Reinforcement Learning and Genetic Algorithms
and Other RL Methods

Rene Gutierrez[1], Keller Jordan[2]

[1]Department of Applied Mathematics & Statistics
UCSC

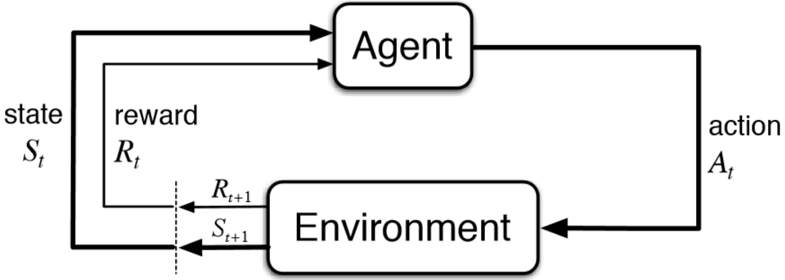
[2]Department of Computer Science
UCSC

CMPS290, Winter 2018

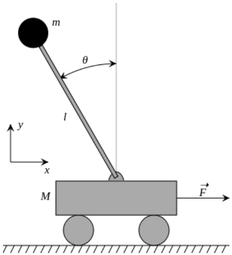
Motivation

What we have done so far

- Supervised Learning:
  - Classification
  - Regression
- Unsupervised Learning:
  - Clustering
  - Dimensionality Reduction
  - Density Estimation

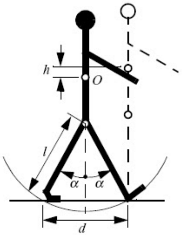

Agent
state
S_t
reward
R_t
action
A_t
R_{t+1}
S_{t+1}
Environment

Examples: Cart-Pole Problem



Objective: Balance a pole on top of a movable cart.
State: Angle, angular speed, position, horizontal velocity.
Action: Horizontal force applied to the cart $\vec{F}$.
Reward: 1 at each time step if the pole is upright.

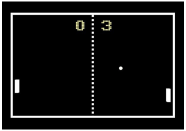
Examples: Robot Locomotion



Objective: Make a Robot move forward.
State: Angle and Position of joints.
Action: Torques applied to joints.
Reward: 1 at each time step if robot moves forward and remains standing.

Examples: Atari Games

(a) Frostbite



(b) Pong

(c) Space Inv

.

Objective: Achieve the highest score.
State: Raw pixel image of the game state.
Action: Game controls. Usually Left, Right, Up, Down.
Reward: Score increase or decrease at each time step.

Examples: GO

Objective: Win the Game.
State: Position of the pieces.
Action: Next move, where to put the next piece down.

Reward: $\begin{cases} 1 & \text{if you win the game} \\ 0 & \text{if you loose the game} \end{cases}$

Theoretical Background

Reinforcement Learning

An Agent interacting with an environment trying to maximize a cumulative reward. The environment is typically modeled as a Markov Decision Process consisting on:

- $\mathcal{S}$: A set of possible states.
- $\mathcal{A}$: A set of possible actions.
- $\mathcal{R}$: A reward distribution on states and actions.
- $\mathbb{P}$: A transition probability distribution given a state and action.
- γ: A discount factor.

Theoretical Background

Key Ideas

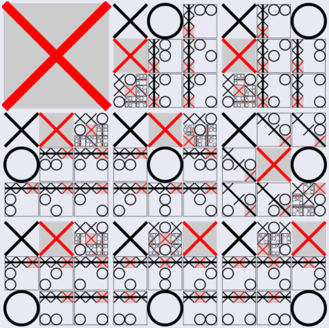
- Markov Property: The current state completely characterises the state of the world (can be relaxed).
- Policy: A policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, is mapping that assigns an action $a \in \mathcal{A}$ to every state $s \in \mathcal{S}$. Or $\pi : \mathcal{S} \rightarrow f_{\mathcal{A}}$ if stochastic.
- Objective: Find a policy π^* such that:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid \pi\right]$$

$$S_0 \sim \mathbb{P}(s_0), a_t \sim \pi(\cdot \mid s_t), s_{t+1} \sim \mathbb{P}(\cdot \mid s_t, a_t)$$

Theoretical Background

Optimal Policy Example



Theoretical Background

Value Function

The value function is a function that assigns a value to each state $s \in \mathcal{S}$ according to the expected cumulative reward following a given policy. That is $V^\pi : \mathcal{S} \to \mathbb{R}$, and is given by:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$$

This function tell us how good is a state.

Theoretical Background

Q-value Function

The Q-value function is a function that assigns a value to each pair of state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ according to the expected cumulative reward of taking an action and then following a given policy. That is $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and is given by:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Theoretical Background

Q-value Function

The Q-value function is a function that assigns a value to each pair of state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ according to the expected cumulative reward of taking an action and then following a given policy. That is $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and is given by:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Theoretical Background

Optimal Q-value function

The optimal Q-value function is a function that maximizes the expected cumulative reward achievable at each pair of state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$. That is $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and is given by:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Theoretical Background

Bellman Equation

Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (See Bellman, 1957, Chap. III.3.)

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}\left[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \mid s_t, a_t\right]$$

Theoretical Background

Value Iteration

We cn use the Bellman Equation as an iterative update

$$Q_{i+1}(s_t, a_t) = \mathbb{E}_{s_{t+1}}\left[r_t + \gamma \max_{a_{t+1}} Q_i(s_{t+1}, a_{t+1}) \mid s_t, a_t\right]$$

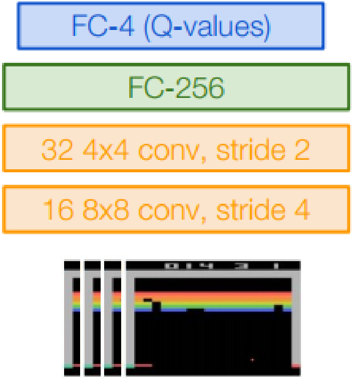
where we will have that:

$$Q_i \to Q^* as i \to \infty$$

Problem: What do we do in continuous state spaces?

- Frame-based games (Atari)
- Robotics control tasks

Solution: use an approximation to $Q(s, a)$ or $\pi(a|s)$



Current state s_t: 84x84x4 stack of last 4 frames

How can we train deep Q-Networks or Policy-Networks?

- Deep Q-Learning
- A3C
- Evolution Strategies (ES)
- Genetic Algorithms (GA)

Deep Q-Learning [1]

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (Dec 2013). Playing Atari with deep reinforcement learning. Technical Report arXiv:1312.5602 [cs.LG], Deepmind Technologies.

Key Features

- Variant of Q-Learning where we use gradient descent to update model parameters
- Uses "replay memory" containing last N experienced transitions, from which minibatches are drawn. This increases data efficiency and decreases variance of updates.
- Surpassed human performance on 3/7 games

Algorithm

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Results

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Key Ideas

We can explore a parametrized space of the policy functions:

$$\Pi = \{\pi_\theta \mid \theta \in \mathbb{R}^m\}$$

and define the corresponding cumulative expected reward

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi_\theta\right]$$

and solve

$$\theta^* = \arg\max_\theta J(\theta)$$

UNIVERSITY OF CALIFORNIA SANTA CRUZ

Key Ideas

If we define a trajectory τ as:

$$\tau = (s_0, a_0, r_0, s_1, \ldots)$$

and the reward associated with it as:

$$r(\tau)$$

we have that:

$$J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}(\tau;\theta)} \left[r(\tau) \right]$$

Key Ideas

How to optimize? Using gradient ascent!

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim \mathbb{P}(\tau;\theta)}\left[r(\tau)\right] \\ &= \int_{\tau} r(\tau)\mathbb{P}(\tau;\theta)d\tau \end{aligned}$$

And we can differentiate:

$$\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau)\nabla_{\theta}\mathbb{P}(\tau;\theta)d\tau$$

which is actually hard to compute!

First Trick

$$\begin{aligned}\nabla_\theta \mathbb{P}(\tau;\theta) &= \mathbb{P}(\tau;\theta)\frac{\nabla_\theta \mathbb{P}(\tau;\theta)}{\mathbb{P}(\tau;\theta)} \\ &= \mathbb{P}(\tau;\theta)\nabla_\theta \log\left(\mathbb{P}(\tau;\theta)\right)\end{aligned}$$

then

$$\begin{aligned}\nabla_\theta J(\theta) &= \int_\tau r(\tau)\mathbb{P}(\tau;\theta)\nabla_\theta \log\left(\mathbb{P}(\tau;\theta)\right) d\tau \\ &= \mathbb{E}_{\tau\sim\mathbb{P}(\tau;\theta)}\left[r(\tau)\nabla_\theta \log\left(\mathbb{P}(\tau;\theta)\right)\right]\end{aligned}$$

which we can estimate using Monte Carlo sampling.

Second Trick

Note that:

$$\mathbb{P}(\tau;\theta) = \prod_{t\geq 0} \mathbb{P}(s_{t+1} \mid s_t, a_t)\pi_\theta(a_t \mid s_t)$$

Then:

$$\log\left(\mathbb{P}(\tau;\theta)\right) = \sum_{t\geq 0} \left[\log\left(\mathbb{P}(s_{t+1} \mid s_t, a_t)\right) + \log\left(\pi_\theta(a_t \mid s_t)\right)\right]$$

Then:

$$\nabla_\theta \log\left(\mathbb{P}(\tau;\theta)\right) = \sum_{t\geq 0} \nabla_\theta \log\left(\pi_\theta(a_t \mid s_t)\right)$$

which does not depend on the transition probabilities.

Result

Applying both tricks we have that:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log\left(\pi_\theta(a_t \mid s_t)\right)$$

Unfortunately, this estimate suffers for high variance, and a lot for trajectories would be needed to obtain a good estimate.

Tricks for Variance Reduction

- Only consider future rewards. At time t instead of $r(\tau)$ use $\sum_{t' \geq t} r_{t'}$.
- Use a discount factor. That is, instead of $\sum_{t' \geq t} r_{t'}$ use $\sum_{t' \geq t} \gamma^{t'-t} r_{t'}$.
- Don't use the raw value of the reward but consider it against a base line. That is, instead of $\sum_{t' \geq t} \gamma^{t'-t} r_{t'}$ use $\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t)$.
- A good excess reward indicator is given by the Q-function and the value function. That is:

$$\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \approx Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$$

Tricks for Variance Reduction

So to reduce the variance, use:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \right) \nabla_\theta \log \left(\pi_\theta(a_t \mid s_t) \right)$$

since we don't know Q and V we can use Q-learning!

Actor-Critic Policy Gradient Methods

Key Ideas

- The Actor decides the policy π_θ.
- The Critic tells the actor how good the policy is:

$$A^\pi(s,a) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \approx \sum_{t' \geq t} \gamma^{t'-t} r_{t'} - V^{\pi_\theta}(s_t)$$

  which we can call the advantage of policy π.
- The Critic only needs to learn the advantage for the policy trajectory.

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ϕ

For iteration=1, 2 … **do**

Sample m trajectories under the current policy

$\Delta\theta \leftarrow 0$

For i=1, …, m **do**

For t=1, … , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$\theta \leftarrow \alpha\Delta\theta$

$\phi \leftarrow \beta\Delta\phi$

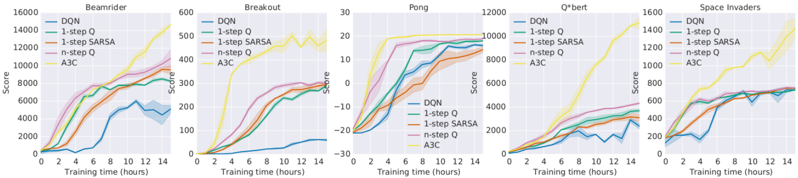
End for

A3C

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
// Assume global shared parameter vectors θ and θ_v and global shared counter T = 0
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter t ← 1
repeat
    Reset gradients: dθ ← 0 and dθ_v ← 0.
    Synchronize thread-specific parameters θ' = θ and θ'_v = θ_v
    t_start = t
    Get state s_t
    repeat
        Perform a_t according to policy π(a_t|s_t; θ')
        Receive reward r_t and new state s_{t+1}
        t ← t + 1
        T ← T + 1
    until terminal s_t or t − t_start == t_max
    R = { 0            for terminal s_t
        { V(s_t, θ'_v)  for non-terminal s_t // Bootstrap from last state
    for i ∈ {t − 1, ..., t_start} do
        R ← r_i + γR
        Accumulate gradients wrt θ': dθ ← dθ + ∇_θ' log π(a_i|s_i; θ')(R − V(s_i; θ'_v))
        Accumulate gradients wrt θ'_v: dθ_v ← dθ_v + ∂(R − V(s_i; θ'_v))^2 / ∂θ'_v
    end for
    Perform asynchronous update of θ using dθ and of θ_v using dθ_v.
until T > T_max
```

A3C Comparison



Evolution Strategies [2]

[2] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864, 2017. URL http://arxiv.org/abs/1703. 03864.

Key Features

- Simpler and fewer assumptions compared to current RL algorithms
- Finite-differences gradient approximation
- Highly parallelizable
- Competitive with other methods on benchmarks

Algorithm

Algorithm 1 Evolution Strategies

1: **Input:** Learning rate α, noise standard deviation σ, initial policy parameters θ_0
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
4: Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
5: Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
6: **end for**

Algorithm

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$ using known random seeds
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

Results

- On Atari, ES is competitive with A3C (does better in 23 / 51 games)
- Avoids complexity of Deep Q-Learning/A3C
  - No backward pass or gradient updates
  - Policy function approximator can be nondifferentiable
  - No need to store replay memory
- Highly Parallelizable (communicate only rewards if seeds are known)
- Therefore much lower training time when distributed across many cpus
- Avoids issue of credit assignment over long time scale

Deep Genetic Algorithms [3]

[3]Petroski Such, Felipe, Madhavan, Vashisht, Conti, Edoardo, Lehman, Joel, Stanley, Kenneth O., and Clune, Jeff. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint to appear, 2017.

Key Features

- Truly gradient-free method
- Maintains “generation” of models, allowing for greater diversity in policies
- Even more parallelizable

Algorithm

Algorithm 1 Simple Genetic Algorithm

Input: mutation power σ, population size N, number of selected individuals T, policy initialization routine ϕ.
for $g = 1, 2...G$ generations **do**
  for $i = 1, ..., N$ in next generation's population **do**
    if $g = 1$ **then**
      $\mathcal{P}_i^g = \phi(\mathcal{N}(0, I))$ {initialize random DNN}
      $F_i^g = F(\mathcal{P}_i^g)$ {assess its fitness}
    else
      if $i = 1$ **then**
        $\mathcal{P}_i^g = \mathcal{P}_i^{g-1}; F_i^g = F_i^{g-1}$ {copy the elite}
      else
        $k = \text{uniformRandom}(1, T)$ {select parent}
        Sample $\epsilon \sim \mathcal{N}(0, I)$
        $\mathcal{P}_i^g = \mathcal{P}_k^{g-1} + \sigma\epsilon$ {mutate parent}
        $F_i^g = F(\mathcal{P}_i^g)$ {assess its fitness}
  Sort $\mathcal{P}^g$ and F^g with descending order by F^g
Return: highest performing policy, $\mathcal{P}_1^g$

Algorithm

Even more parallelizable than ES

- Represent each model by the sequence of random seeds that generated it

Results

- DQN, ES, GA all score highest on 3 games, A3C 4
- Sometimes GA beats other methods within first generation – what if we try random search?

Results

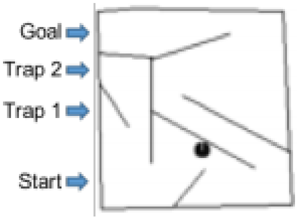
	DQN	Evolution Strategies	Random Search	GA	GA	A3C
Frames, Time	200M, ~7-10d	1B, ~ 1h	1B, ~ 1h	1B, ~ 1h	4B, ~ 4h	1.28B, 4d
Forward Passes	450M	250M	250M	250M	1B	960M
Backward Passes	400M	0	0	0	0	640M
Operations	1.25B U	250M U	250M U	250M U	1B U	2.24B U
Amidar	**978**	112	151	216	294	264
Assault	4,280	1,674	642	819	1,006	**5,475**
Asterix	4,359	1,440	1,175	1,885	2,392	**22,140**
Asteroids	1,365	1,562	1,404	2,056	2,056	**4,475**
Atlantis	279,987	**1,267,410**	45,257	79,793	125,883	911,091
Enduro	**729**	95	32	39	50	-82
Frostbite	797	370	1,379	**4,801**	**5,623**	191
Gravitar	473	**805**	290	462	637	304
Kangaroo	7,259	**11,200**	1,773	8,667	10,920	94
Seaquest	**5,861**	1,390	559	807	1,241	2,355
Skiing	-13,062	-15,442	-8,816	**-6,995**	**-6,522**	-10,911
Venture	163	760	547	**810**	**1,093**	23
Zaxxon	5,363	6,380	2,943	5,183	6,827	**24,622**

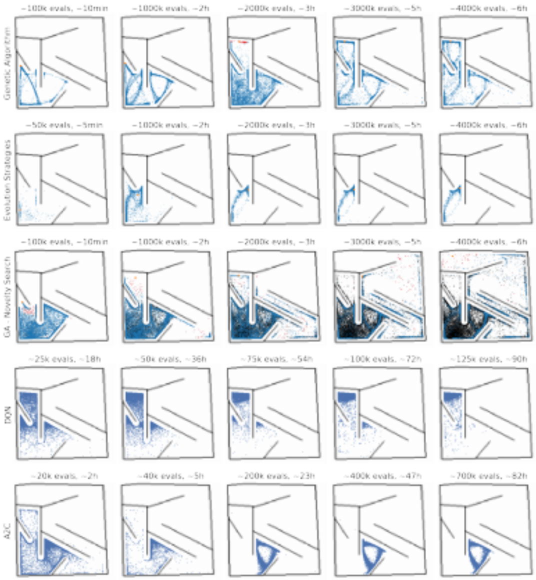
Results

- DQN, ES, GA all score highest on 3 games, A3C 4
- Sometimes GA beats other methods within first generation – what if we try random search?
- Random search outperforms DQN on 4 out of 13 games, ES on 3, A3C on 5 (!)
- Can use novelty search techniques to solve problems no other approach can tackle

Results

Image Hard Maze




~100k evals, ~10min
~1000k evals, ~2h
~2000k evals, ~3h
~3000k evals, ~5h
~4000k evals, ~6h
Genetic Algorithm
~50k evals, ~5min
~1000k evals, ~2h
~2000k evals, ~3h
~3000k evals, ~5h
~4000k evals, ~6h
Evolution Strategies
~100k evals, ~10min
~1000k evals, ~2h
~2000k evals, ~3h
~3000k evals, ~5h
~4000k evals, ~6h
GA - Novelty Search
~25k evals, ~18h
~50k evals, ~36h
~75k evals, ~54h
~100k evals, ~72h
~125k evals, ~90h
DQN
~20k evals, ~2h
~40k evals, ~5h
~200k evals, ~23h
~400k evals, ~47h
~700k evals, ~82h
A2C

Results

- DQN, ES, GA all score highest on 3 games, A3C 4
- Sometimes GA beats other methods within first generation – what if we try random search?
- Random search outperforms DQN on 4 out of 13 games, ES on 3, A3C on 5 (!)
- Can use novelty search techniques to solve problems no other approach can tackle
- Compact network encoding
  - Number of generations is typically low for these problems (10-200)
  - Networks are represented as sequence of seeds to generate perturbations from initial weights
  - 4m parameters → thousands of bytes (10,000 fold compression)

Takeaways

- Each approach performs well on some tasks and poorly on others
- Some standard benchmarks are simple enough to solve using random search
- Future research directions:
  - Implement existing methods from GA literature to deep GA
  - Potential to use nondifferentiable models (binary networks)
  - Create new algorithms that combine best of each