Introduction to Deep RL, Part 2

Joshua Achiam

OpenAI

February 2, 2019

2.1: What is RL currently achieving?

If you have infinite simulator data and well-defined rewards, you can make substantial progress on extremely hard problems!

- Atari
- Simulated robotics
- Go (Deepmind's AlphaZero)
- Dota (OpenAI Five)
- Starcraft (Deepmind's AlphaStar)

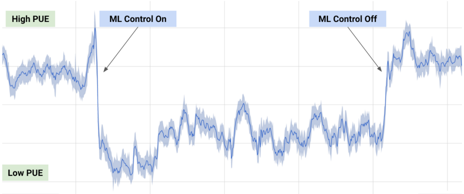
Hard to overstate what an unbelievable accomplishment this was

Previously: Go was considered unassailable stronghold for human experts, AI 10+ years away

What RL Can Currently Do: RL in the Real World

RL is beginning to see profitable real-world applications!

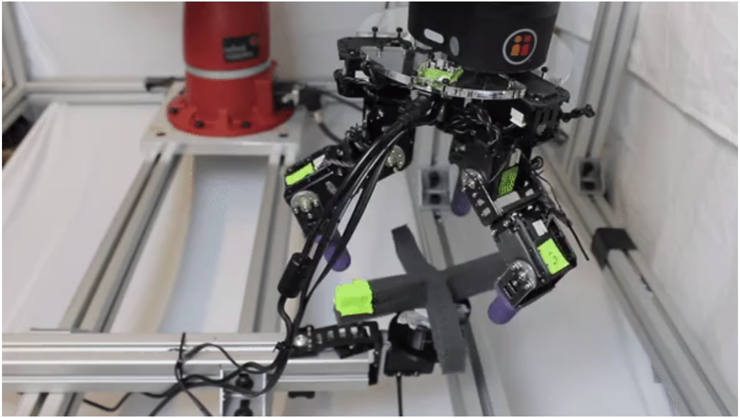
- Facebook uses RL (DQN) for push notifications (Horizon)
- DeepMind integrated RL into data center cooling
- Several promising early efforts at applying RL for robotics



2.1.1: Spotlight on RL for Real-World Robotics

Tasks that can't easily be simulated

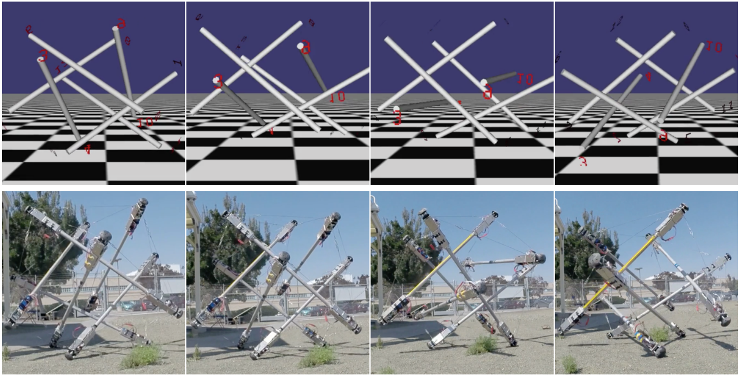
Zhu et al. trained low-cost robots from scratch in the real world on hard-to-simulate tasks using natural policy gradient. (Note: observation space here is hand state and valve state)



[1] Zhu et al, 2018: "Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost"

Robots that are hard to control conventionally

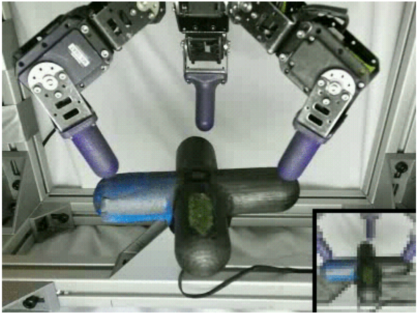
Zhang et al. trained a tensegrity robot with deep RL in simulation and demonstrated transfer to the real world



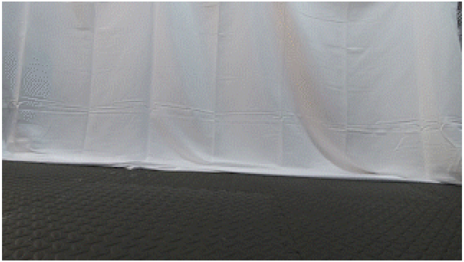
[1]Zhang et al, 2016: "Deep Reinforcement Learning for Tensegrity Robot Locomotion"

Soft Actor-Critic

Haarnoja et al. trained robots in the real world with Soft Actor-Critic, and demonstrated efficient and robust control on hard domains



Manipulation from visual input, agent sees lower-right (took 20 hours to learn)



Robust control of a legged robot (took 2 hours to learn)

[1] Haarnoja et al, 2018: "Soft Actor-Critic Algorithms and Applications"

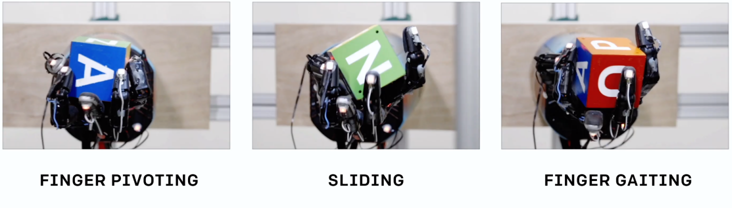
Anymal

Hwangbo et al. used real data to learn a better simulator, and then used lots of simulator data to train complex locomotion and recovery policies with TRPO for the Anymal robot:

[1]Hwangbo et al, 2018: "Learning agile and dynamic motor skills for legged robots"

Learning Dexterity

OpenAI et al. trained policies with PPO to dextrously manipulate a complex hand robot, using sim2real by domain randomization:



[1]OpenAI et al, 2018: "Learning Dexterous In-Hand Manipulation"

2.2: What are the challenges in modern RL?

Some grand challenges for RL

Let's talk about where RL fails, what we need that we aren't getting, and what research is happening.

- **Sample efficiency**: Modern deep RL algorithms are very inefficient with data, requiring in some cases 100x or more experience than a human to achieve good performance. How can we make them faster?

Some grand challenges for RL

Let's talk about where RL fails, what we need that we aren't getting, and what research is happening.

- **Sample efficiency**: Modern deep RL algorithms are very inefficient with data, requiring in some cases 100x or more experience than a human to achieve good performance. How can we make them faster?
- **Exploration**: Modern RL is terrible at environments where rewards are very rare—how can we get agents to try out new things?

Some grand challenges for RL

Let's talk about where RL fails, what we need that we aren't getting, and what research is happening.

- **Sample efficiency**: Modern deep RL algorithms are very inefficient with data, requiring in some cases 100x or more experience than a human to achieve good performance. How can we make them faster?
- **Exploration**: Modern RL is terrible at environments where rewards are very rare—how can we get agents to try out new things?
- **Generalization**: Policies trained for one task are brittle and cannot be used outside of training environment: how can we make them generalize?

Some grand challenges for RL

Let's talk about where RL fails, what we need that we aren't getting, and what research is happening.

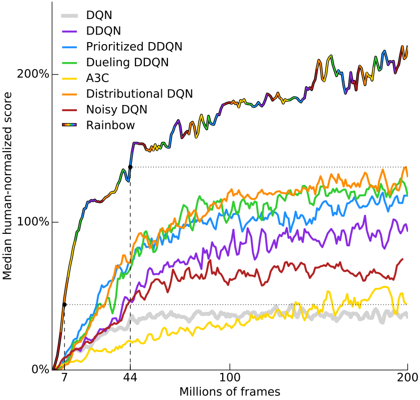
- **Sample efficiency**: Modern deep RL algorithms are very inefficient with data, requiring in some cases 100x or more experience than a human to achieve good performance. How can we make them faster?
- **Exploration**: Modern RL is terrible at environments where rewards are very rare—how can we get agents to try out new things?
- **Generalization**: Policies trained for one task are brittle and cannot be used outside of training environment: how can we make them generalize?
- **Safety**: How can we make sure that agents trained by deep RL behave in ways consistent with human preferences? (I'll let Dario cover this in his talk, but it's a critical challenge and I would be remiss not to mention it.)

2.2.1: Research on Improving Sample Efficiency

Combining Model-Free Improvements

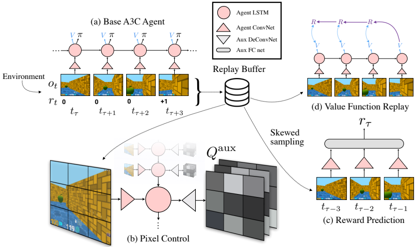
Rainbow DQN: combines...

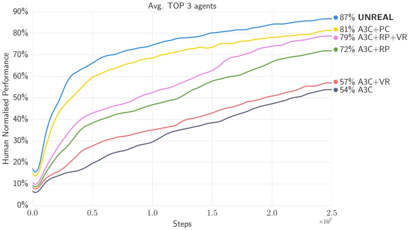
- Dueling architecture
- Double Q-Learning
- Prioritized experience replay
- N-step Q-Learning
- Distributional Q-Learning
- Parameter-space noise



[1] Hessel et al, 2017: “Rainbow: Combining Improvements in Deep Reinforcement Learning”

UNREAL uses various unsupervised auxilliary tasks to speed up learning:

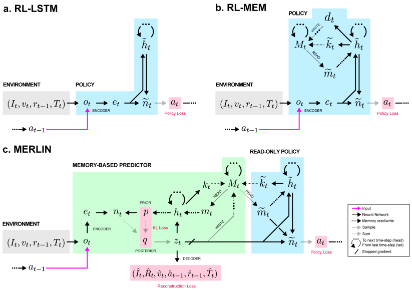


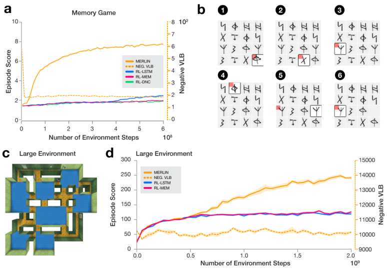


[1] Jaderberg et al, 2016: "Reinforcement Learning with Unsupervised Auxiliary Tasks"

Combining with Memory Mechanisms

MERLIN combines unsupervised learning and attention-based memory to improve over baseline architectures:

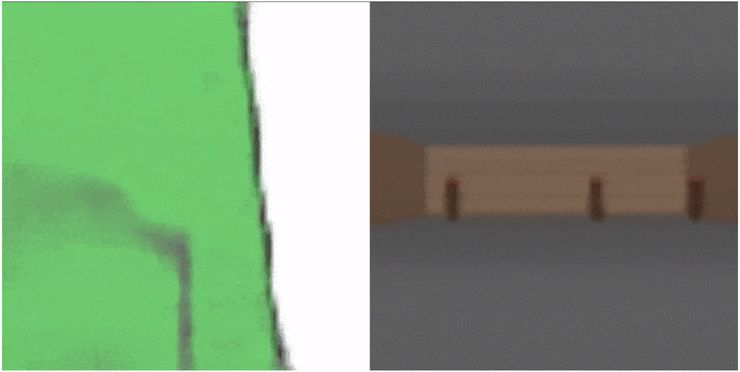




Note: this avenue may not help on arbitrary problems, but seems likely to help on sophisticated partially-observed ones

[1] Wayne et al, 2018: "Unsupervised Predictive Memory in a Goal-Directed Agent"

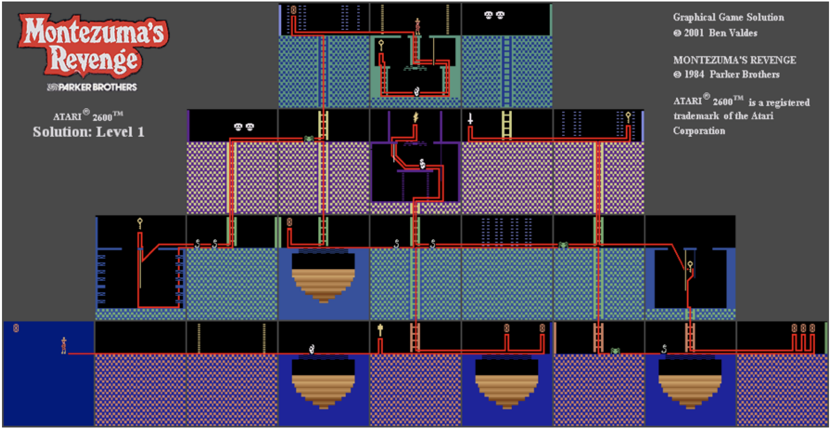
Techniques like **World Models**, that allow an agent to learn from simulated experience, are another way to bring down the needed number of real samples:



[1]Ha and Schmidhuber, 2018: "World Models"

2.2.2: Research on Improving Exploration

AKA, the Quest to Solve Montezuma's Revenge



- Famously hard Atari game
- Most RL algorithms do terribly
- Very sparse rewards, many actions lead to death
- But humans can do just fine!

Various **intrinsic motivation** schemes exist, which work by adding a non-environment-dependent reward:

$$R(s,a,s') \rightarrow R(s,a,s') + \eta R_{int}(s,a,s')$$

Usually, intrinsic reward prefers **new experiences**. Examples:

- Pseudocount-based methods[1]: approximates number of state visitations with density model, rewards are:

$$R_{int}(s,a,s') \propto \frac{1}{\sqrt{N(s)}}$$

- Information gain[2]: approximates change in info state about environment model based on new experience

$$R_{int}(s,a,s') \approx D_{KL}(P(\mathcal{E}|\mathcal{D}_t, s_{t+1})||P(\mathcal{E}|\mathcal{D}_t))$$

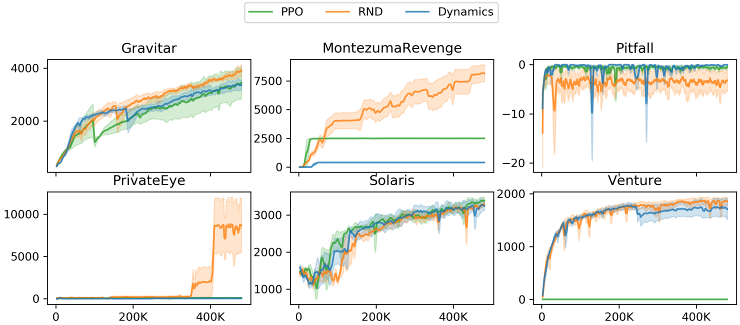
[1] Bellemare et al, 2016: "Unifying Count-Based Exploration and Intrinsic Motivation"
[2] Houthooft et al, 2016: "Variational Information Maximizing Exploration"

Random Network Distillation[3]: A new and appealingly simple form of intrinsic motivation:

$$R_{int}(s) = \|f_\phi(s) - f_\xi(s)\|,$$

where f_ϕ is a random neural network (the target) and f_ξ is trained to match f_ϕ

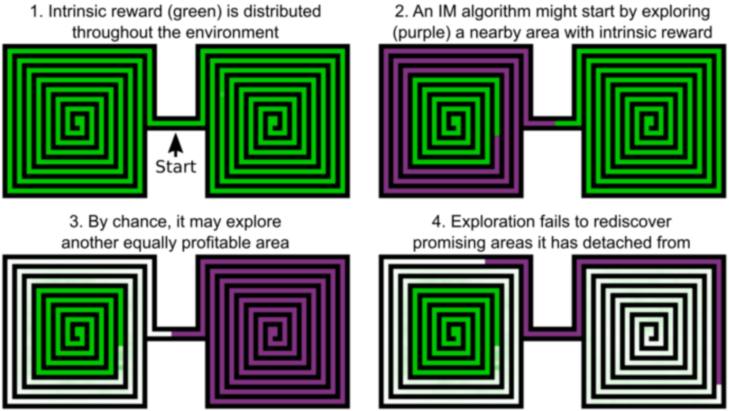
- Exploits generalization properties of networks: f_ϕ matches f_ξ on old states but not new states!



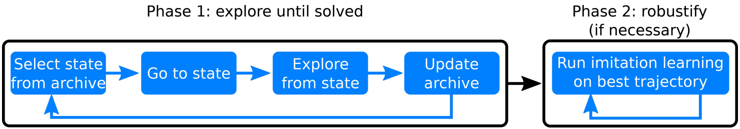
[3] Burda et al, 2018: "Exploration by Random Network Distillation"

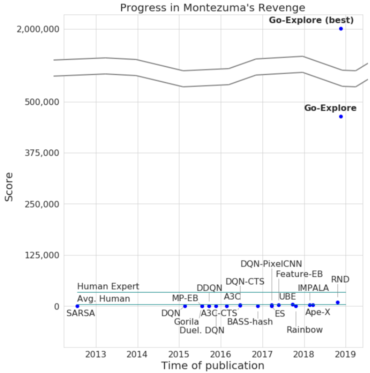
Go-Explore

Ecoffet et al, 2018 identify a problem with intrinsic motivation that they call **detachment**:



They rectify detachment with **Go-Explore**[4] algorithm:



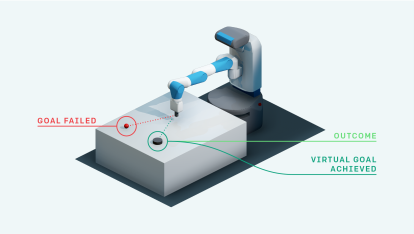


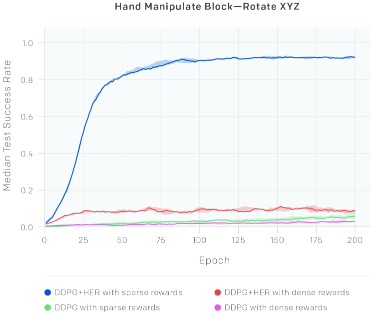
[4] Ecoffet et al, 2018: "Go-Explore: a New Approach for Hard-Exploration Problems"

2.2.3: Research on Generalization

Simple idea behind **HER**[5]:

- If you are trying to learn to go to a goal, goals are hard to get to.
- Unless you pretend that the place you ended up was where you meant to go: then you get to goals all the time!

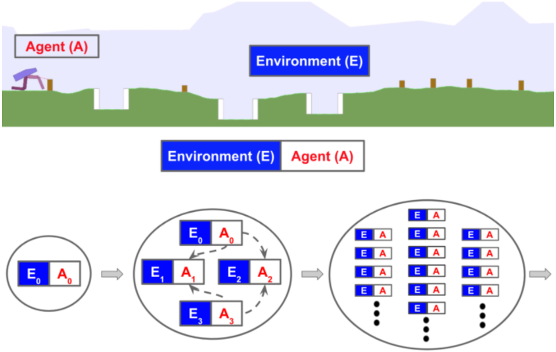




[5] Andrychowicz et al, 2017: "Hindsight Experience Replay"

POET

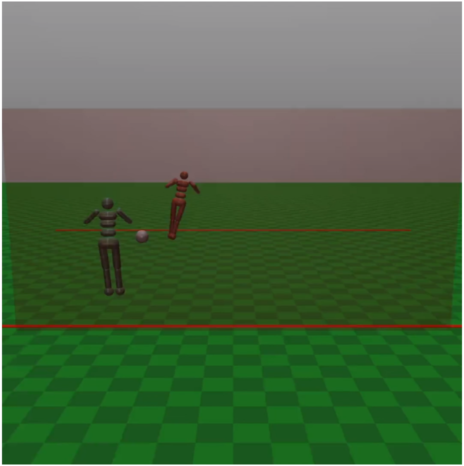
Paired Open-Ended Trailblazer (POET)[6] is an algorithm for evolving populations of agent-environment pairs towards a mix of performance and diversity



Key idea: unboundedly complex environments challenge agents to develop robust, general behaviors

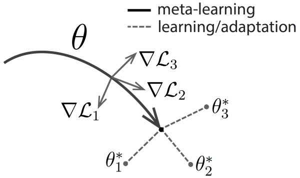
[6] Wang et al, 2019: "Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions "

Competitive multi-agent games create natural curricula that inspire general behaviors (because any fault will get exploited by the opponent). In self-play, agents play games against old versions of themselves to improve:[7]



[7] Bansal et al, 2017: "Emergent Complexity via Multi-Agent Competition"

Model Agnostic Meta-Learning (MAML)[8] trains a model with an objective describing its adaptability: the model at optimum should only need a few gradient steps to quickly learn a new task.



[8] Finn et al, 2017: "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks"

2.3: Spinning Up in Deep RL

Whether you intend to be a practitioner or a researcher:

- Brush up on math! Be familiar with vectors, matrices, gradients, gradient descent, random variables, expectations, variance, and a few other things.
- Brush up on deep learning! **As much as you can get.**
- Become familiar with TF or PyTorch! **You need to tinker to learn.**
- Stay fresh on RL basics! **Fundamentals go a long way.**

For maximum understanding: learn by doing

- Write your own implementations
- Simplicity is critical
- Start with the basics: VPG, DQN, PPO, DDPG
- Focus on understanding: it's the only way you will catch bugs

How silent are silent RL bugs? Pop quiz:

```
for t in range(total_steps):
    act = get_action(obs, epsilon)
    next_obs, rew, done, _ = env.step(act)
    replay_buffer.store(obs, act, rew, next_obs, done)
    ep_ret += rew
    ep_len += 1

    if done:
        epoch_rets.append(ep_ret)
        epoch_lens.append(ep_len)
        obs, rew, done, ep_ret, ep_len = env.reset(), 0, False, 0, 0

    if t > steps_before_training:
        batch = replay_buffer.sample_batch(batch_size)
        feed_dict = {obs_ph: batch['obs1'],
                     obs_targ_ph: batch['obs2'],
                     act_ph: batch['acts'],
                     rew_ph: batch['rews'],
                     done_ph: batch['done']
                     }
        step_loss, cur_q, _ = sess.run([loss, q_vals, train_op], feed_dict=feed_dict)
        epoch_losses.append(step_loss)
        epoch_qs.append(cur_q)

    if t % target_update_freq == 0:
        sess.run(target_update_op)

    epsilon = 1 + (final_epsilon - 1)*min(1, t/finish_decay)

    if (t - steps_before_training) % steps_per_epoch == 0 and (t - steps_before_training)>0:
        # handle logging outputs
```

- If you read a paper, scour it: read all the ablations and search for tricks!
- Skip the hype: find the thing the author's embarassed to admit about why the method doesn't solve the whole field yet
- But don't overfit: sometimes researchers load more tricks than necessary and their idea is better than they realize

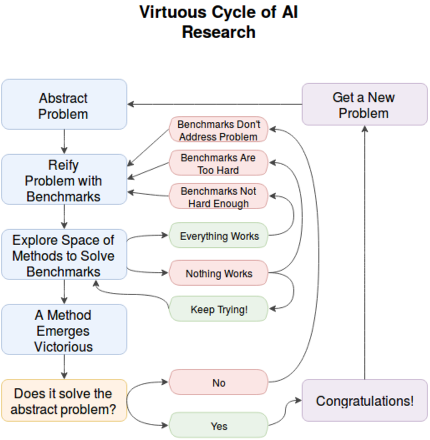
For maximum understanding: "Let all results count with you, but none too much"

- Study existing implementations of algorithms, but don't overfit to them: they made trade-offs you don't have to
- Iterate fast in simple environments, but don't get too confident if your implementation (or new research idea) works in the simplest thing: everything that isn't bugged beats Cartpole, and so do some bugs
- But if it doesn't work, assume there's a bug
- Measure everything. But if your measurements say everything is fine while nothing's working, get new things to measure
- When things start to work, lean in, scale up!
- Keep these habits for research!

If you want to hack with RL tools that are out there...

- OpenAI Gym, Baselines, and Spinning Up
- Many other algorithm libraries: rllab, rllib, rlkit, Intel's Coach, softlearning, ChainerRL, vel, Stable-Baselines, and more
- Very few primitives libraries: TRFL
- Note: there's not a "Tensorflow for RL" yet: if you want it, you have to build it yourself
- RL currently ships as solutions as opposed to modular components

- Do a deep dive in the literature
- Find something you enjoy and care about—"keep only those research ideas which spark joy"
- Consider approach 1: improve something that exists
- Consider approach 2: attack an unsolved benchmark
- Consider approach 3: invent a new problem
- But whatever you pick, really do a thorough lit review to make sure it's new!



End of Part 2

Thank you all! Questions?