Jeff Clune

Senior Research Manager (Staff Scientist)

Harris Associate Professor, Computer Science

@jeffclune
Go-Explore
A new type of algorithm for hard-exploration problems

Adrien Ecoffet  Joost Huizinga  Joel Lehman  Ken Stanley*  Jeff Clune*

*Co-senior authors
Grand Challenge in RL: Effective Exploration

• Hard-exploration problems
  • Sparse-reward problems
    • rare feedback
    • Montezuma’s Revenge
      • has become mini-grand challenge
“Nevertheless, games demanding more temporally extended planning strategies still constitute a major challenge for all existing agents including DQN (for example, Montezuma’s Revenge).”

Mnih et al. 2015
Grand Challenge in RL: Effective Exploration

• Hard-exploration problems
  • Sparse-reward problems
    • rare feedback
    • Montezuma’s Revenge
      • has become mini-grand challenge

• Deceptive problems
  • wrong feedback! (locally)
  • local optima

Reward = distance to goal

Lehman & Stanley
Grand Challenge in RL: Effective Exploration

• Hard-exploration problems

• Sparse-reward problems
  • rare feedback
  • Montezuma’s Revenge
    • has become mini-grand challenge

• Deceptive problems
  • wrong feedback! (locally)
  • local optima

Pitfall
Prior Results

• Classic Deep RL algorithms:
  • Montezuma’s Revenge: 0 - ~2500
  • Pitfall: 0 or negative
Prior Results

• Classic Deep RL algorithms:
  • Montezuma’s Revenge: 0 - ~2500
  • Pitfall: 0 or negative
• UCT/MCTS + Game Emulator (deterministic)
  • Montezuma’s Revenge: 0
    • Bellemare et al. 2012
Intrinsic Motivation

- reward reaching new states / reducing uncertainty
- lots of variants
  - pseudo-counts (Bellemare et al. 2016)
  - #exploration (Tang et al. 2016)
  - RND (Burda et al. 2018)
  - Deep Curiosity Search (Stanton & Clune 2018, this workshop: next poster session)
  - Prediction (e.g. Pathak et al. 2017)
  - Bootstrapped DQN (Osband et al. 2016)
  - many more…
Intrinsic Motivation

• **SOTA: ~11,500**
  - Random Network Distillation, Burda et al. 2018

• **Solved level one in 1/10 runs**
  - big advance!

Burda et al. 2018
Intrinsic Motivation

• Helps, but why doesn’t it work better?
“Detachment”

1. Intrinsic reward (green) is distributed throughout the environment
2. An IM algorithm might start by exploring (purple) a nearby area with intrinsic reward
3. By chance, it may explore another equally profitable area
4. Exploration fails to rediscover promising areas it has detached from
Detachment

• Replay buffers
  • Should remember in theory, but forget/fail in practice
    • replay buffer size must be very large
    • but that causes optimization/stability issues
Detachment

- Proposal: **explicitly remember**
  - where promising locations are
  - how to get back to them
“Derailment”

- Most RL algorithms:
  - take promising policy, perturb it, hope it explores further
  - most likely breaks policy!
    - especially as length, complexity, & precision of sequence increases
Derailment

- Insight: First return, then explore
Derailment

• Insight: First return, then explore
• counter: hurt robustness?
Go-Explore Strategy

- Phase 1: First solve
- Phase 2: Then robustify
  - pay the cost to robustify only once you know what needs to be robustified
Go-Explore: Phase 1

- initialization:
  - take random actions, store cells visited
Go-Explore: Phase 1

• Phase 1: explore until solved
  A. choose a cell from archive
  B. Go back to it
  C. Explore from it
  D. add newly found cells to archive
    • if better, replace old way of reaching cell
Avoids Detachment
By Remembering Promising Exploration Stepping Stones

• Intrinsic motivation:
  • narrow beam mining intrinsic motivation and moving on

• Go-Explore
  • continuously expands sphere of knowledge
Cell Representations

- For large state spaces (e.g. Atari), need conflation
  - similar states map to same cell
  - interestingly different states map to different cells
Cell Representations

- First attempt: downsampling images
  - dumb
  - fast
  - no game-specific domain knowledge
Choosing Cells

• Prefer cells that are
  • led to new states (productive)
  • less tested (newer)
Returning to Cells

• resettable & deterministic: reset state (or replay actions)
• stochastic environment:
  • goal-conditioned policy
    • e.g. UVFA (Schaul et al. 2015), HER (Andrychowicz et al. 2017)
    • other ideas
Returning to Cells

- save action-sequence trajectories to cells
  - open loop
  - no neural network!
Exploration

• after returning to a cell
• take random actions (k=100)
Montezuma’s Revenge Results: Phase 1

- Solves level 1 65% of runs
Go-Explore
Separates learning a solution into two phases

Phase 1: Explore Until Solved
- Select state from archive
- Go to state
- Explore from state
- Update archive

Phase 2: Robustify (if necessary)
- Run imitation learning on best trajectory

no neural networks (in current work)
produces neural network robust to stochasticity
Phase 2: Robustify

• Imitation learning can work well with human demonstrations
  • including on Montezuma’s Revenge & Pitfall (e.g. Aytar 2018)

• Go-Explore Phase 1 generates solution demonstrations
  • automatically
  • quickly
  • cheaply
  • as many as you want
Phase 2: Robustify

• Most imitation learning algorithms should work
• We chose the “backward algorithm”
  • from Salimans & Chen 2018 (this workshop!)
  • similar: Backplay from Resnick et al. 2018
Backwards Imitation Learning

Starting with demonstration (solution trajectory)

1. Start at the end
2. Backup k steps
3. Run RL until $\geq$ original score
4. go to 1
Robustifies Due To Added Noise

- no-ops
- action sampling
Improves Over Demonstration

- Can discover
  - higher scores
  - more efficient paths due to discount factor
Many Demonstrations

• Backwards Imitation somewhat unreliable from one demonstration
• We modified it to learn from many
  • here, 4
• Add no-ops at beginning
• Success rate: 100%

Example Successful Attempt
Results with Robust Deep Neural Networks

- 3x previous state of the art!
- Robust to stochasticity
  - random # no-ops up to 30
- No game-specific domain knowledge
Go-Explore created a healthy debate

- When should we require stochasticity?
  - test time (only)?
  - training time too?
Stochasticity at **Test Time**

- Classic way: random number up to 30 no-ops
Stochasticity at **Test** Time

- New proposal: “sticky actions” Machado et al. 2017
  - repeat last action with probability $p$
  - we were not aware to what extent there was support for this proposal
  - added them during Phase 2: Robustification

- **Avg. score with sticky actions**: 33,836 ~3x state of the art
  - new data (might change a bit)
Stochasticity at Training Time

Two types of problems

1. We want a robust solution, do not care how it is obtained
   - robotics
Stochasticity at **Training Time**

Two types of problems

1. We want a robust solution, do not care how it is obtained
   - robotics
   - realistic behaviors for agents/characters in video games, simulations
Stochasticity at Training Time

Two types of problems

1. We want a robust solution, do not care how it is obtained
   - robotics
   - realistic behaviors for agents/characters in video games, simulations
   - control of complex industrial processes
Stochasticity at Training Time

Two types of problems

1. We want a robust solution, do not care how it is obtained
   • robotics
   • realistic behaviors for agents/characters in video games, simulations
   • control of complex industrial processes
   • many more (theorem proving, protein folding, etc.)

Determinism-during-training is OK
Stochasticity at Training Time

• Most RL applications will require simulator
  • sample inefficient
  • not safety aware
• Might as well take advantage of it fully
Stochasticity at Training Time

Two types of problems
1. We want a robust solution, do not care how it is obtained
2. We require training in the face of noise
   • will have to learn in the real world
   • understanding biological learning
Proposal: Two Versions of Atari Benchmarks

1. Stochasticity at test time only
   • all our current scores and claims

2. Stochasticity during training and testing
   • have not tried yet
   • we think Go-Explore will ultimately work here too (with research)
Adding Domain Knowledge

• Ideal algorithms (my opinion)
  • work without domain knowledge
  • benefit from domain knowledge when provided
    • especially when easily provided
Adding Domain Knowledge

• Go-Explore can add it via cell representation
  • Important notes:
    • final post-robustification policies still play from pixels only
      • do not consume domain knowledge at eval time
    • wrote simple code to extract info from pixels (not emulator)
  • Montezuma’s Revenge
    • unique combinations of: x, y location, room, level, numKeysHeld
  • Pitfall
    • <x, y> location, room
Montezuma’s Results with Domain Knowledge

Phase 1: Exploration (deterministic eval)

- Solves 9 levels on average
- in half the time
- Solves all 3 unique levels
- Levels 3+ nearly identical
  - slight timing differences
  - score changes state/inputs
- On average
  - scores over 469,209!
  - solves 29 levels!
- Sticky action eval:
  - scores 281,264
  - level 18

Results with Robust Deep Neural Networks

![Graph showing results with Robust Deep Neural Networks. The graph compares scores over time, with Go-Explore (domain knowledge) outperforming other methods.]
Results with Robust Deep Neural Networks

- Increased Gym’s time limits
- Best neural network
  - scores over 4 million
  - reaches level 295
- Beats human world record
  - 1,219,200
  - achieves strictest definition of “super-human”
Results with Robust Deep Neural Networks

- Even beats previous work from human demonstrations
  - better demos
  - more demos
Not Expensive!

- Solving Level 1 of Montezuma’s Revenge
  - Phase 1: Exploration
    - ~1 hour!
    - single machine (22 CPUs, 50GB RAM)
    - runs produce ~4GB of data
  - Phase 2: Robustification
    - ~1 day
      - 16 GPUS

- All told: ~1 day on modest hardware
  - compare to billions of frames, thousands of workers
Pitfall

- no prior algorithm scores > 0
- without:
  - fully deterministic test environment
  - or human demonstration
Pitfall Results: Without Domain Knowledge

• Reaches 22 rooms
• No positive reward
• Reason: over-conflates
  • more research required

Results: Phase 1 Exploration With Domain Knowledge

Deterministic Eval
Results with Robust Deep Neural Networks

- avg over 21,000
  - 14,592 with sticky actions
- advances state of the art substantially
Future (Current) Directions
Each piece/option is simple, demonstrating the value of the decomposition/strategy.

Phase 1: Explore Until Solved

1. Select state from archive
2. Go to state
3. Explore from state
4. Update archive

Phase 2: Robustify (if necessary)

- Run imitation learning on best trajectory

No neural networks (in current work)

Produces neural network robust to stochasticity
random exploration → intelligent exploration

- learns and reuses useful skills for exploration
  - e.g. walking, jumping, climbing over fences
- key to scaling to very hard problems
Learned Representations

• instead of random or engineered cell representations
• many options
  • auto-encoders
  • prediction (e.g. Pathak et al. 2017)
  • auxiliary tasks (e.g. Jaderberg 2016)
  • many more…
Stochastic Training

- Goal-conditioned policies
  - UVFA (Schaul et al. 205), HER (Andrychowicz et al. 2017), etc.
- Still benefits from Go-Explore insights
  - explicitly remember stepping stones, first return / then explore, etc.
Robotics

- Solve hard problems in simulation
  - “Robot, fetch me a beer”

- Solve in deterministic simulator
- Robustify in stochastic simulator
- Transfer to reality
- Learn in reality (optional)

Intelligent trial & error
Cully, Tarapore, Mouret, & Clune

Transferability approach
Koos, Mouret, Doncieux

Domain randomization
Tobin et al. 2017
Also Try With Different

- archives
- heuristics for which cells to return to
- imitation learning algorithms
  - GAIL (Ho et al. 2016)
  - Deep Mimic (Peng et al. 2018)
  - DQfD (Hester et al. 2017)
  - etc.
Similarities to Previous Algorithms
Remember good exploration stepping stones (avoids detachment)
Quality Diversity Algorithms

• seek diverse, high-quality solutions
  • Novelty Search + Local Competition (Lehman & Stanley)
  • MAP-Elites (Mouret & Clune)
Robots that adapt like animals
Nature, 2015

Back on its feet
Using an intelligent trial-and-error learning algorithm this robot adapts to injury in minutes

Pages 426 & 503

Damage occurs (leg loses power)
Robots that adapt like animals
Nature, 2015

Back on its feet
Using an intelligent trial-and-error learning algorithm this robot adapts to injury in minutes
PAGES 426 & 503

Antoine Cully
UPMC Université France

Jeff Clune
University of Wyoming USA

Danesh Tarapore
UPMC Université France

Jean-Baptiste Mouret
UPMC Université France

Behavior-performance Map
Forward Speed (m/s)
0.24
Trajectory
Similar to graph search? (e.g. breadth-first search)

• Yes, but
1. Go-Explore adds Phase 2: Robustification to handle noise
2. Such algorithms are intractable in raw high-D state space
Similar to graph search? (e.g. breadth-first search)

• Yes, but
  1. Go-Explore adds Phase 2: Robustification to handle noise
  2. Such algorithms are intractable in raw high-D state space

  • Go-Explore insight: run these great graph search algorithms in low-D conflated spaces!
    • adds many challenges
      • which cells can you reach from current cell? how (have to search)? can’t replace subpaths. etc.
    • requires many algorithmic innovations
    • new research area: porting classic graph algorithms to high-D (conflated representations): BFS, DFS, Dijkstra’s, A-Star, etc.
Similar to MCTS? UCT?

• Somewhat, but
• MCTS
  • does not reward/seek novel states (despite $1/\sqrt{n}$)
  • no conflation (except provably equivalent types, like rotation)
  • does not perform well on these games (Guo 2014, Bellemare 2012)
Guided Policy Search
Levine et al. 2016

- also first solves, then robustifies (so does Guo 2014)
- GPS requires
  - non-deceptive, non-sparse, differentiable loss function to find solutions
    - cannot handle discrete, sparse, and/or deceptive rewards (Atari, most of real-world)
  - differentiable model of the world or learning a set of local models
    - requires fully observable state during training
Go-Explore Conclusions

Phase 1: Explore Until Solved
- Select state from archive
- Go to state
- Explore from state
- Update archive

Phase 2: Robustify
- Run imitation learning on best trajectory

- new type of algorithm for hard-exploration problems
- opens many new exciting research directions
  - using Go-Explore
    - already mentioned many
  - hybridizing insights from Go-Explore with other RL algorithms
    - first return, then explore
    - first solve, then robustify
    - we are excited to see what the community comes up with
Go-Explore
A new type of algorithm for hard-exploration problems

Thanks for discussions:
Peter Dayan, Shimon Whiteson,
Zoubin Ghahramani, Juergen Schmidhuber,
Yura Burda, Harri Edwards, ML community (reddit/Twitter)

*Co-senior authors