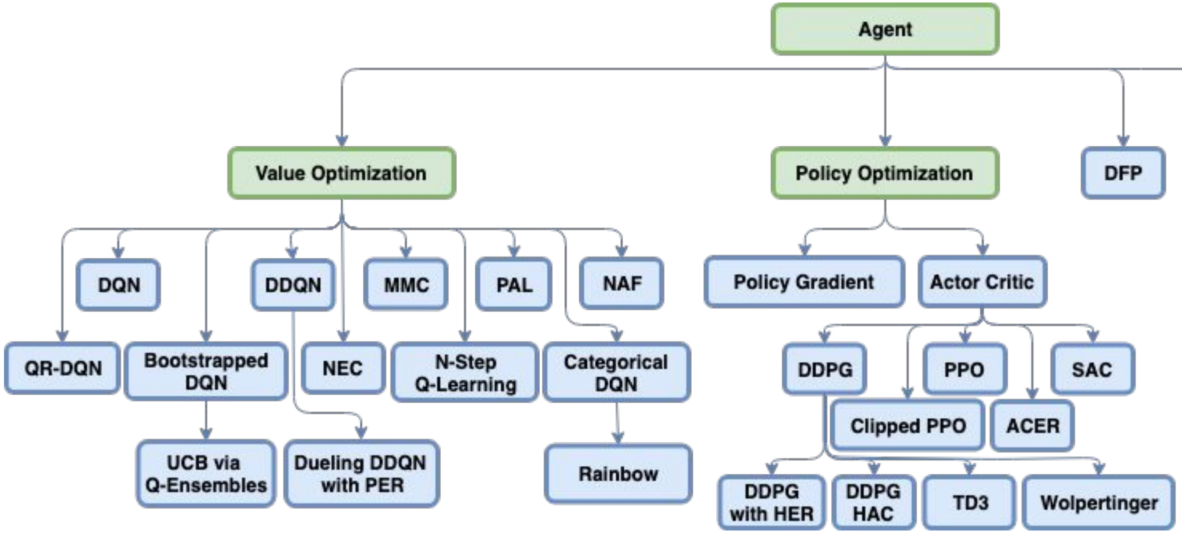
Evolving Reinforcement Learning Algorithms

JD Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Sergey Levine,
Quoc V. Le, Honglak Lee, Aleksandra Faust

BAIR
BERKELEY ARTIFICIAL INTELLIGENCE RESEARCH

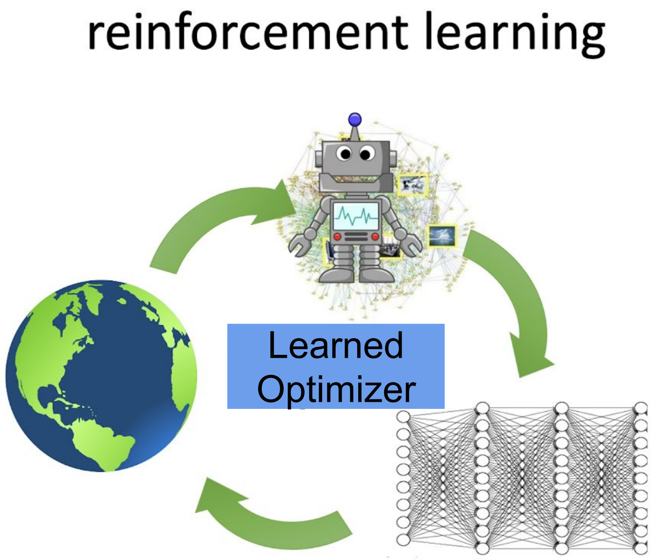
Wide Choice of RL Algorithms



[Nervana Systems Coach Diagram]

Desire: General purpose RL algorithms without manual effort.
Problem: Can we meta-learn RL algorithms that generalize well on unseen tasks?

RL Algorithm as a Learned Optimizer



- Learning procedure which takes in MDP and transforms experience into optimal behavior
- Can we meta-learn the optimizer?
    - Improved performance
    - Generalize to unseen environments
    - Interpretable
    - Scale with data and compute

Example: Simple Modifications to Existing Algorithms

$$\delta^2 = (Q(s_t, a_t) - (r_t + \gamma * \max_a Q(s_{t+1}, a)))^2$$

[1] Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative Q-Learning for Offline Reinforcement Learning. *ArXiv, abs/2006.04779*.

Example: Simple Modifications to Existing Algorithms

CQL: adds scaled log softmax policy to TD error

$$\delta^2 + \beta \log \sum_a \exp\left(Q(s_t, a)\right) - Q(s_t, a_t)$$

[1] Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative Q-Learning for Offline Reinforcement Learning. *ArXiv, abs/2006.04779*.

Example: Simple Modifications to Existing Algorithms

CQL: adds scaled log softmax policy to TD error

$$\delta^2 + \beta \log \sum_a \exp\left(Q(s_t, a)\right) - Q(s_t, a_t)$$

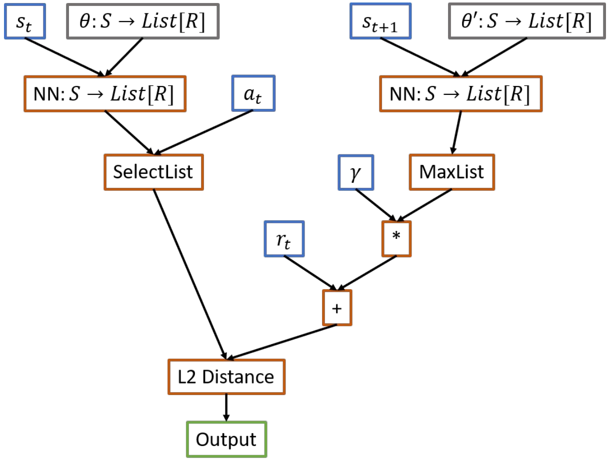
M-DQN: adds scaled log policy to reward

$$\hat{q}_{\text{m-dqn}}(r_t, s_{t+1}) = r_t + \alpha\tau \ln \pi_{\bar{\theta}}(a_t|s_t) + \gamma \sum_{a' \in \mathcal{A}} \pi_{\bar{\theta}}(a'|s_{t+1}) \Big(q_{\bar{\theta}}(s_{t+1}, a') - \tau \ln \pi_{\bar{\theta}}(a'|s_{t+1}) \Big)$$

[1] Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative Q-Learning for Offline Reinforcement Learning. *ArXiv, abs/2006.04779*.
[2] Vieillard, N., Pietquin, O., & Geist, M. (2020). Munchausen Reinforcement Learning. *ArXiv, abs/2007.14430*.

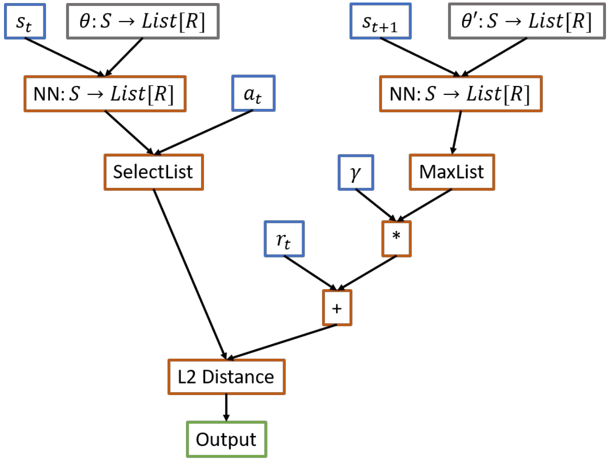
Evolving RL Algorithms

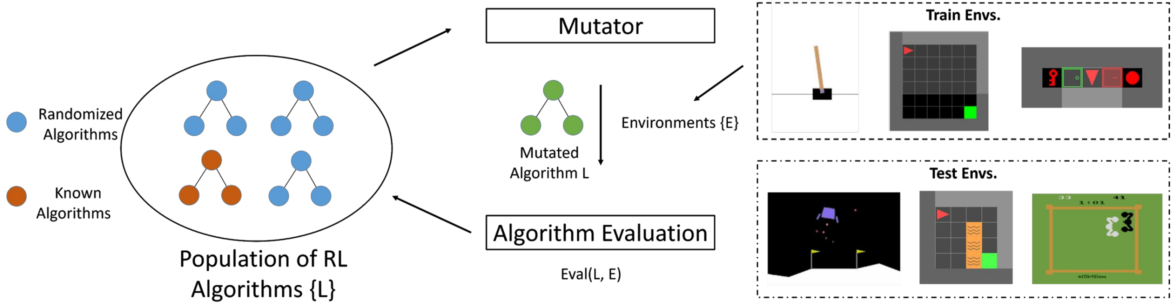
- **Insight:** RL algorithm as a computational graph



Evolving RL Algorithms

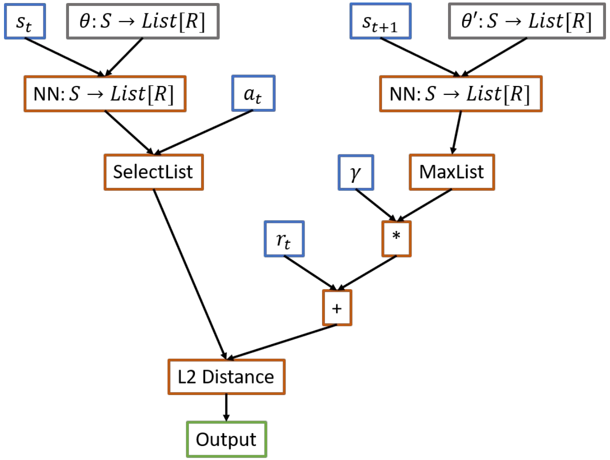
- **Insight:** RL algorithm as a computational graph
- **Method:** Evolve population of graphs by mutating, training, and evaluating RL agents

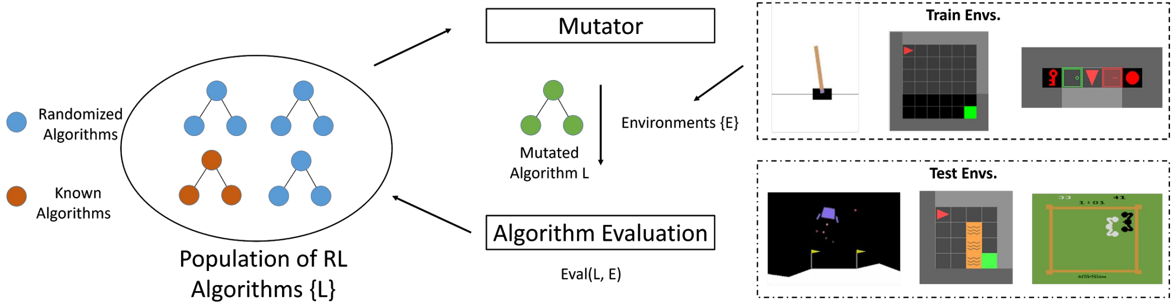




Evolving RL Algorithms

- **Insight:** RL algorithm as a computational graph
- **Method:** Evolve population of graphs by mutating, training, and evaluating RL agents
- **Result:** Learn new algorithms which generalize to unseen environments





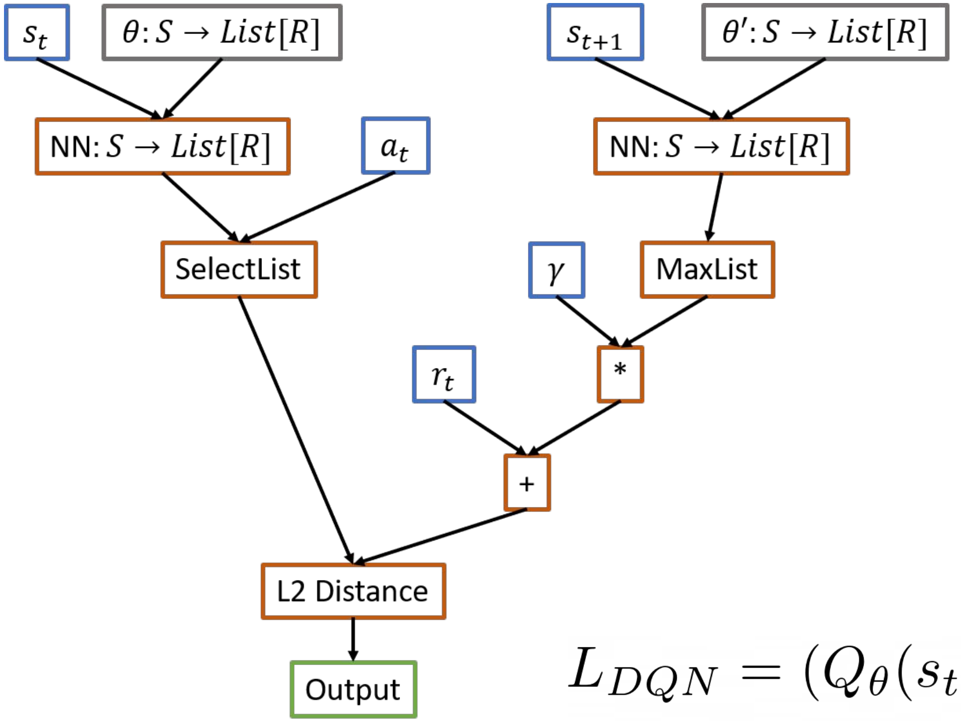
Prior Work

- **Genetic Programming**
  - Holland 1975, Koza 1993, Schmidhuber 1987
  - **AutoML**: Zoph & Le 2016, Hutter 2018, Real et al. 2020
  - Mostly applied to SL
- **Meta-learning in RL**
  - **Adaptation:** Finn & Levine 2018
  - **RNNs:** Duan et al. 2016, Wang et al. 2017
  - Not domain agnostic
- **Learning RL Algorithms**
  - **Metagradients**: Kirsch et al. 2020, Oh et al. 2020
  - Not interpretable
  - **Exploration**: Alet et al. 2020

Algorithm Representation

Expressive **Interpretable** **Generalizable**

RL Algorithm as a Computational Graph



Node Types

Inputs

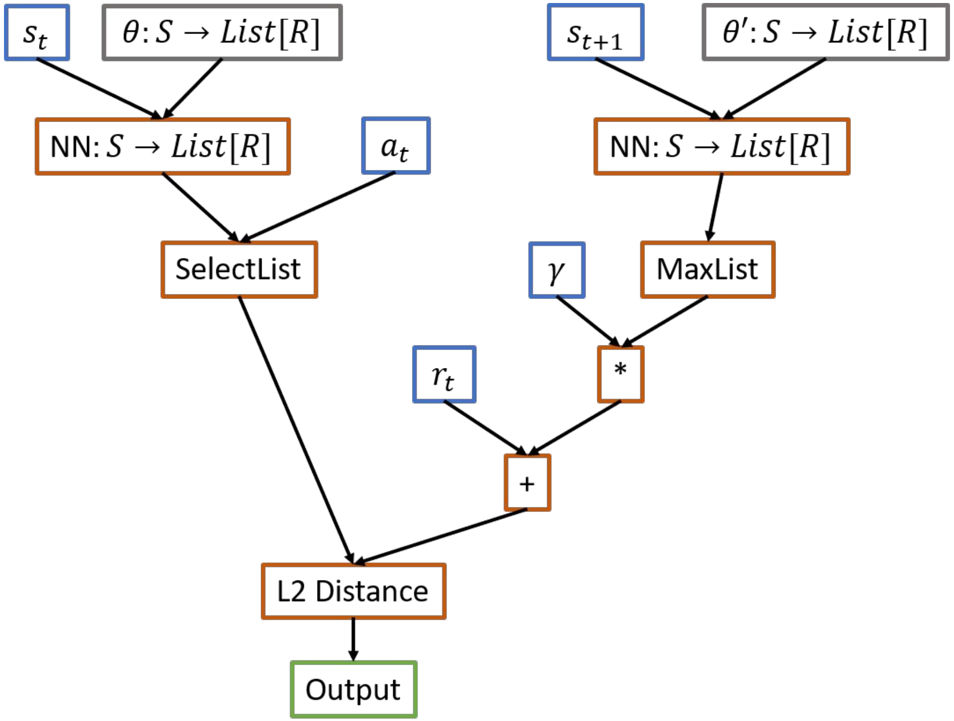
Operators

Parameters

Output

$$L_{DQN} = (Q_\theta(s_t, a_t) - (r_t + \gamma * \max_a Q_{\theta'}(s_{t+1}, a)))^2$$

RL Algorithm as a Computational Graph

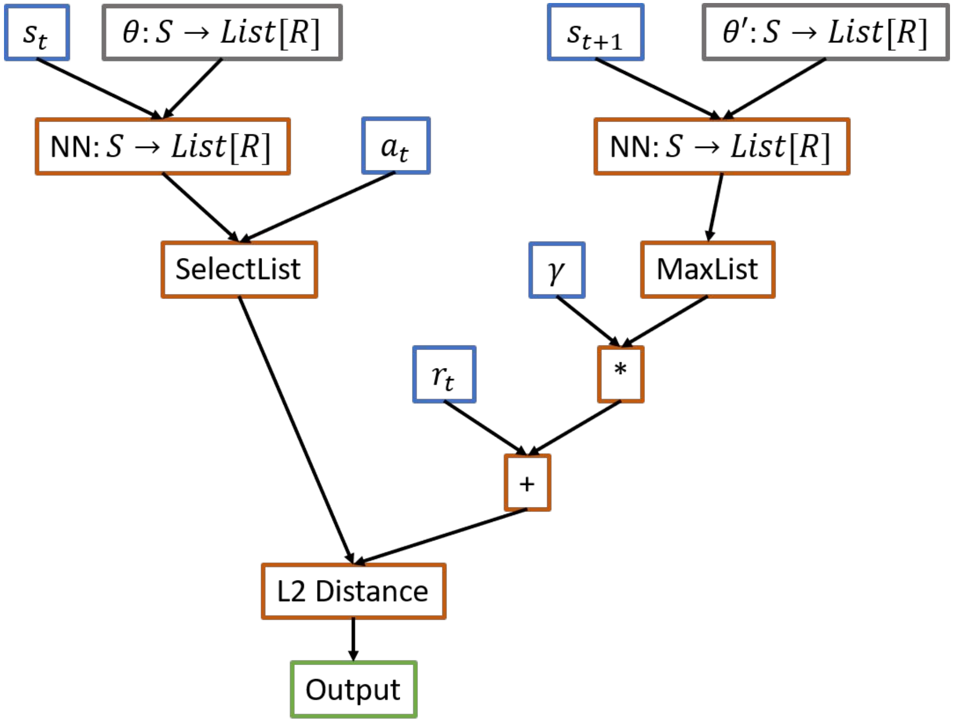


Data Types

State	$\mathbb{S}$
Action	$\mathbb{A}$
Float	$\mathbb{R}$
List	$List[\mathbb{X}]$
Probability	$\mathbb{P}$
Vector	$\mathbb{V}$

Typing allows for domain agnostic programs and type checking

RL Algorithm as a Computational Graph

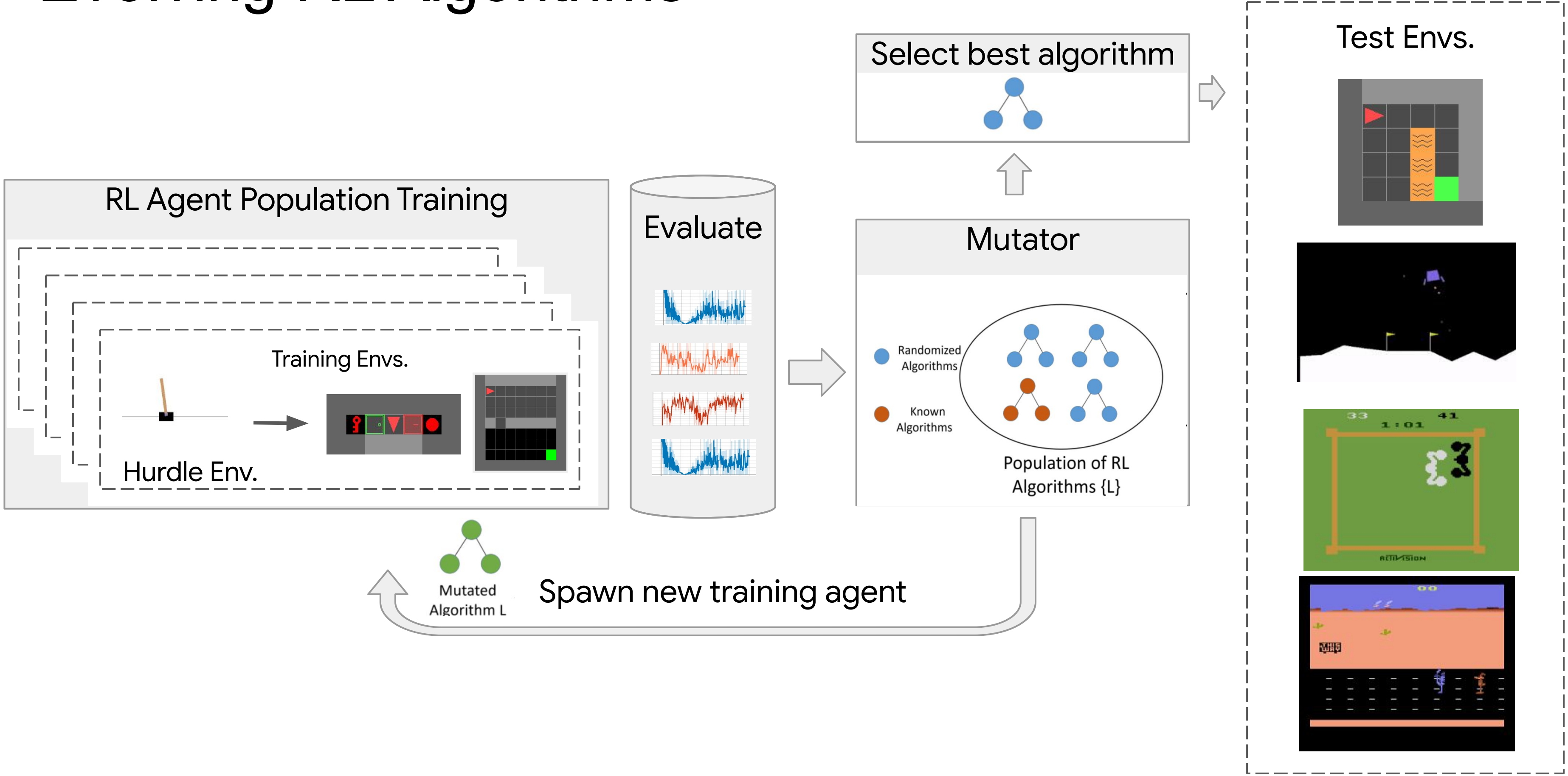


Operation	Input Types	Output Type
Add	$\mathbb{X}, \mathbb{X}$	$\mathbb{X}$
Subtract	$\mathbb{X}, \mathbb{X}$	$\mathbb{X}$
Max	$\mathbb{X}, \mathbb{X}$	$\mathbb{X}$
Min	$\mathbb{X}, \mathbb{X}$	$\mathbb{X}$
DotProduct	$\mathbb{X}, \mathbb{X}$	$\mathbb{R}$
Div	$\mathbb{X}, \mathbb{X}$	$\mathbb{X}$
L2Distance	$\mathbb{X}, \mathbb{X}$	$\mathbb{R}$
MaxList	$List[\mathbb{R}]$	$\mathbb{R}$
MinList	$List[\mathbb{R}]$	$\mathbb{R}$
ArgMaxList	$List[\mathbb{R}]$	$\mathbb{Z}$
SelectList	$List[\mathbb{X}], \mathbb{Z}$	$\mathbb{X}$
MeanList	$List[\mathbb{X}]$	$\mathbb{X}$
VarianceList	$List[\mathbb{X}]$	$\mathbb{X}$
Log	$\mathbb{X}$	$\mathbb{X}$
Exp	$\mathbb{X}$	$\mathbb{X}$
Abs	$\mathbb{X}$	$\mathbb{X}$
(C)NN:$\mathbb{S} \to List[\mathbb{R}]$	$\mathbb{S}$	$List[\mathbb{R}]$
(C)NN:$\mathbb{S} \to \mathbb{R}$	$\mathbb{S}$	$\mathbb{R}$
(C)NN:$\mathbb{S} \to \mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$
Softmax	$List[\mathbb{R}]$	$\mathbb{P}$
KLDiv	$\mathbb{P}, \mathbb{P}$	$\mathbb{R}$
Entropy	$\mathbb{P}$	$\mathbb{R}$
Constant		1, 0.5, 0.2, 0.1, 0.01
MultiplyTenth	$\mathbb{X}$	$\mathbb{X}$
Normal(0, 1)		$\mathbb{R}$
Uniform(0, 1)		$\mathbb{R}$

Outer loop Optimization

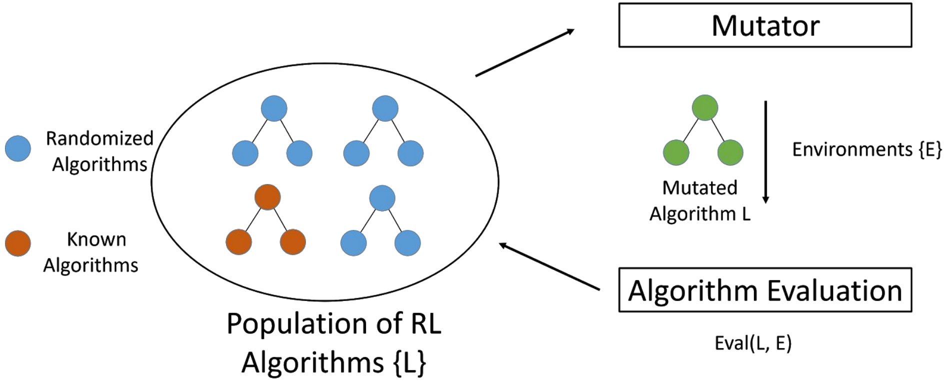
How to scale with compute?

Evolving RL Algorithms



Evolving Reinforcement Learning Algorithms, Co-Reyes, Miao, Peng, Real, Levine, Le, Lee, Faust, ICLR 2021 https://sites.google.com/view/evolvingrl

Meta-Learn RL Algorithms



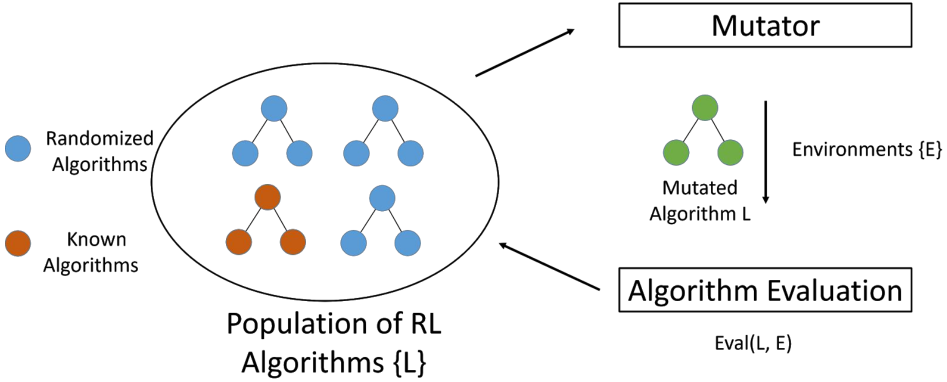
Algorithm 1 Algorithm Evaluation, $\mathrm{Eval}(L, \mathcal{E})$

```
Input: RL Algorithm L, Environment ℰ, training episodes M
Initialize: Q-value parameters θ, target parameters θ′ empty replay buffer 𝒟
for i = 1 to M do
    for t = 0 to T do
        With probability ϵ, select a random action a_t,
        otherwise select a_t = arg max_a Q(s_t, a)
        Step environment s_{t+1}, r_t ~ ℰ(a_t, s_t)
        𝒟 ← 𝒟 ∪ {s_t, a_t, r_t, s_{t+1}}
        Update parameters θ ← θ − ∇_θ L(s_t, a_t, r_t, s_{t+1}, θ, γ)
        Update target θ′ ← θ
    end for
    Compute episode return R_m = Σ_{t=0}^{T} r_t
end for
Output:
    Normalized training performance (1/M) Σ_{m=1}^{M} (R_m − R_min)/(R_max − R_min)
```

- Learn loss function for DQN style update procedure
- Score each algorithm with normalized training performance

Evolving Reinforcement Learning Algorithms, Co-Reyes, Miao, Peng, Real, Levine, Le, Lee, Faust, ICLR 2021 https://sites.google.com/view/evolvingrl

Meta-Learn RL Algorithms

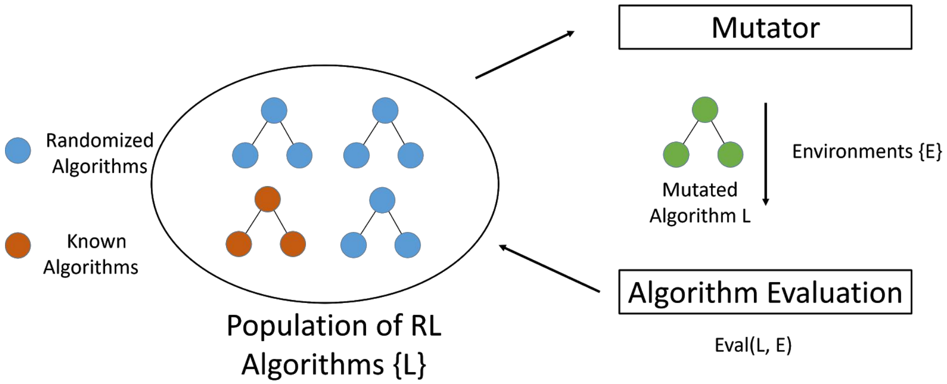


Algorithm 2 Evolving RL Algorithms

Input: Training environments $\{\mathcal{E}\}$, hurdle environment $\mathcal{E}_h$, hurdle threshold α, optional existing algorithm A
Initialize: Population P of RL algorithms $\{L\}$, history H, randomized inputs I. If bootstrapping, initialize P with A.
Score each L in P with $H[L].score \leftarrow \sum_{\mathcal{E}} \text{Eval}(L, \mathcal{E})$
for $c = 0$ **to** C **do**
    Sample tournament $T \sim Uniform(P)$
    Parent algorithm $L \leftarrow$ highest score algorithm in T
    Child algorithm $L' \leftarrow$ Mutate(L)
    $H[L'].hash \leftarrow \text{Hash}(L'(I))$
    if $H[L'].hash$ was new **and** $\text{Eval}(L', \mathcal{E}_h) > \alpha$ **then**
        $H[L'].score \leftarrow \sum_{\mathcal{E}} \text{Eval}(L', \mathcal{E})$
    end if
    Add L' to population P
    Remove oldest L from population
end for
Output: Algorithm L with highest score

- Regularized Evolution for outer loop optimization

Optimizations

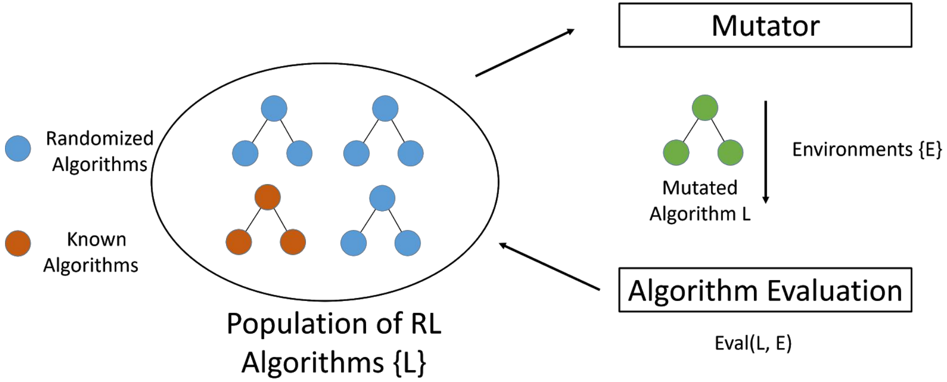


Algorithm 2 Evolving RL Algorithms

Input: Training environments $\{\mathcal{E}\}$, hurdle environment $\mathcal{E}_h$, hurdle threshold α, optional existing algorithm A
Initialize: Population P of RL algorithms $\{L\}$, history H, randomized inputs I. If bootstrapping, initialize P with A.
Score each L in P with $H[L].score \leftarrow \sum_{\mathcal{E}} \mathrm{Eval}(L, \mathcal{E})$
for $c = 0$ **to** C **do**
  Sample tournament $T \sim Uniform(P)$
  Parent algorithm $L \leftarrow$ highest score algorithm in T
  Child algorithm $L' \leftarrow$ Mutate(L)
  $H[L'].hash \leftarrow \mathrm{Hash}(L'(I))$
  if $H[L'].hash$ was new **and** $\mathrm{Eval}(L', \mathcal{E}_h) > \alpha$ **then**
    $H[L'].score \leftarrow \sum_{\mathcal{E}} \mathrm{Eval}(L', \mathcal{E})$
  end if
  Add L' to population P
  Remove oldest L from population
end for
Output: Algorithm L with highest score

- Don't reevaluate functionally equivalent or duplicate programs
- Saves 70% of computation

Optimizations

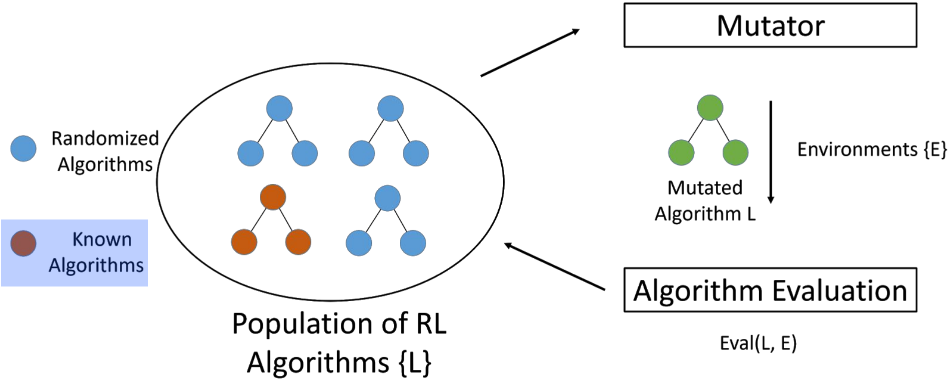


Algorithm 2 Evolving RL Algorithms

Input: Training environments $\{\mathcal{E}\}$, hurdle environment $\mathcal{E}_h$, hurdle threshold α, optional existing algorithm A
Initialize: Population P of RL algorithms $\{L\}$, history H, randomized inputs I. If bootstrapping, initialize P with A.
Score each L in P with $H[L].score \leftarrow \sum_{\mathcal{E}} \text{Eval}(L, \mathcal{E})$
for $c = 0$ **to** C **do**
    Sample tournament $T \sim Uniform(P)$
    Parent algorithm $L \leftarrow$ highest score algorithm in T
    Child algorithm $L' \leftarrow$ Mutate(L)
    $H[L'].hash \leftarrow \text{Hash}(L'(I))$
    if $H[L'].hash$ was new **and** $\text{Eval}(L', \mathcal{E}_h) > \alpha$ **then**
        $H[L'].score \leftarrow \sum_{\mathcal{E}} \text{Eval}(L', \mathcal{E})$
    end if
    Add L' to population P
    Remove oldest L from population
end for
Output: Algorithm L with highest score

- Stop early if performance on hurdle environment is bad
- Saves additional 30% of computation

Bootstrap from existing algorithms



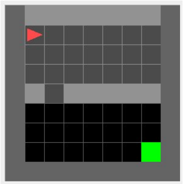
Algorithm 2 Evolving RL Algorithms

Input: Training environments $\{\mathcal{E}\}$, hurdle environment $\mathcal{E}_h$, hurdle threshold α, optional existing algorithm A
Initialize: Population P of RL algorithms $\{L\}$, history H, randomized inputs I. If bootstrapping, initialize P with A.
Score each L in P with $H[L].score \leftarrow \sum_{\mathcal{E}} \text{Eval}(L, \mathcal{E})$
for $c = 0$ **to** C **do**
    Sample tournament $T \sim Uniform(P)$
    Parent algorithm $L \leftarrow$ highest score algorithm in T
    Child algorithm $L' \leftarrow$ Mutate(L)
    $H[L'].hash \leftarrow \text{Hash}(L'(I))$
    if $H[L'].hash$ was new **and** $\text{Eval}(L', \mathcal{E}_h) > \alpha$ **then**
        $H[L'].score \leftarrow \sum_{\mathcal{E}} \text{Eval}(L', \mathcal{E})$
    end if
    Add L' to population P
    Remove oldest L from population
end for
Output: Algorithm L with highest score

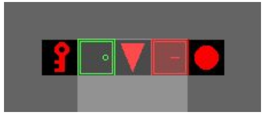
- Can initialize population with existing algorithms

Environments

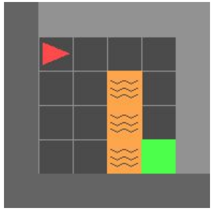
<u>Train</u>

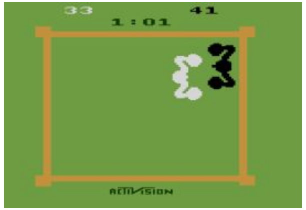


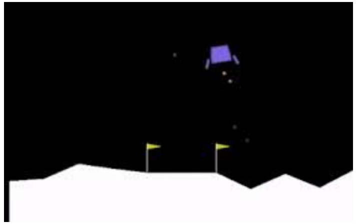
Hurdle Env.



<u>Test</u>







- Want training environments that are computationally cheap but diverse
- Test environments include completely different state and action sizes (including image observations)

Results

Learned Algorithm 1: DQN_Clipped as Constrained Optimization

$$Y_t = r_t + \gamma * \max_a Q_{targ}(s_t, a), \text{ and } \delta = Q(s_t, a_t) - Y_t.$$

$$L_{\text{DQNClipped}} = \max\left[Q(s_t, a_t), \delta^2 + Y_t\right] + \max\left[Q(s_t, a_t) - Y_t, \gamma(\max_a Q_{targ}(s_t, a))^2\right]$$

Learned Algorithm 1: DQN_Clipped as Constrained Optimization

$$Y_t = r_t + \gamma * \max_a Q_{targ}(s_t, a), \text{ and } \delta = Q(s_t, a_t) - Y_t.$$

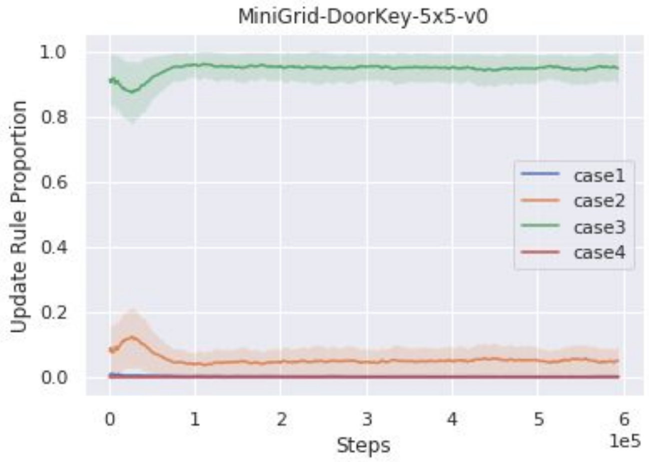
$$L_{\text{DQNClipped}} = \max\left[Q(s_t, a_t), \delta^2 + Y_t\right] + \max\left[Q(s_t, a_t) - Y_t, \gamma(\max_a Q_{targ}(s_t, a))^2\right]$$

$$\text{Case 2: } Q(s_t, a_t) - Y_t > \delta^2$$

- Minimize Q

$$\text{Case 3: } Q(s_t, a_t) - Y_t \leq \delta^2$$

- Minimize normal TD error



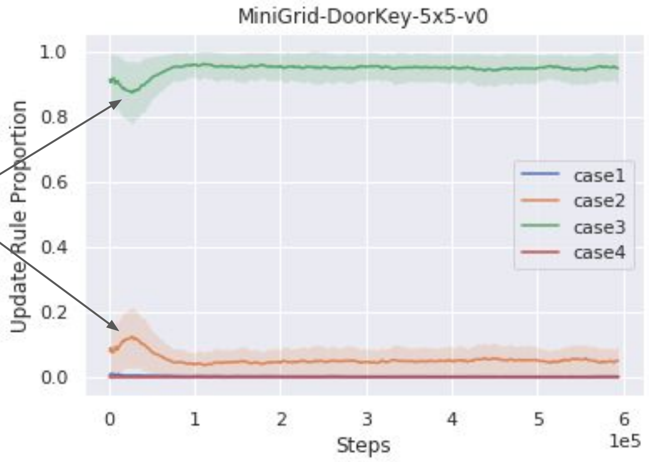
Learned Algorithm 1: DQN_Clipped as Constrained Optimization

$$Y_t = r_t + \gamma * \max_a Q_{targ}(s_t, a), \text{ and } \delta = Q(s_t, a_t) - Y_t.$$

$$L_{\text{DQNClipped}} = \max\left[Q(s_t, a_t), \delta^2 + Y_t\right] + \max\left[Q(s_t, a_t) - Y_t, \gamma(\max_a Q_{targ}(s_t, a))^2\right]$$

Case 2: $Q(s_t, a_t) - Y_t > \delta^2$

- Minimize Q



Case 3: $Q(s_t, a_t) - Y_t \leq \delta^2$

- Minimize normal TD error

Learned Algorithm 2: DQN_Reg as Soft Constraint

$$Y_t = r_t + \gamma * \max_a Q_{targ}(s_t, a), \text{ and } \delta = Q(s_t, a_t) - Y_t.$$

$$L_{\text{DQNReg}} = 0.1 * Q(s_t, a_t) + \delta^2$$

Learned Algorithm 2: DQN_Reg as Soft Constraint

$$Y_t = r_t + \gamma * \max_a Q_{targ}(s_t, a), \text{ and } \delta = Q(s_t, a_t) - Y_t.$$

$$L_{\text{DQNReg}} = 0.1 * Q(s_t, a_t) + \delta^2$$

$$L_{CQL} = \beta \log \sum_a \exp\left(Q(s_t, a)\right) - Q(s_t, a_t) + \delta^2$$

Learned Algorithm 2: DQN_Reg as Soft Constraint

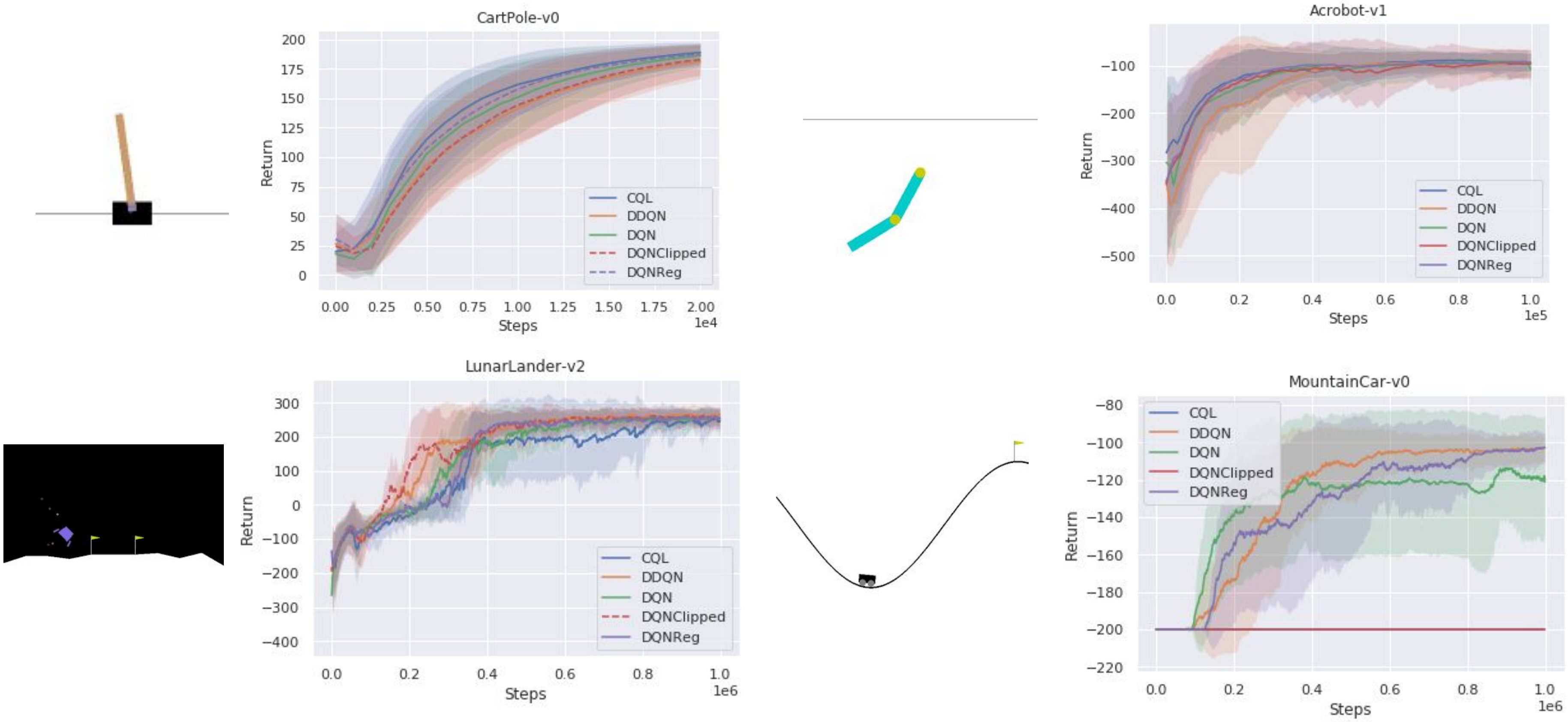
$$Y_t = r_t + \gamma * \max_a Q_{targ}(s_t, a), \text{ and } \delta = Q(s_t, a_t) - Y_t.$$

$$L_{\text{DQNReg}} = 0.1 * Q(s_t, a_t) + \delta^2$$

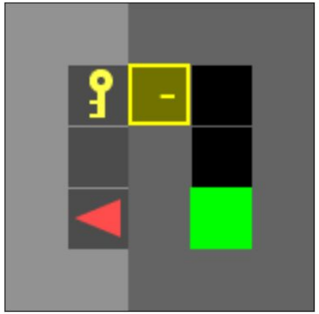
$$L_{CQL} = \beta \log \sum_a \exp\left(Q(s_t, a)\right) - Q(s_t, a_t) + \delta^2$$

- DQNReg as version of entropy regularization that penalizes Q-values on dataset to prevent overfitting

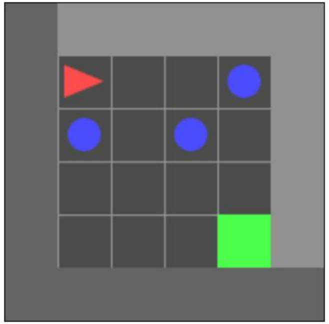
Generalize to Unseen Environments



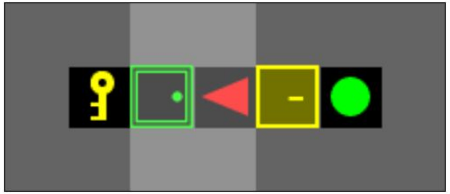
DQNReg Outperforms on Sparse Reward Train Envs.



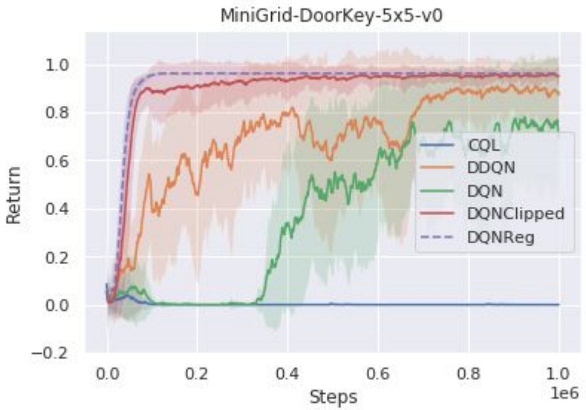
use the key to open the door and then get to the goal

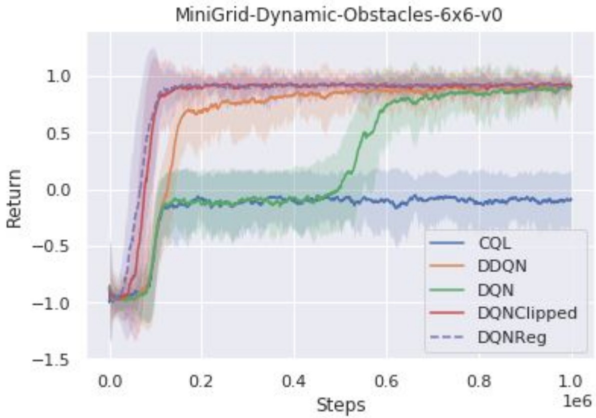


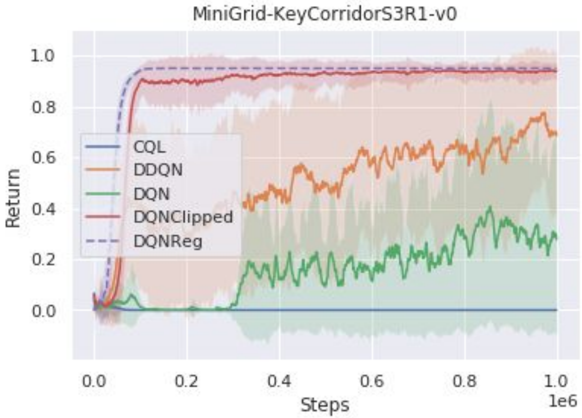
get to the green goal square



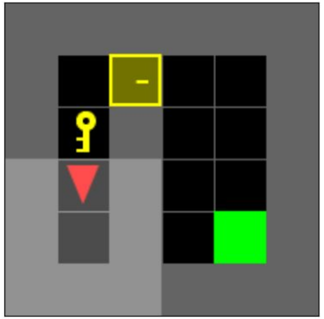
pick up the green ball



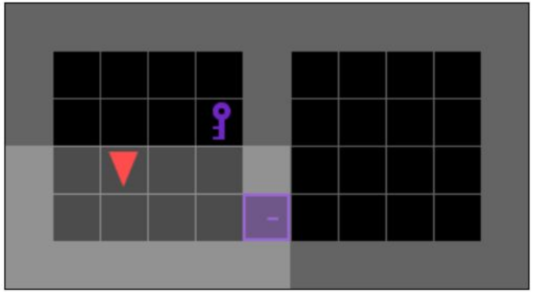




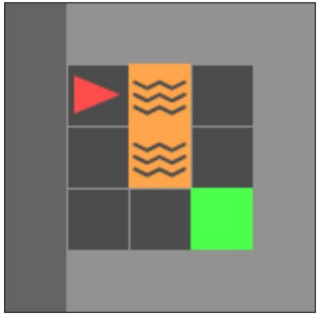
DQNReg Generalizes to Sparse Reward Test Envs.



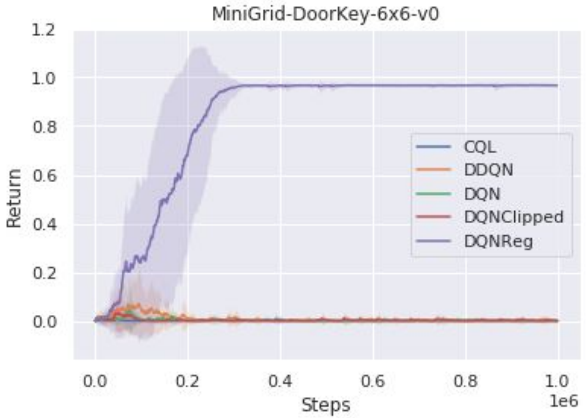
use the key to open the door and then get to the goal

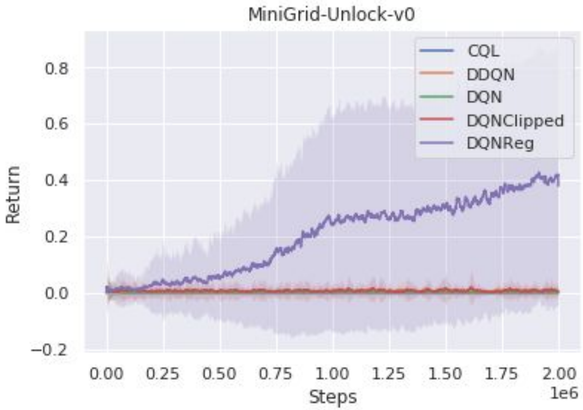


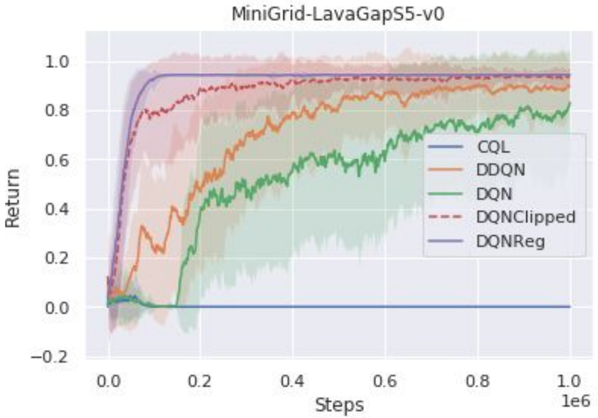
open the door



avoid the lava and get to the green goal square

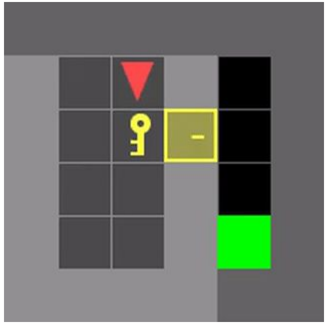


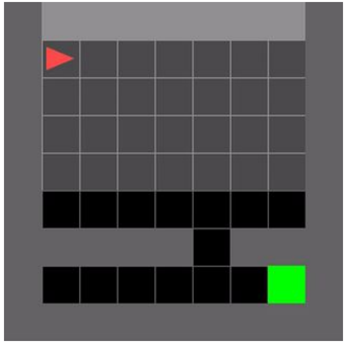


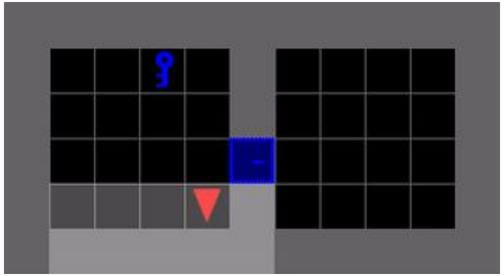


Generalize to Unseen Environments

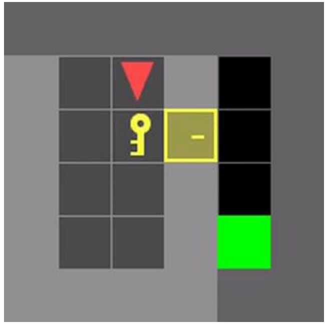
DQN

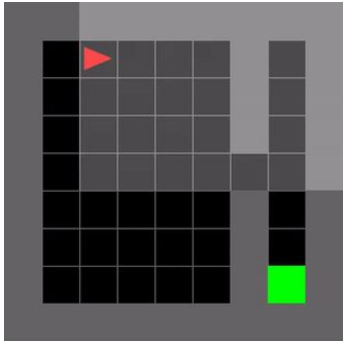


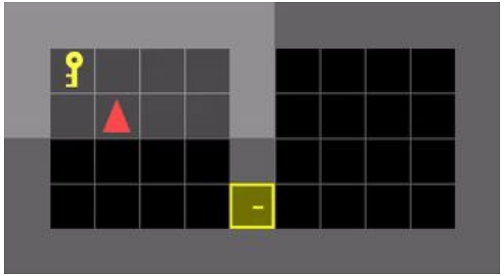




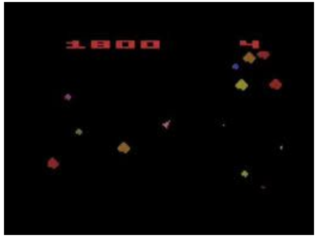
DQNReg

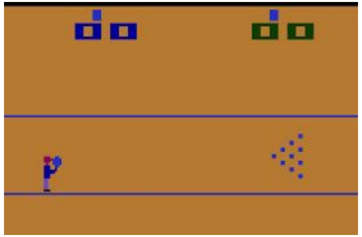


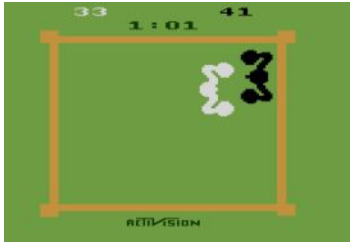




Atari Performance





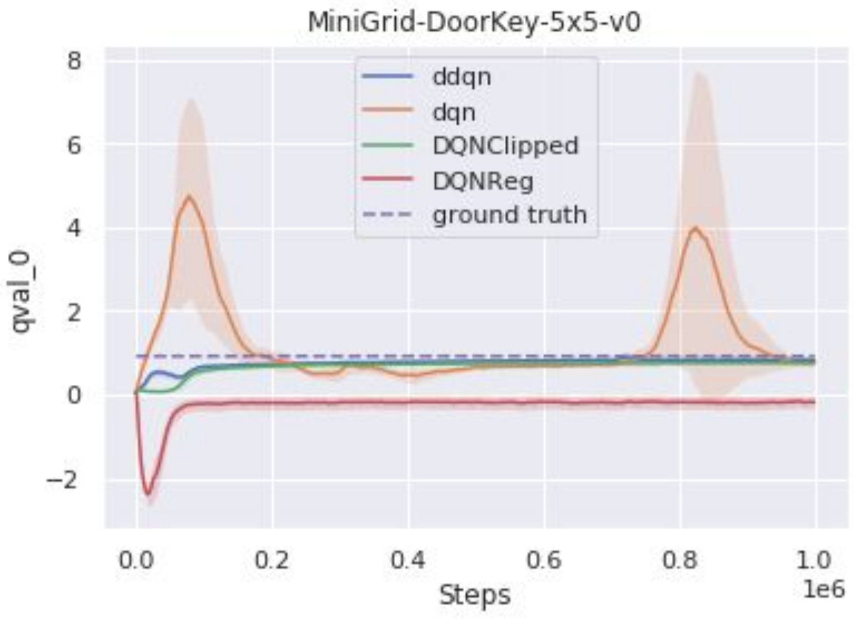


Env	DQN	DDQN	PPO	DQNReg
Asteroid	1364.5	734.7	2097.5	**2390.4**
Bowling	50.4	68.1	40.1	**80.5**
Boxing	88.0	91.6	94.6	**100.0**
RoadRunner	39544.0	44127.0	35466.0	**65516.0**

Baselines taken from reported numbers.

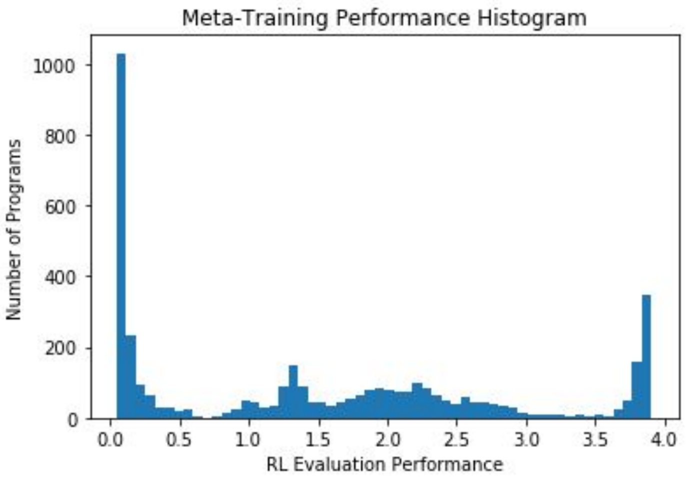
Learned algorithm (DQNReg) generalizes to Atari games when meta-training was on non-image based environments. Not tuned to Atari games.

DQNReg



DQN overestimates Q values while learned algorithms DQNClipped and DQNReg overcome this issue and underestimate Q values

Learning Convergence



Raw Equation	Simplified Equation	Score	Rank
$\delta^2 + 0.1 * Q(s_t, a_t) + r_t - (\gamma * Q_{targ} - 0.1 * Q(s_t, a_t))$	$\delta^2 + 0.2 * Q(s_t, a_t)$	3.903	2
$\delta^2 + 0.1 * Q(s_t, a_t) - \gamma + Q_{targ}$	$\delta^2 + 0.1 * Q(s_t, a_t)$	3.902	3
$\delta^2 + ((r_t + \gamma * Q_{targ} + Q(s_t, a_t)) * (\gamma - \max(\gamma, 0.1 * Q(s_t, a_t)) - \gamma * Q_{targ} - 0.1 * Q(s_t, a_t))$	NA	3.846	11146
$\delta^2 + (\delta^2 + 0.1 * Q(s_t, a_t))^2$	NA	3.65	12146
$\delta^2 + Q(s_t, a_t)$	$\delta^2 + Q(s_t, a_t)$	2.8	12446
δ^2	δ^2	2.28	13246

Top performing algorithms have similar structure

With different training environments or initialization, could find other families of models with better performance

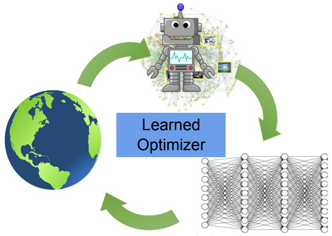
Conclusion

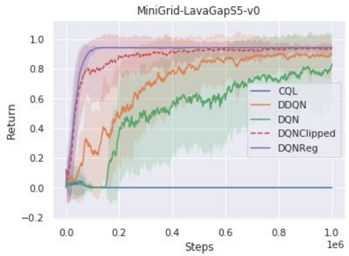
- RL algorithm as a computational graph

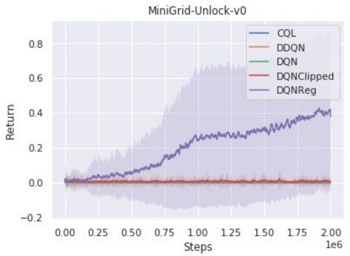
- Evolve new RL algorithms

- Learned algorithms generalize to unseen environments

reinforcement learning



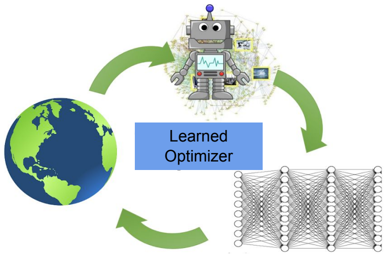




Env	DQN	DDQN	PPO	DQNReg
Asteroid	1364.5	734.7	2097.5	**2390.4**
Bowling	50.4	68.1	40.1	**80.5**
Boxing	88.0	91.6	94.6	**100.0**
RoadRunner	39544.0	44127.0	35466.0	**65516.0**

Discussion

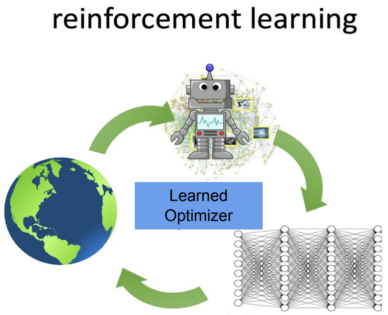
- Incorporate learned modifications into existing algorithms



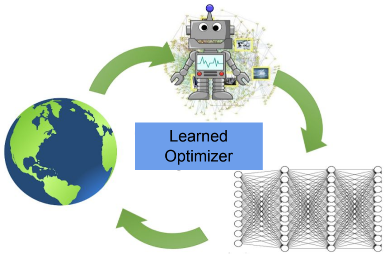
Discussion

- Incorporate learned modifications into existing algorithms

- Machine assisted algorithm development



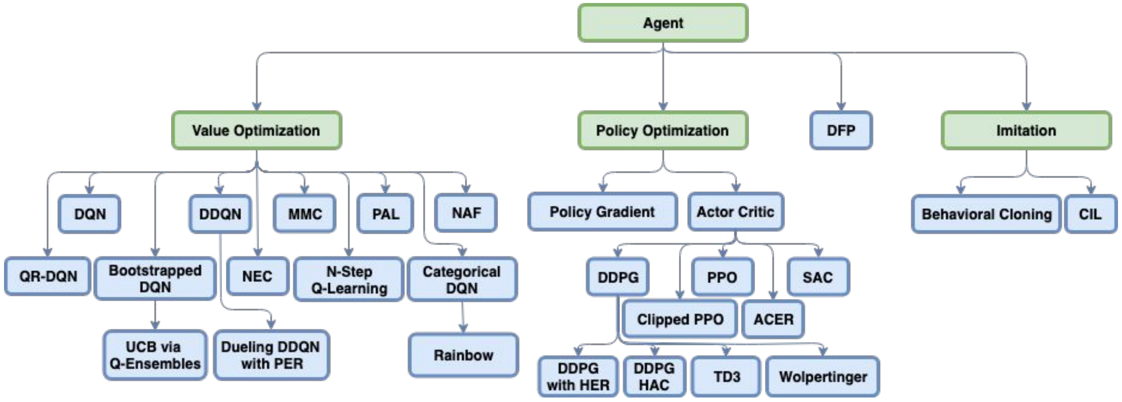
Discussion



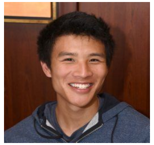
- Incorporate learned modifications into existing algorithms

- Machine assisted algorithm development

- Extend to other families: actor critic, offline RL



Thank you to collaborators!


JD Co-Reyes

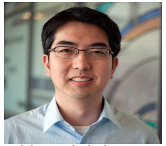

Yingjie Miao


Daiyi Peng


Esteban Real


Sergey Levine


Quoc Le


Honglak Lee


Aleksandra Faust

Questions?