max N \)) at the root node
MCTSnet model: a single simulation (tree-policy phase)

- Root embedding
- Tree after some sims
- Simulation policy network
MCTSnet model: a single simulation (tree-policy phase)

(Using true model for each transition)
MCTSnet model: a single simulation (tree-policy phase)
MCTSnets model: a single simulation (backup phase)

Note: reward and action should also be provided as input to bnet
MCTSnset model: a single simulation (backup phase)
Embeddings represent a tree-shaped memory of past rollouts

Simulation that expanded node $x$ (first time embedNet is called)

Later, another simulation visits the same node $x$ (and expands a node $y$)

Later, another simulation visits the same node $x$ (and expands a node $y$)
Multiple simulations / search

A single forward of the MCTSnet:
Backup along traversed path

Simulation down
The tree

Evaluate/embed new tree node

MCTSnet architecture (cartoon)
Recap of MCTSnet modules

- **Embed network**
- **Backup network**
- **Simulation policy network**
- **Readout network**

$\text{logits}$
Problem setting: classification

Data:

Input: \( x \) - Sokoban frame

Target: \( a^* \) - “oracle” action (obtained from running long MCTS+vnet+TT search)
Loss

Classification loss (predict the oracle action in each state $x$):

$$l(x) = E_z(x) [- \log p_\theta(a^* | x, z)]$$

Gradient of the loss splits into **differentiable** and **non-differentiable** parts.

$$\nabla_\theta l(x) = -E_z \left[ \nabla_\theta \log p_\theta(a^* | x, z) + \left( \sum_i \nabla_\theta \log \pi_s(a_i | H_i) \right) \log p_\theta(a^* | x, z) \right]$$

- **Straight-through BP**
- **REINFORCE trained**
Thanks!!

@OriolVinyalsML
May 2018