AutoRL Tutorial @ AutoML

Aleksandra Faust
Senior Staff Research Scientist
faust@google.com
https://www.afaust.info/

Yingjie Miao
Senior Software Engineer
yingjiemiao@google.com

Richard Song
Research Scientist
xingyousong@google.com
https://xingyousong.github.io/

Google Brain, July 2022

Google Research

Content

Why AutoRL?

- Why does RL need automation?
- AutoRL problem formulation

AutoRL training methods and tools

What parts of RL we can automate?

What are open problems?

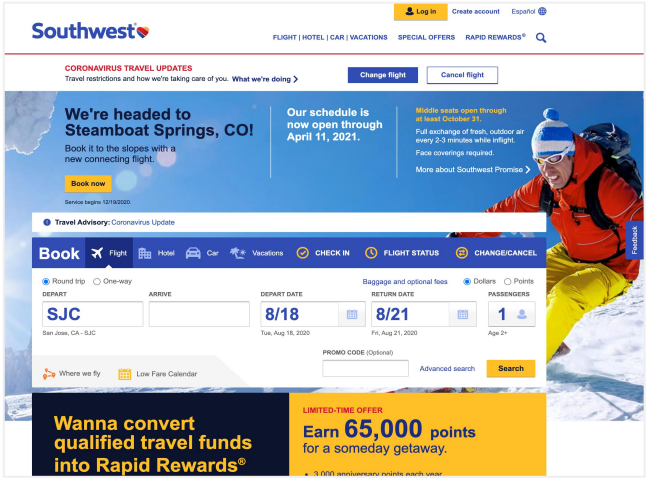
TI;Dr;

Why AutoRL?

Learn through trial and error by interacting with the environment.

RL Primer - What is RL?

Image credit: Waymo



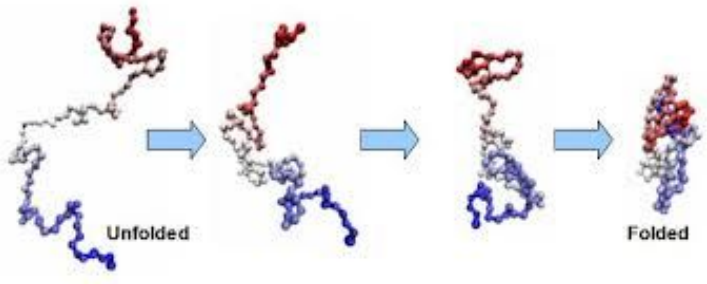


Image credit: UVM

Image credit: Loon

Google Research

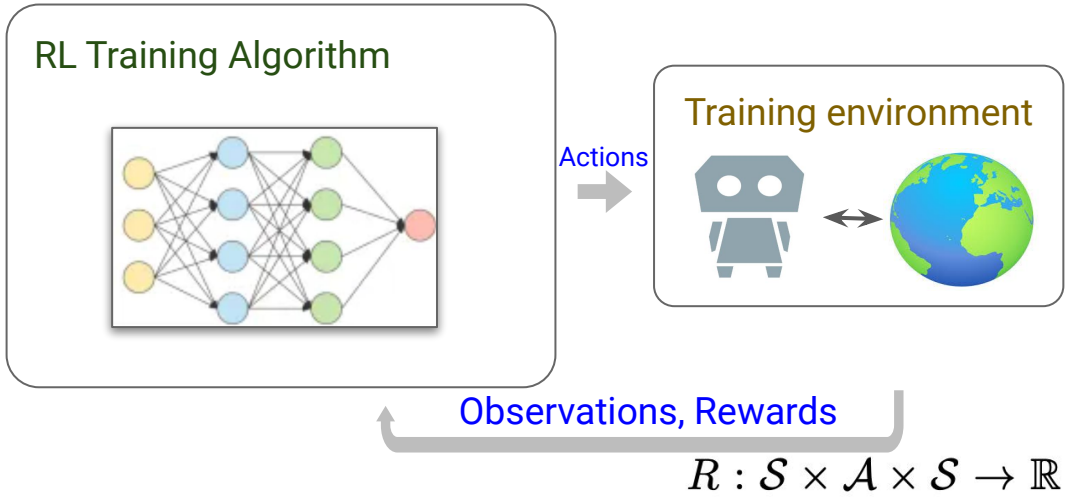
What is Reinforcement Learning?

Learned solution to a PoMDP

$(\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Omega, \rho_o, \mathcal{T}, \gamma)$

Learns a policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$

$$\max_\theta J(\theta; \zeta) \quad \text{where} \quad J(\theta; \zeta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t \geq 0} \gamma^t r_t \right]$$



Learns a policy that maximizes expected cumulative return.

Components of RL

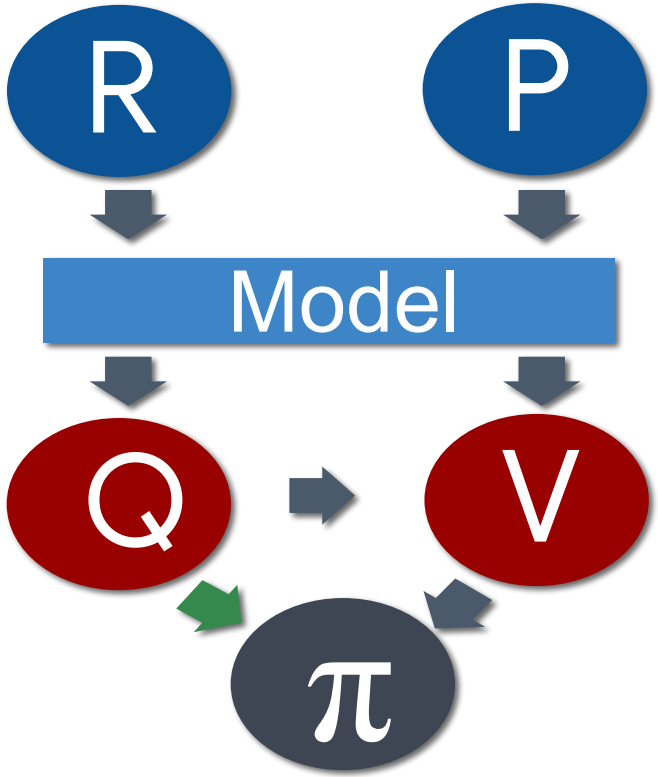
Value functions / estimators:

- State value, V: the potential payoff for the state
- State action value, Q: potential payoff for the action

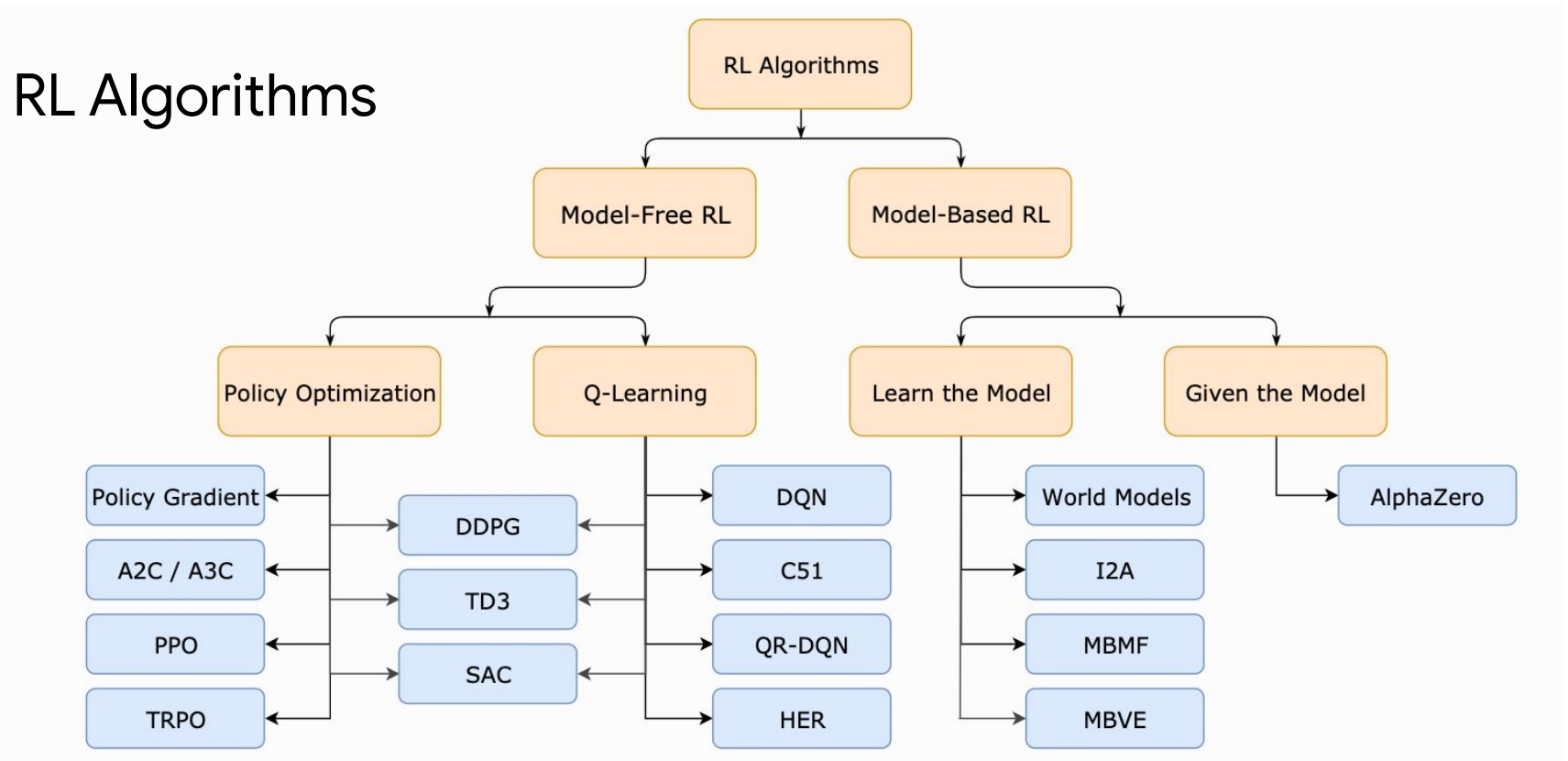
Exploration vs. Exploitation Trade off

Choice of what to learn

- Model – **model-based RL**
- Value functions – **value iteration RL**
- Policy – **policy search**, or **policy iteration**
- Value and policy - **actor critic methods.**



RL Algorithms

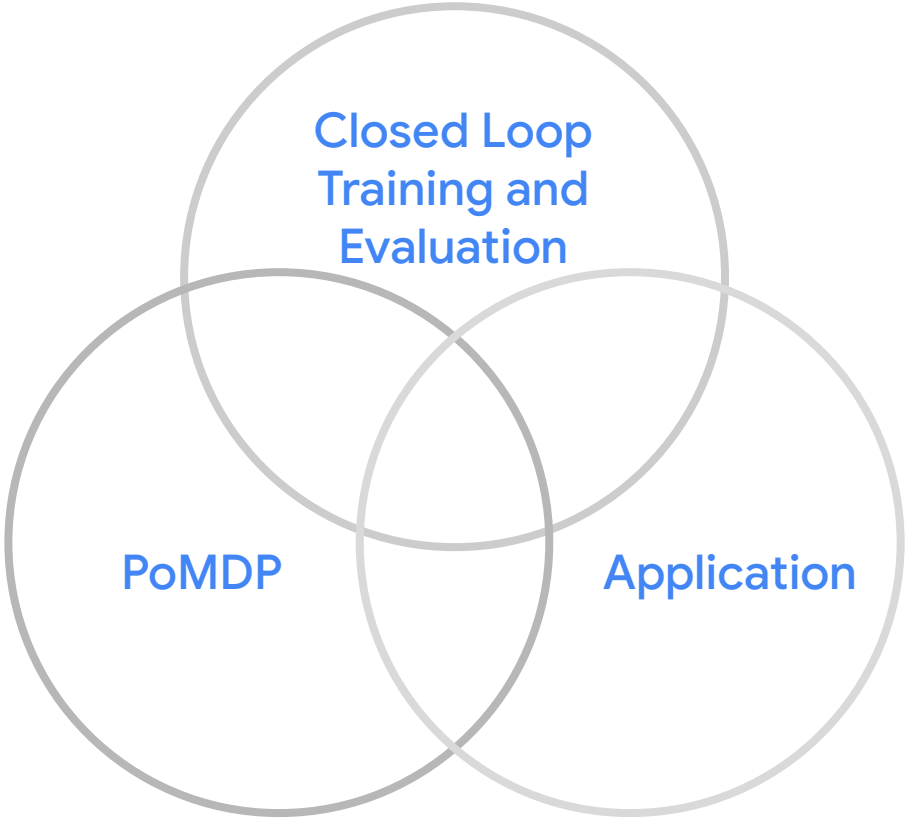


Off-policy, on-policy, offline RL

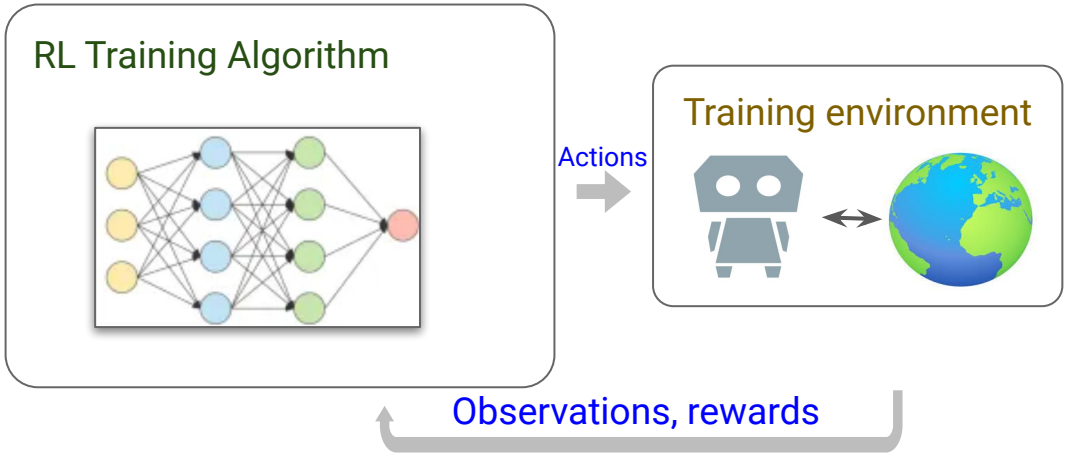
Discrete or continuous actions

Image credit: Open AI

How is RL different from SL? And why is it difficult to train?

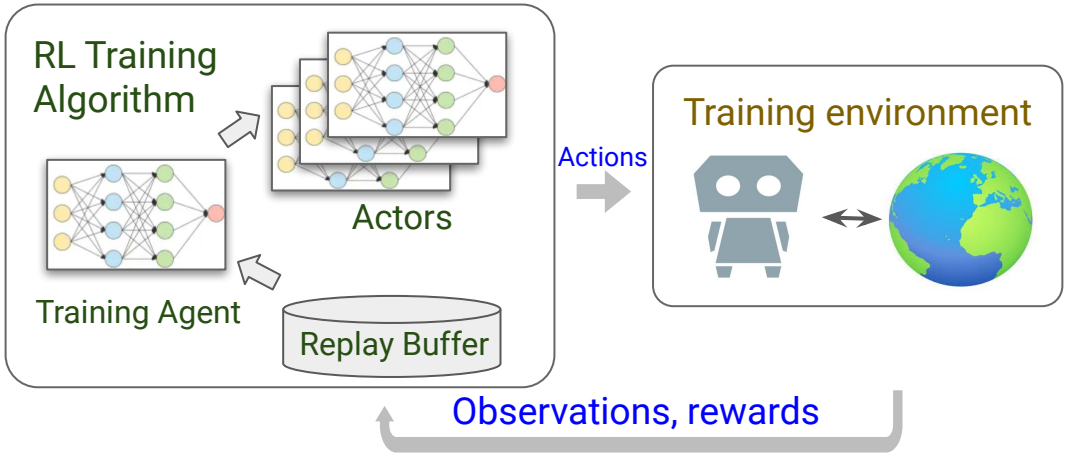


Closed loop training and distributional shift in data



Every training iteration gathers the data from changed environment.

Closed loop training and distributional shift in data

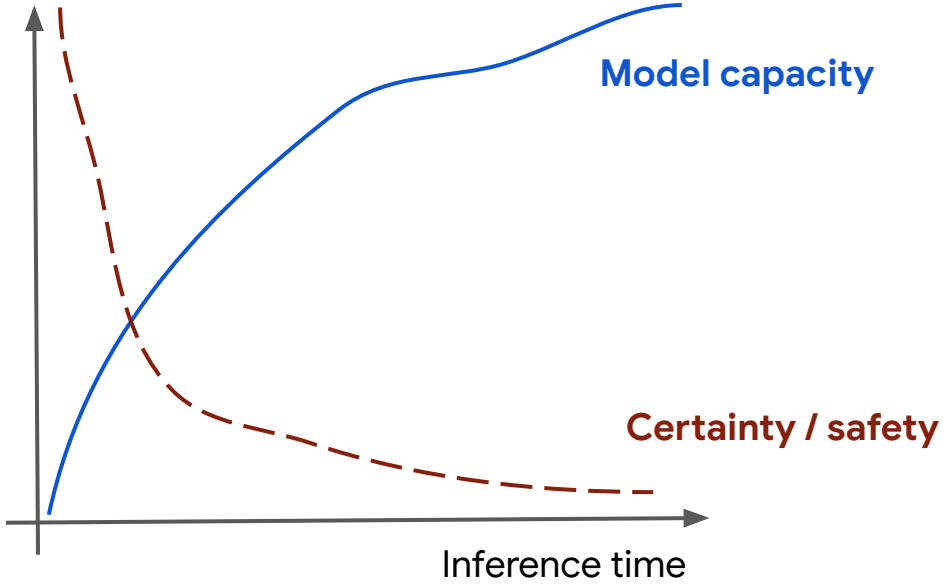


More complicated RL algorithm.

Watch the video at for an example of distributional shift at the evaluation time:

https://www.youtube.com/watch?v=xN-OWX5gKvQ&t=1s

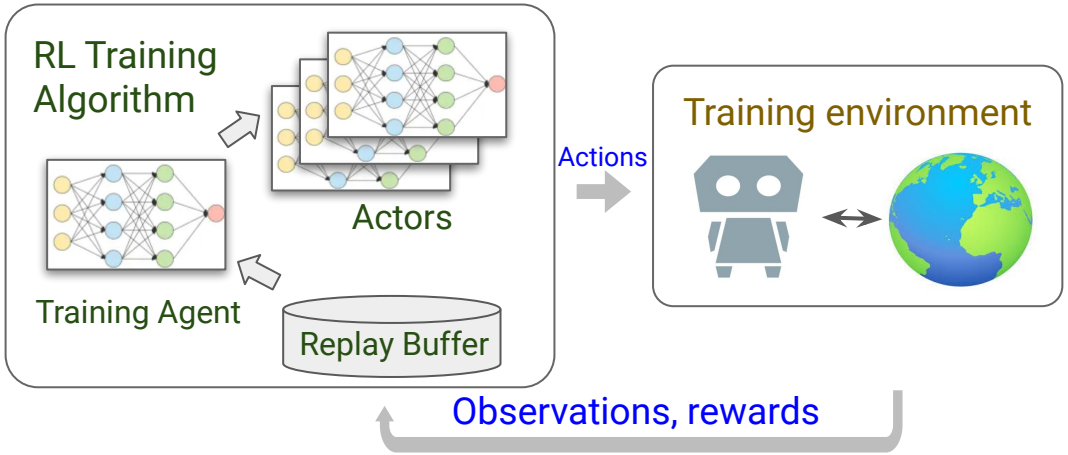
Closed loop evaluation and precision / latency trade off.



Environment distribution shift grows with the policy latency.

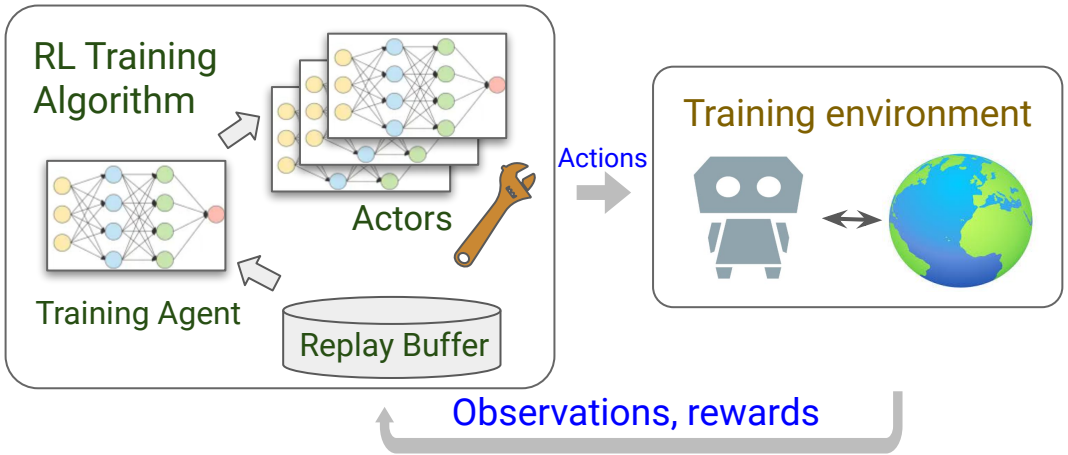
Bigger not always better. Find the sweet spot.

Function approximations and PoMDP $(\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Omega, \rho_o, \mathcal{T}, \gamma)$



PoMDP doesn't say anything about function approximations.

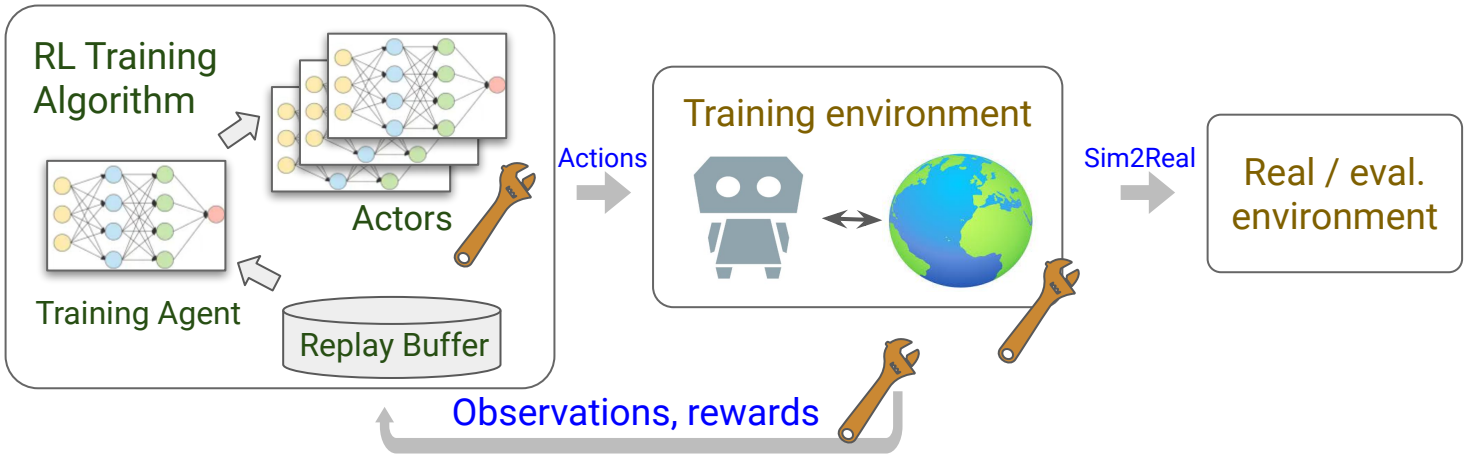
Function approximations and PoMDP $(\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Omega, \rho_o, \mathcal{T}, \gamma)$



PoMDP doesn't say anything about function approximations.

Experimentation with NN architectures.

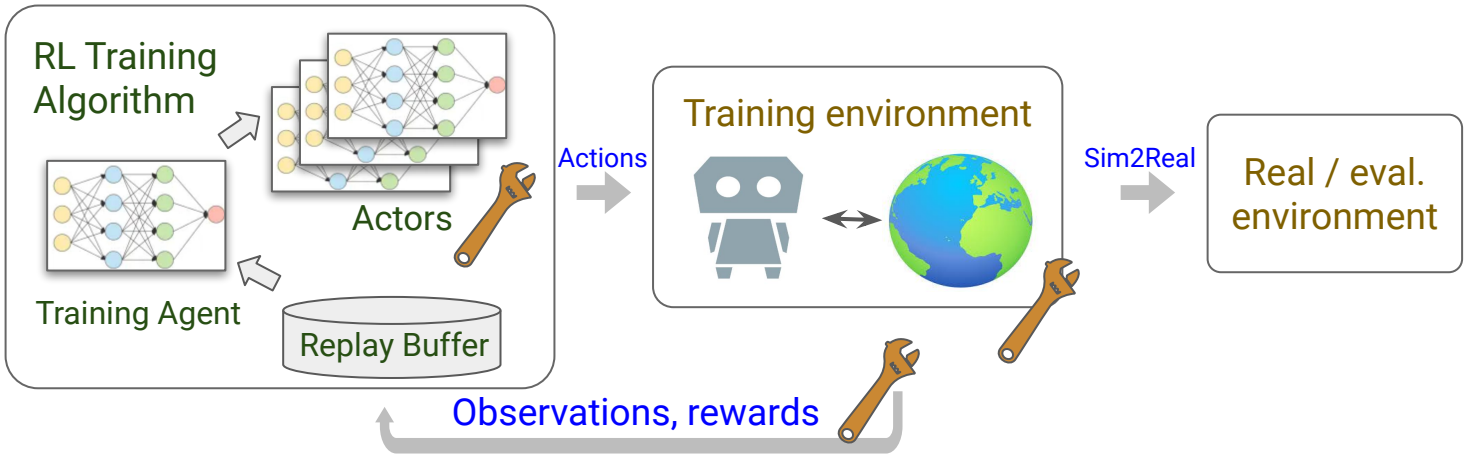
Function approximations and PoMDP $(\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Omega, \rho_o, \mathcal{T}, \gamma)$



Simulator design and Markovian property violation.

More complicated stack.

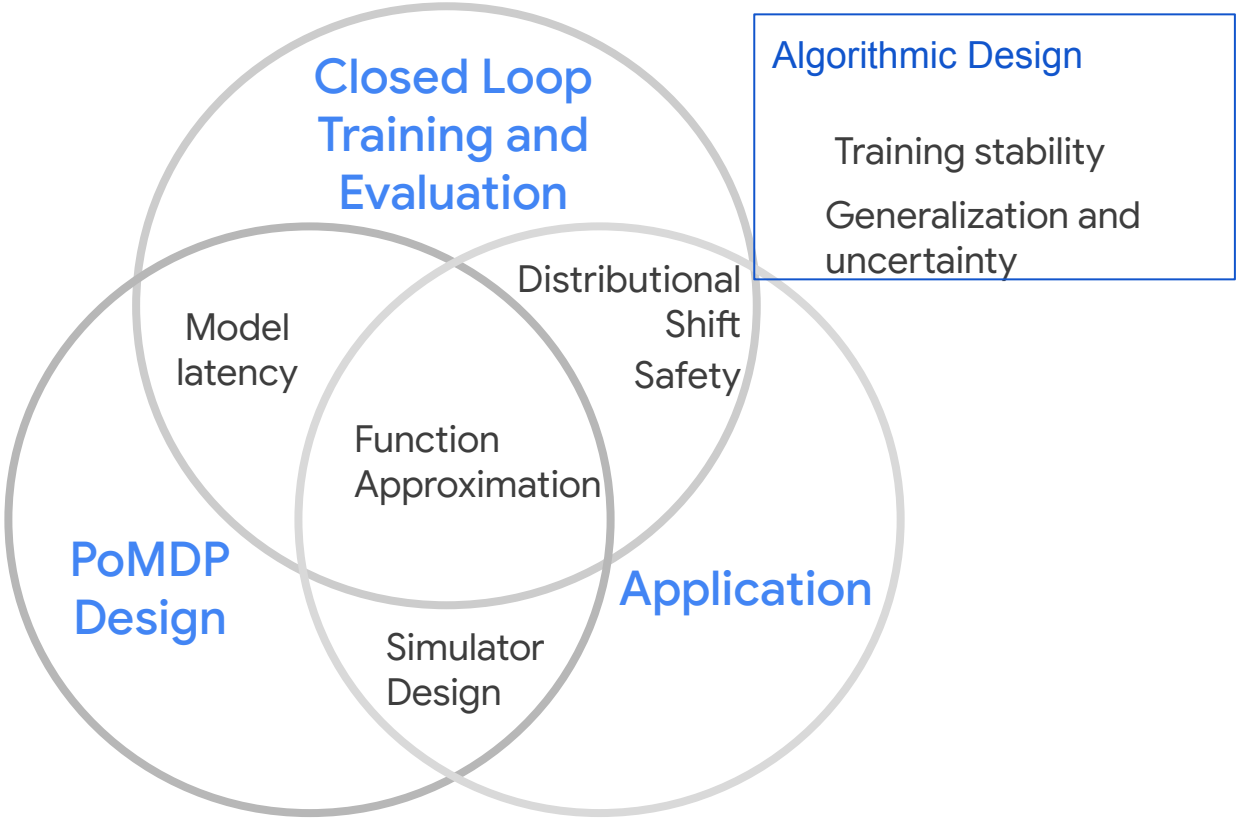
Application Implications: Safety and Generalization



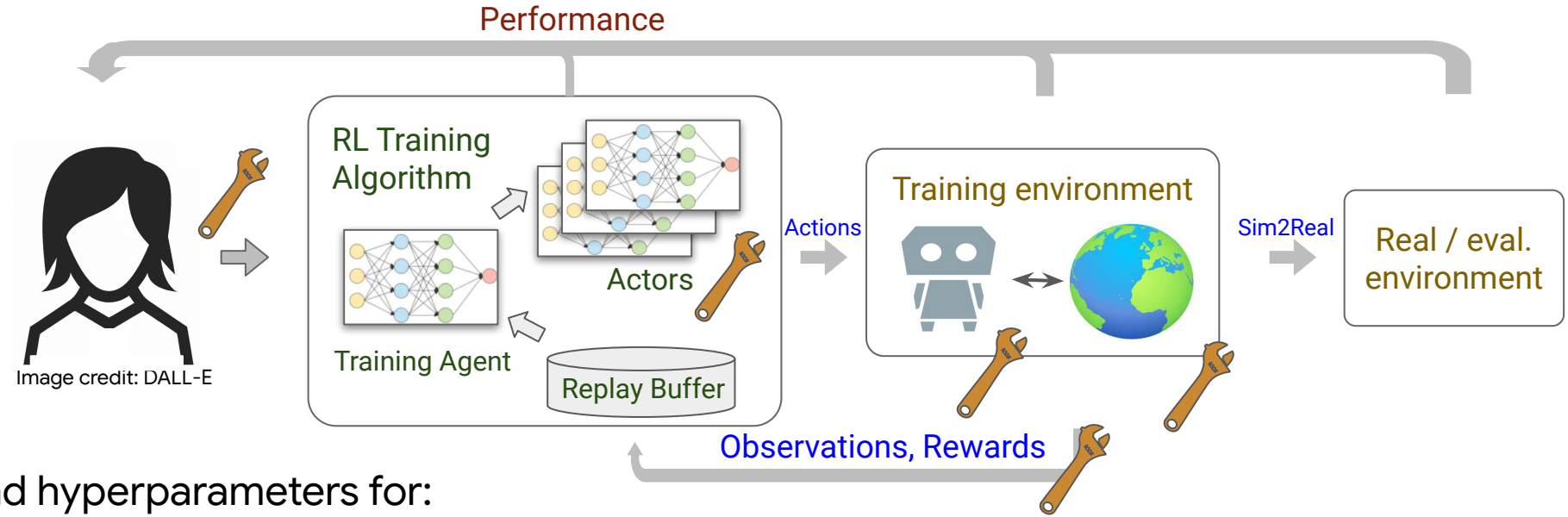
Often can cause damage, and needs to work in unstructured environments.

Training environment design and generalization methods.

How is RL different from SL? And why is it difficult to train?



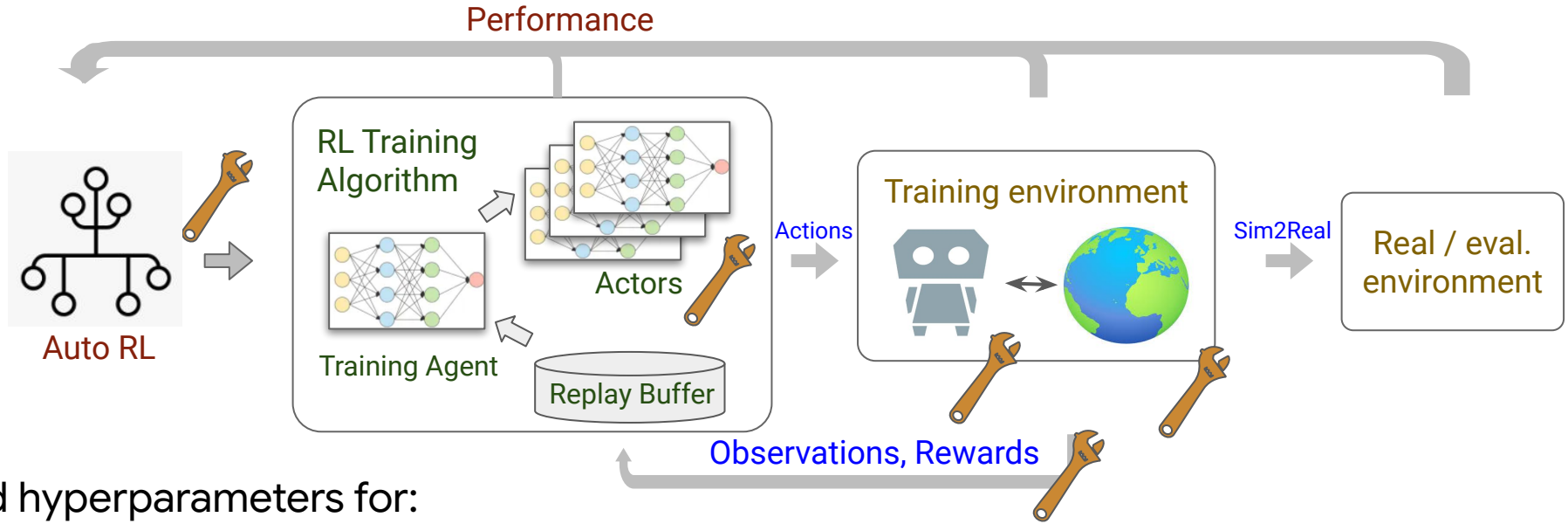
Reality of Training and Evaluating RL Agents



Find hyperparameters for:

- RL-focused components (Algorithm, NAS)
- Task definition (reward)
- Agent (observations, simulator, actions)
- Environment (curriculum)

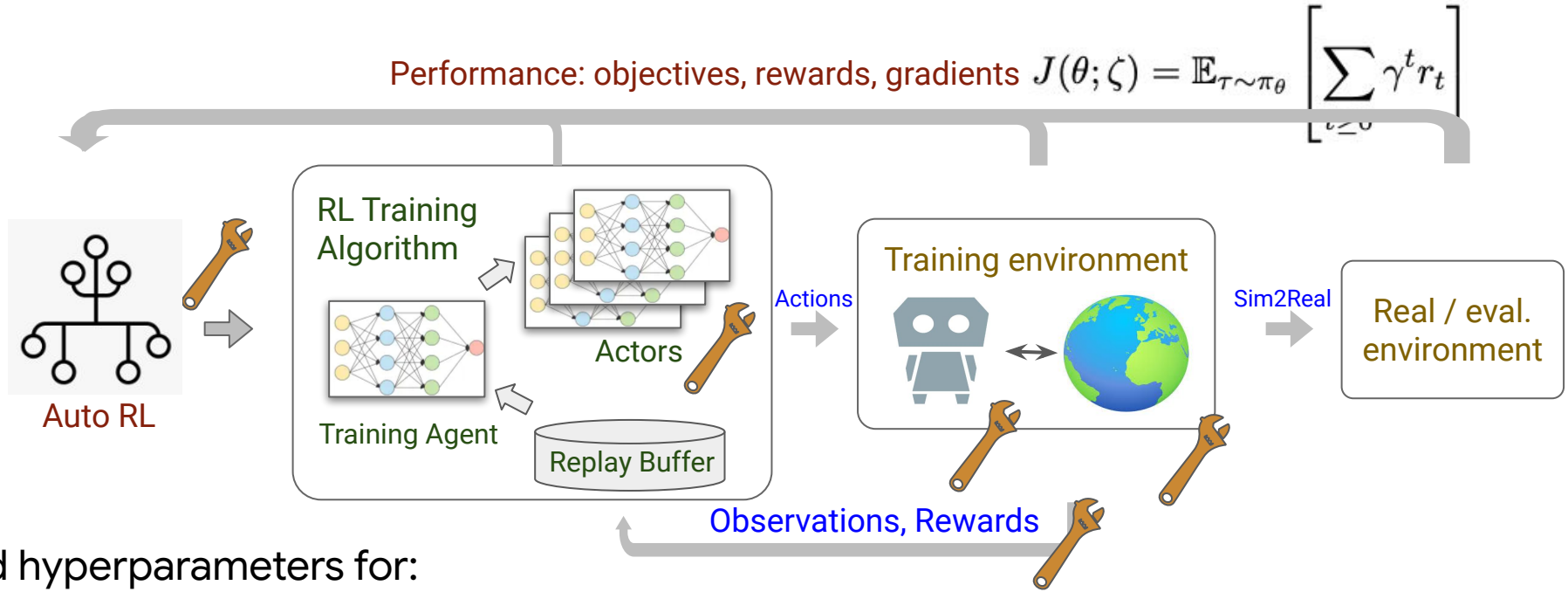
Reality of Training and Evaluating RL Agents



Find hyperparameters for:

- RL-focused components (Algorithm, NAS)
- Task definition (reward)
- Agent (observations, simulator, actions)
- Environment (curriculum)

Reality of Training and Evaluating RL Agents

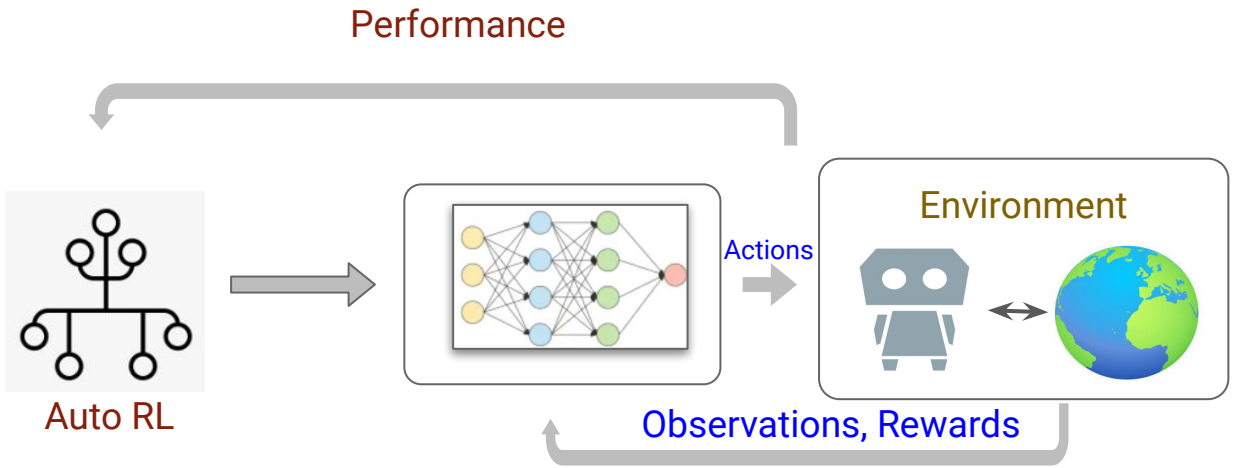


Find hyperparameters for:

- RL-focused components (Algorithm, NAS)
- Task definition (reward)
- Agent (observations, simulator, actions)
- Environment (curriculum)

$$\max_{\zeta} f(\zeta, \theta^*) \quad \text{s.t.} \quad \theta^* \in \arg\max_{\theta} J(\theta;\zeta)$$

AutoRL Problem Formulation



$$\max_{\zeta} f(\zeta, \theta^*) \qquad \theta^* \in \arg\max_{\theta} J(\theta; \zeta) \quad J(\theta; \zeta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t \geq 0} \gamma^t r_t \right]$$

Outer loop

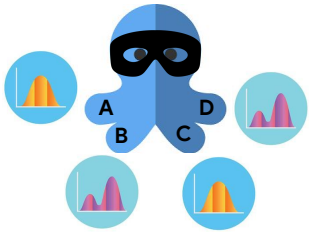
Inner loop

Bi-level optimization problem.

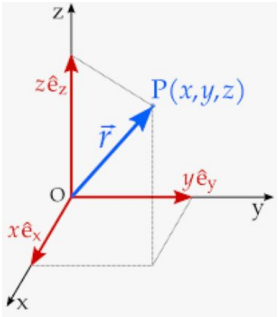
Hyperparameter Taxonomy by Type

Learning Hyperparameters
Low-parameter regime

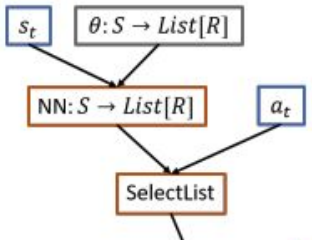
High-parameter regime



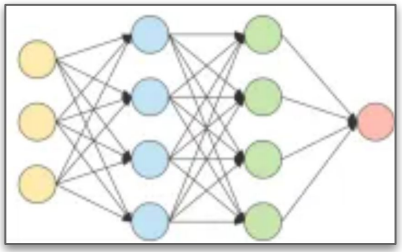
Categorical



Vector



Symbolic



Parametrized functions

Combination

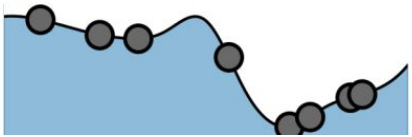
Non-linear co-dependencies?

Image credit: Colton Bishop

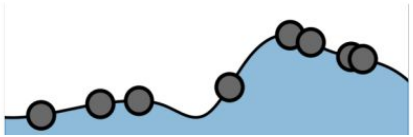
Image credit: [Co-Reyes et al., 2021]

Hyperparameter Taxonomy by Training-step Dependence

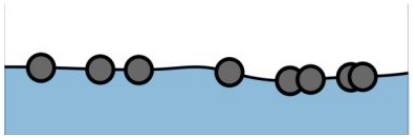
Is there an optimal hyperparameter for the entire inner training loop?



Step 1

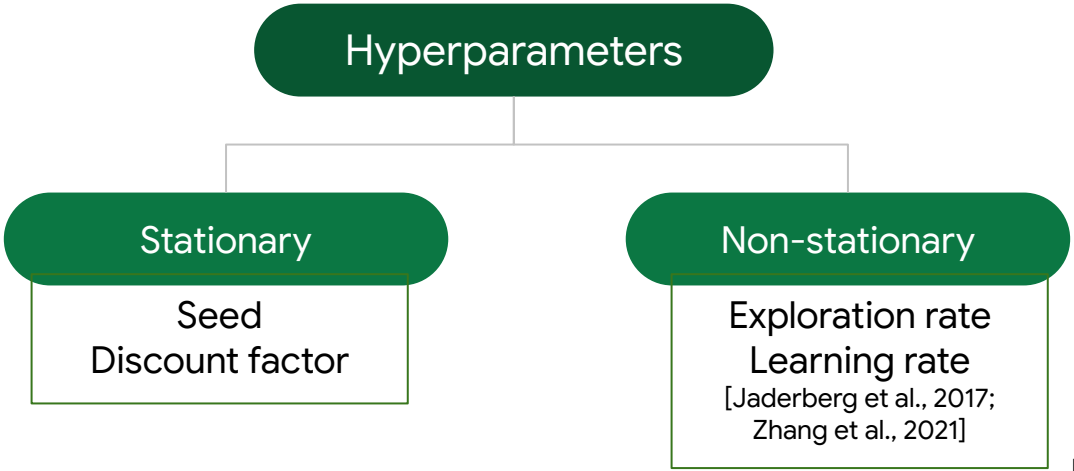


Step 1000



Step 10000

Inner loop training



Neural Network Architecture?
Discount factor?
Replay buffer size?
Replay buffer sampling policy?
Loss function?

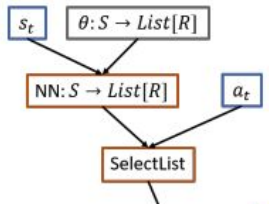
Image credit: [Parker-Holder et al., JAIR 2022]

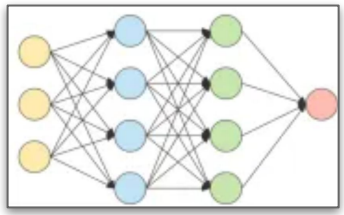
AutoRL Cost

High-parameter regime

class **stable_baselines.sac.SAC**(*policy, env, gamma=0.99, learning_rate=0.0003, buffer_size=50000, learning_starts=100, train_freq=1, batch_size=64, tau=0.005, ent_coef='auto', target_update_interval=1, gradient_steps=1, target_entropy='auto', action_noise=None, random_exploration=0.0, verbose=0, tensorboard_log=None, _init_setup_model=True, policy_kwargs=None, full_tensorboard_log=False, seed=None, n_cpu_tf_sess=None*) [source]

class **stable_baselines.common.policies.BasePolicy**(*sess, ob_space, ac_space, n_env, n_steps, n_batch, reuse=False, scale=False, obs_phs=None, add_action_ph=False*) [source]



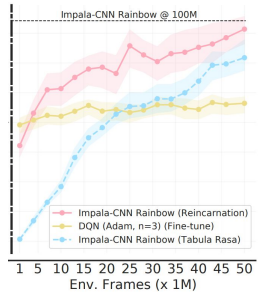


28 hyperparameters w/o NAS, algorithm, and env design

Outer loop search space extreme size

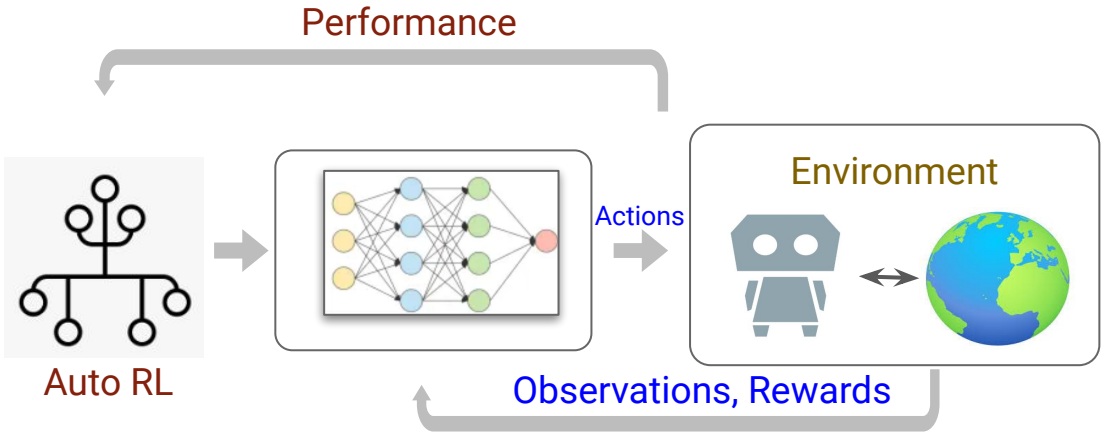
RL training often requires millions of samples and days of training.

Inner loop slow evaluation



[Agarwal et al., 2022]

AutoRL Design Considerations



Hyperparameters -- everything not in PoMDP.

- Categorical, symbolic, continuous, differentiable
- Stationary or nonstationary
- Co-dependencies.

Expensive evaluations.

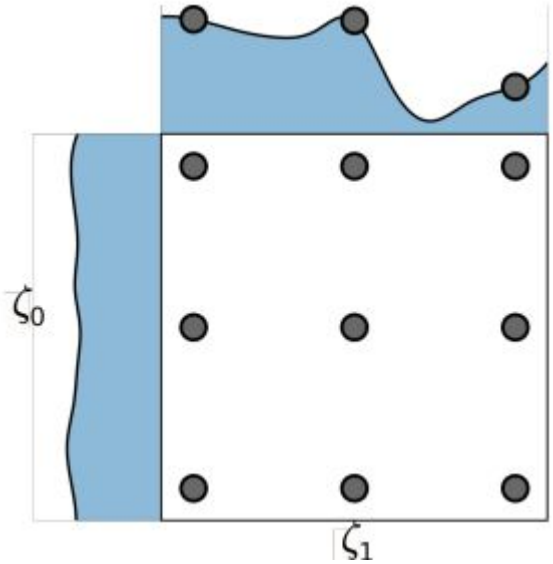
- Search in large spaces
- Long evaluations (hours or days)

AutoRL Training Methods

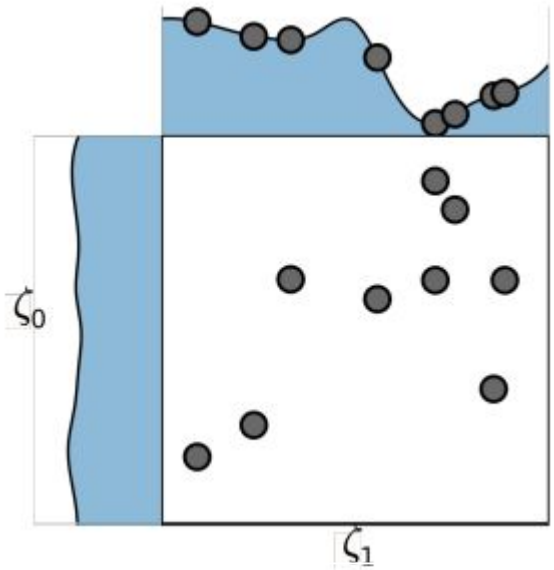
Training Methods

- Search
- Gradient-free methods
  - Evolutionary methods
- Optimization methods
  - Bayesian Optimization
  - Gradient-based (MAML)
- Multi Arm bandits
- Population-based Training

Training Methods - Search



Grid search

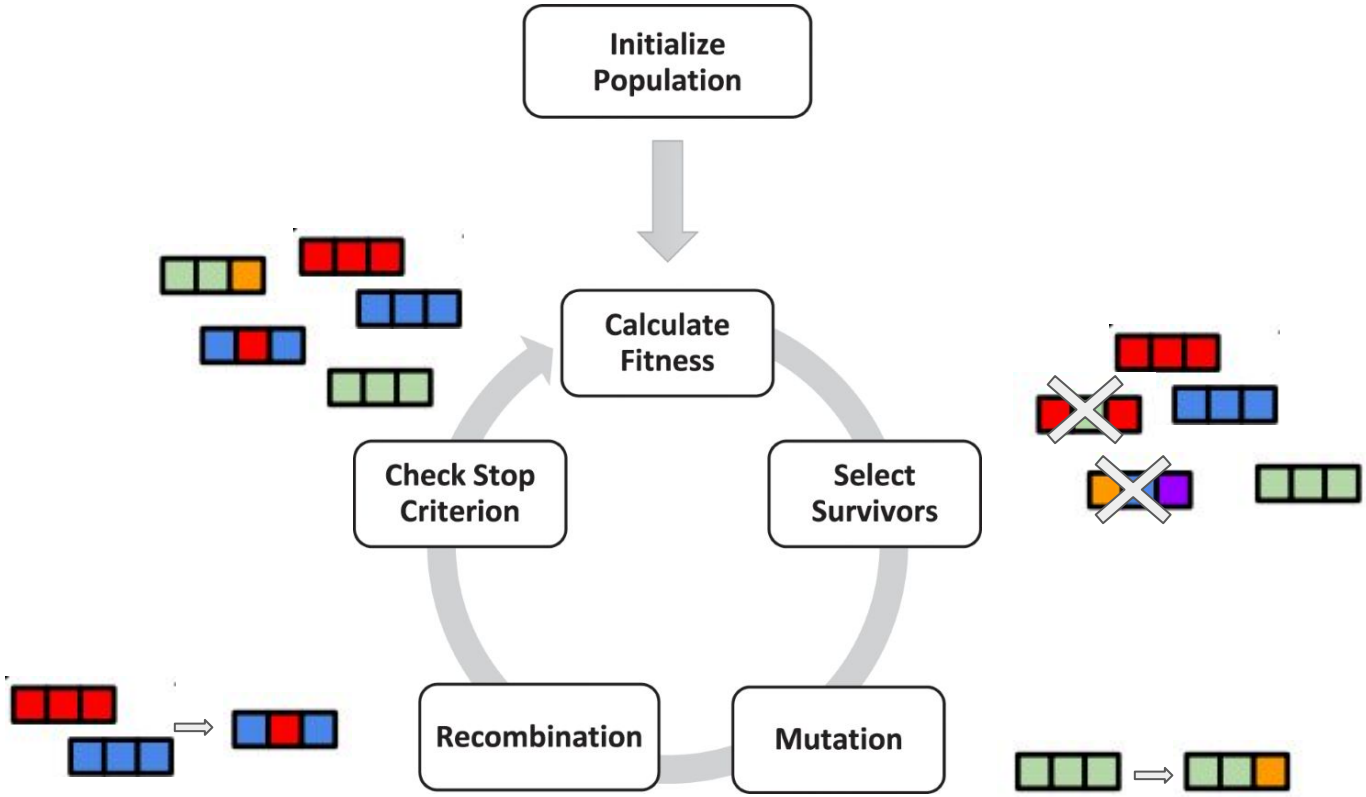


Random search

In multidimensional search spaces, random search provides more information about the objective function.

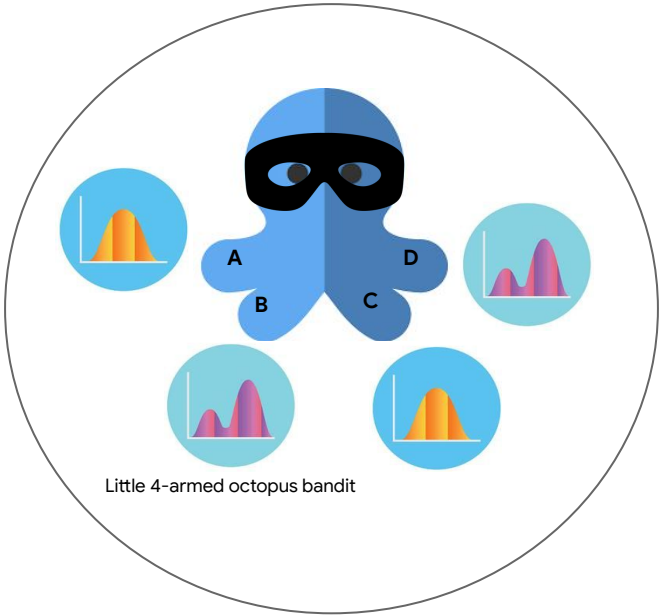
Image credit: [Parker-Holder et al., JAIR 2022]

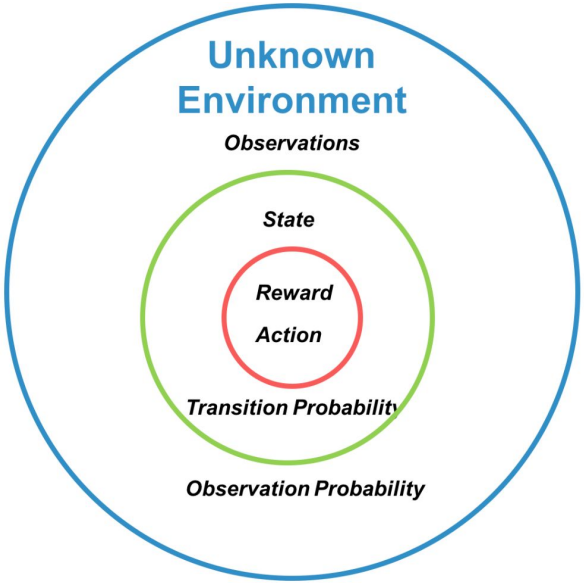
Training Methods - Evolutionary Methods



Training Methods - Multi-Arm Bandits

[Zhou, 2015]





- Bandit Problem
- MDP
- POMDP

Stateless action selection

Image credit: Bo Liu

Training Methods - Bayesian Optimization

Regressor (e.g. Gaussian Process):
Uncertainty estimates of the objective function.

Acquisition function:
Explore / exploit trade-off on f.

Next sample recommendation:
Argmax of the acquisition function.

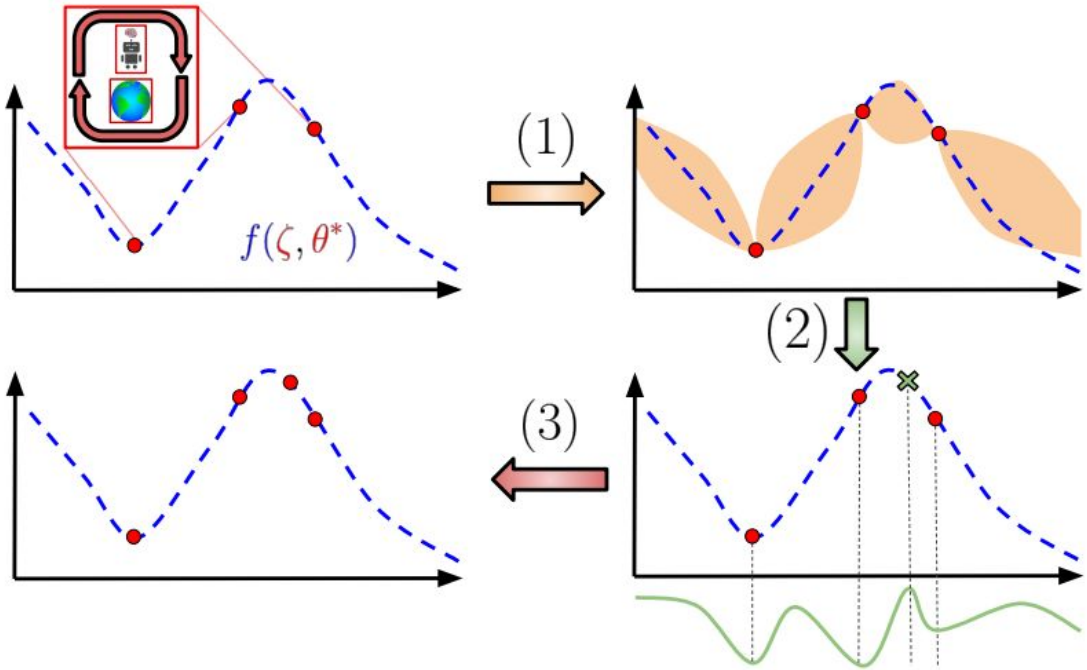


Image credit: [Parker-Holder et al., JAIR 2022]

Training Methods - Optimization - MAML

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

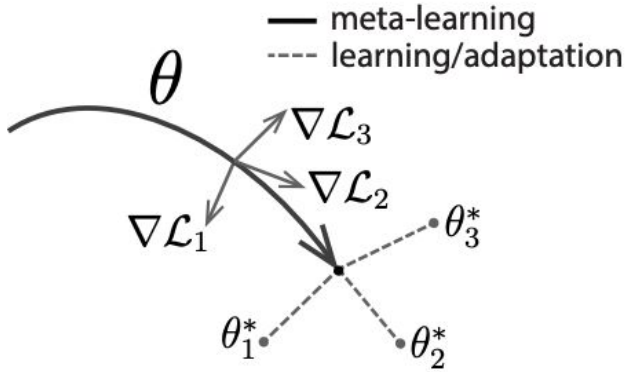
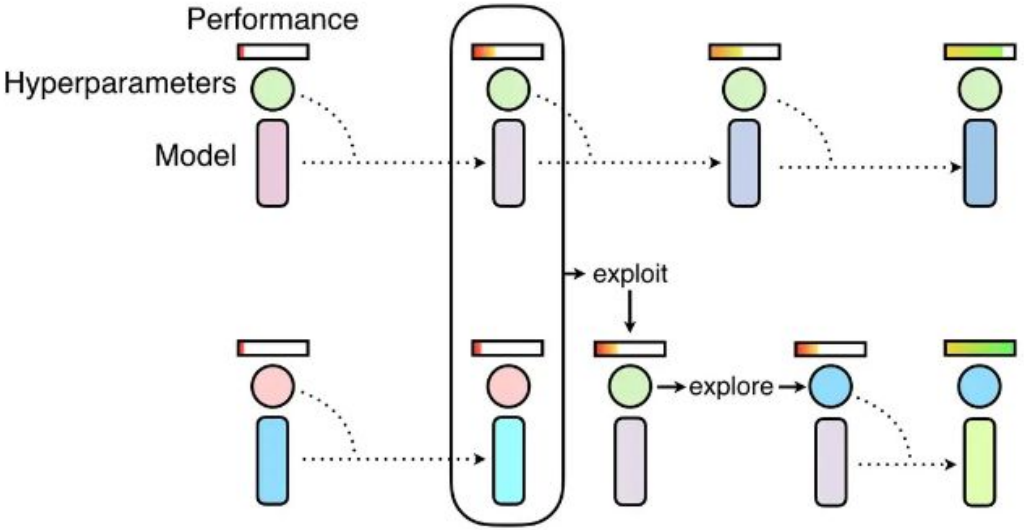


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

Credit: [Finn et al., ICML 2017]

Google Research

Training Methods - Population-based Methods



PBT jointly optimizes a population of models and their Hyperparameters. Nonstationary parameters.
[Jaderberg et al, 2017]

Image Credit: Deepmind

Training Methods - Trade-offs

Class	Algorithm properties					What is automated?
Random/Grid Search (4.1)	†††	■	⇉	✓	≐	hyperparameters, architecture, algorithm
Bayesian Optimization (4.2)	†††	■	⇉	✓	≐	hyperparameters, architecture, algorithm
Evolutionary Approaches (4.3)	†††	■	⇉	✓	≈	hyperparameters, architecture, algorithm
Meta-Gradients (4.4)	†	∇	→	●	≈	hyperparameters
Blackbox Online Tuning (4.5)	†	■	→	✓	≈	hyperparameters
Learning Algorithms (4.6)	†††	■	⇉	●	≐	algorithm
Environment Design (4.7)	†††	■	⇉	●	≈	environment

† only uses a single trial, ††† requires multiple trials

∇ requires differentiable variables, ■ works with non-differentiable hyperparameters

⇉ parallelizable → not parallelizable

✓ works for any RL algorithm, ● works for only some classes of RL algorithms

≐ static optimization, ≈ dynamic optimization

Image credit: [Parker-Holder et al., JAIR 2022]

Google Research

What to learn?

What to learn?

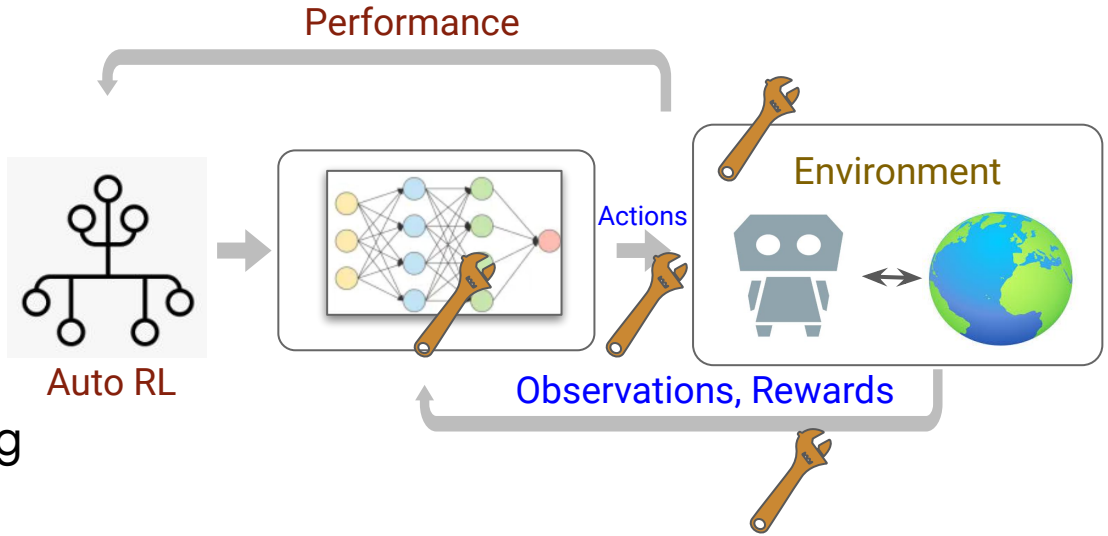
RL-focused components

- Tools for Hyperparameters
- Neural Architecture Search (NAS)
- RL Algorithm Learning

Task components: Reward

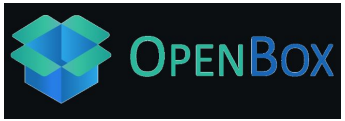
Agent-based environment learning

Environment-based learning



Hyperparameter Tuning Packages

- **Services:** Host algorithms on a server.
  - More flexible + scalable
  - Additional engineering complexity
- **Frameworks:** Execute entire optimization (both algorithm + user evaluation)
  - Convenient, full automation
  - Requires knowledge of entire evaluation pipeline
- **Libraries:** Implement blackbox optimization algorithms
  - Offer most freedom
  - Lack scalability features / limited to single machine



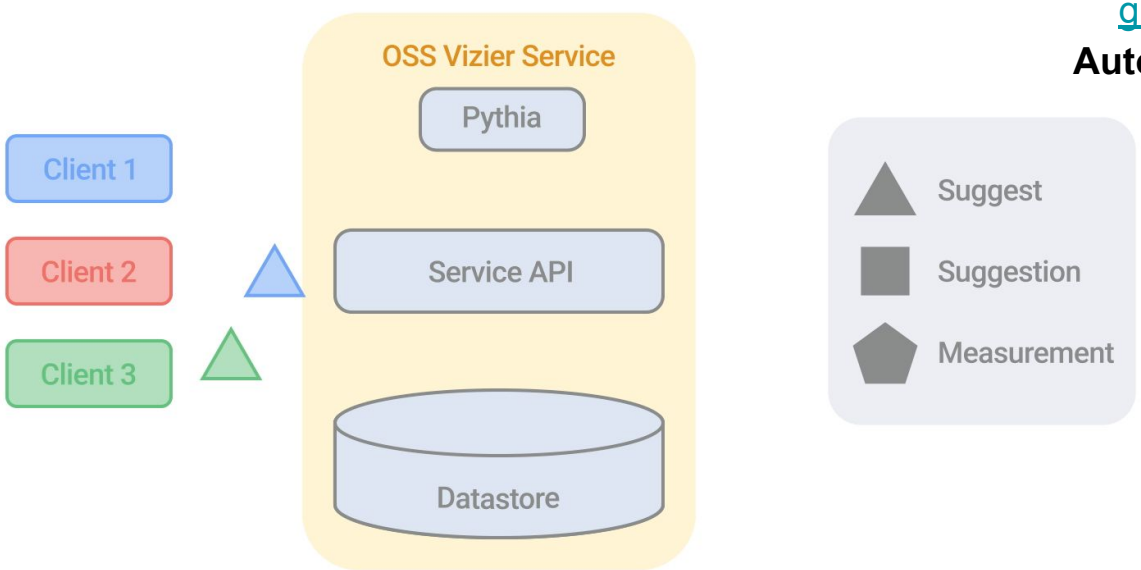
Thou shall learn hyperparameters.

Open Source Vizier Service

OSS Vizier Service: Allows distributed multi-client + reliable tuning
Supports most types of blackbox optimization

github.com/google/vizier

AutoML-Conf Systems 2022



Thou shall learn hyperparameters.

Image Credit: [Song et al., AutoML-Conf Systems 2022]

Google Research

Demo w/ OSS Vizier + Dopamine RL

(More in github.com/google/vizier/tree/main/demos)

Evaluator + Search Space + Algorithm

Client + IDs

Evaluation Loop

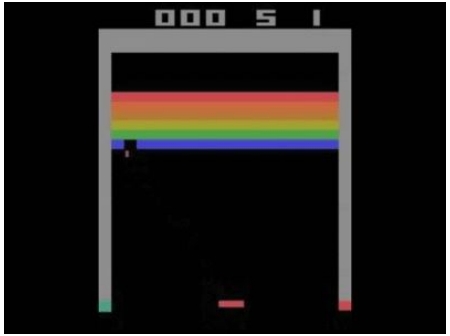
```python
from vizier.benchmarks import rl as rl_benchmarks
from vizier.service import pyvizier
from vizier.service import vizier_client

experimenter = rl_benchmarks.Atari100kExperimenter(
    game_name='Pong', agent_name='DrQ')
problem_statement = experimenter.problem_statement()
study_config = pyvizier.StudyConfig.from_problem(problem_statement)
study_config.algorithm = pyvizier.Algorithm.GAUSSIAN_PROCESS_BANDIT

client = vizier_client.create_or_load_study(
    service_endpoint='localhost:6006',
    owner_id='my_username',
    study_id='atari100k_pong',
    study_config=study_config,
    client_id='1')

for _ in range(10):
  suggestions = client.get_suggestions(suggestion_count=1)
  completed_trials = experimenter.evaluate(suggestions)
  for completed_trial in completed_trials:
    client.complete_trial(completed_trial.id,
                          completed_trial.final_measurement)
```

Vizier + Dopamine



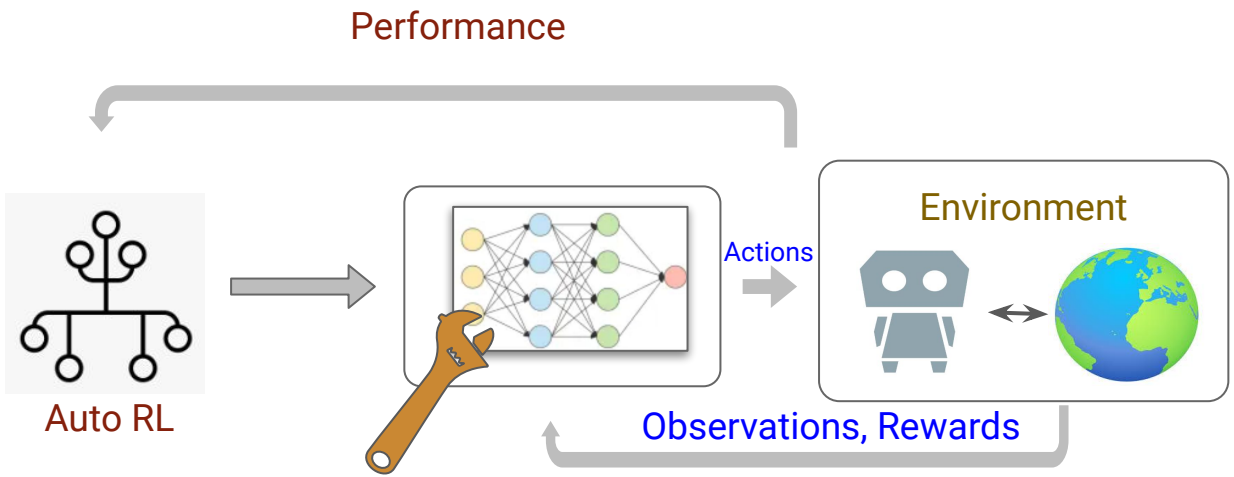
Atari100K

Thou shall learn hyperparameters.

Image Credit: [Machado et al, JAIR 2018]

Google Research

What to Learn? Neural Network Architectures



What to learn? Neural Network Architectures

Architectures matter for:

- Inference Speed
- Generalization
- Sample Complexity
- Adaptation

Credit: [Ghost Robotics]

Credit: [Yu et al. CoRL 2019]

Credit: [Cobbe et al. ICML 2020]

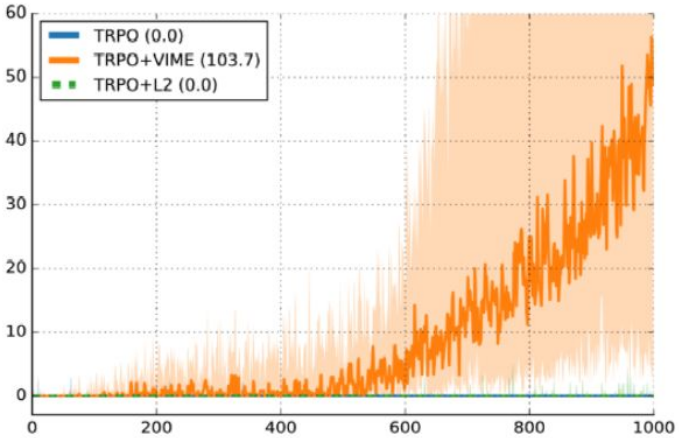
Efficient Methods in NAS for RL

Original trial-by-trial formulation: **Way too slow + expensive.**

- CIFAR-10 already takes 2000+ GPU days to do NAS! [Zoph et al, CVPR 2018]
- In RL, noisy evaluation only makes it worse!

What are some efficient methods?

Credit: Depositphotos (Stock)



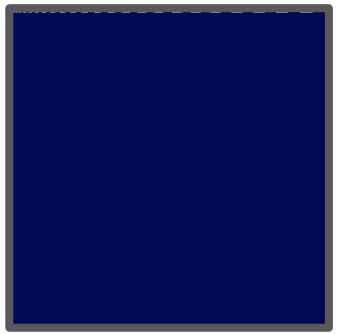
Credit: [Houthooft, NeurIPS 2016]

Low Parameter Regime

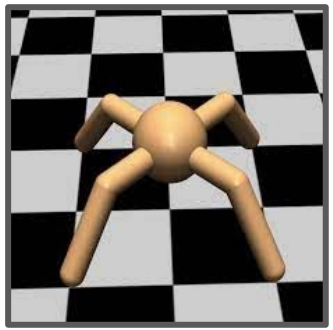
Low Parameter Regime (~10K weights): Optimize architecture + weights simultaneously

- Evolutionary Methods
  - NEAT [Stanley + Miikkulainen, EC 2002]
  - Regularized Evolution [Real et al., AAAI 2019]
  - ES-ENAS [Song et al, 2022]

Environments

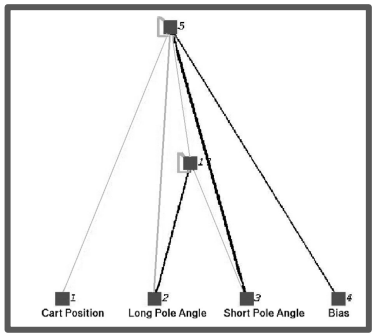


Cartpole

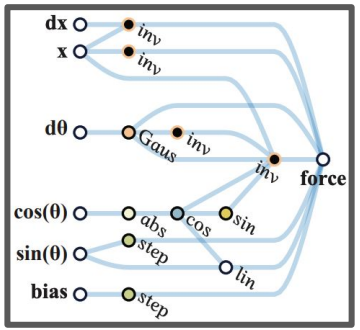


Mujoco

Architectures



Stanley + Miikkulainen, Evolutionary Computation 2002



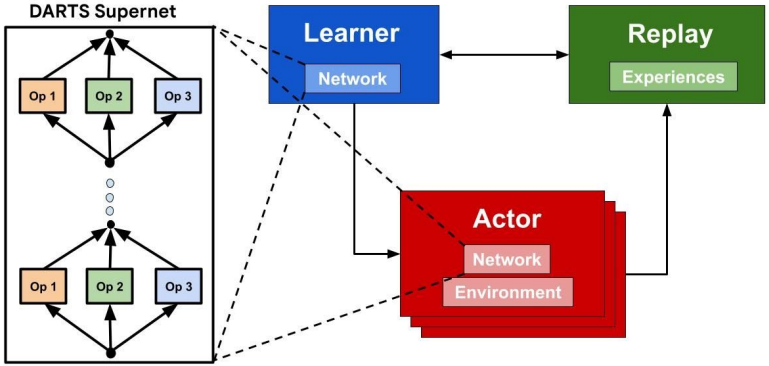
Gaier + Ha, NeurIPS 2019

Caveat: Zero-order methods suffer under too many weight parameters.

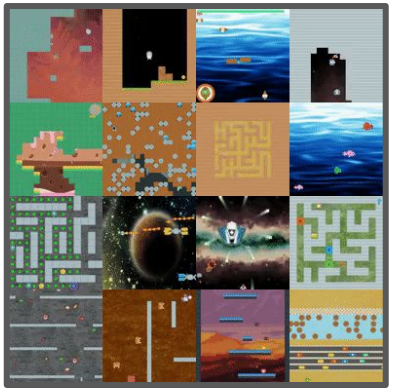
High Parameter Regime: DARTS

High Parameter Regime (1M+ weights): Gradients for weight-training

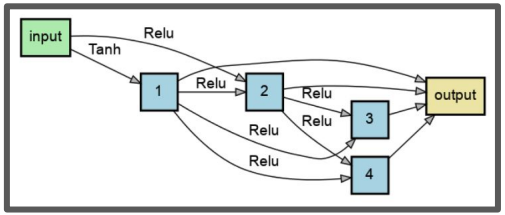
- DARTS [Liu et al, ICLR 2019] used for RL in [Miao et al, AutoML-Conf 2022]: **Minimally invasive!**



Credit: [Miao et al. AutoML-Conf 2022]



Credit: [Cobbe et al. ICML 2020]



Credit: [Miao et al. AutoML-Conf 2022]

```
#feature_network = base_policies.make_impala_cnn_network(depths=[16, 32, 32])
feature_network = darts_policies.DartsIMPALACNN(depths=[16, 32, 32])
```

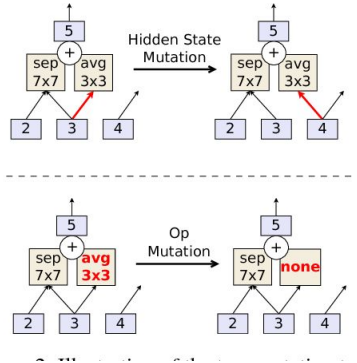
github.com/google/brain_autorl/tree/main/rl_darts

Google Research

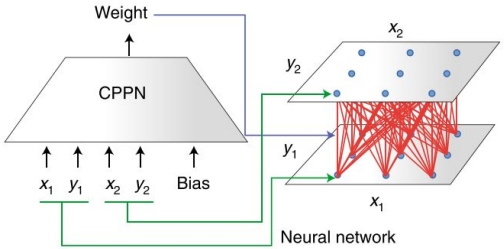
High Parameter Regime (Open Questions)

Possible Methods for NAS in RL? Little work so far.

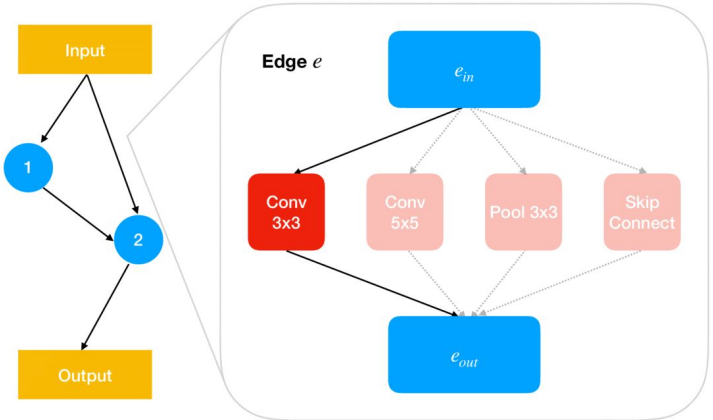
- Blackbox Optimization? **How to deal w/ strong noise?**
  - Regularized Evolution [Real et al, AAAI 2019]?
  - BayesOpt?
- Other weight sharing methods? **How to make simple?**
  - ENAS [Pham et al, ICML 2018]?
- Others? **How to scale to modern DL?**
  - HyperNEAT [Stanley et al, Artificial Life 2009]?



Credit: [Real et al, AAAI 2019]

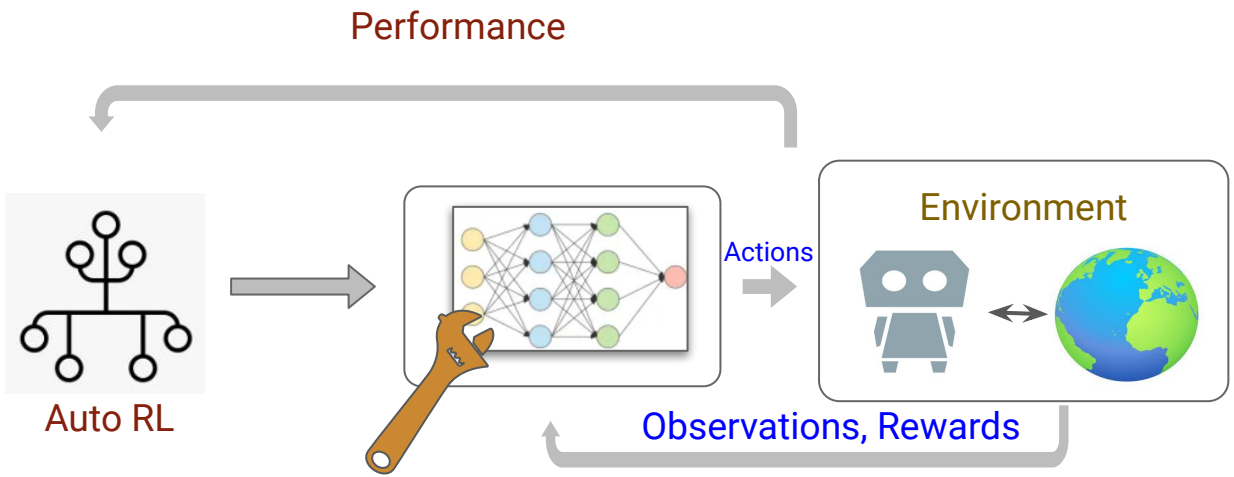


Credit: Roos, Blogpost on HyperNEAT, 2020



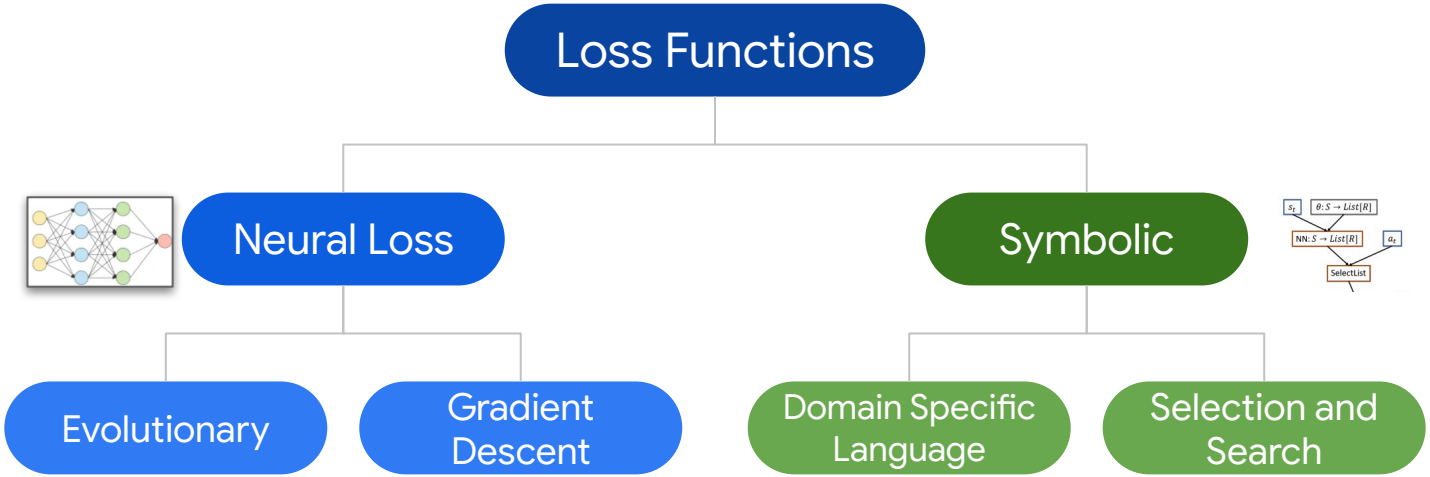
Credit: CMU ML Blog, 2020

What to Learn? RL Loss Functions



Main idea: RL loss function, $\mathcal{L}(\theta; \zeta)$ is a parameterized object.

What are we learning? Algorithms!



Search space millions of parameters and ~10^{100} programs [Garau-Luis et al., 2022]

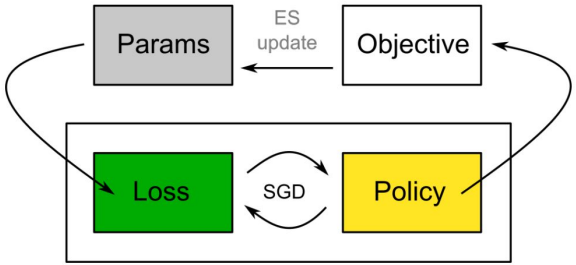
Careful design + pruning + optimization.

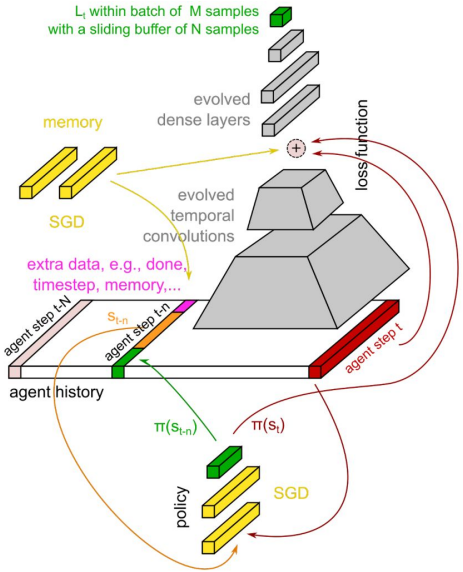
What are we learning?

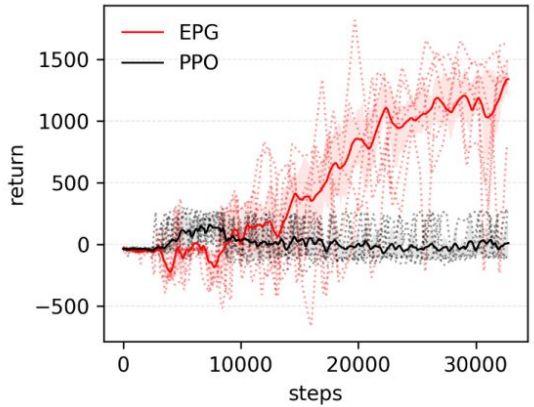
Loss Functions | Neural Loss | Evolutionary

[Houthooft et al., 2018]

Loss function adaptive to environment and agent history.







Loss function: temporal convolutions NN.

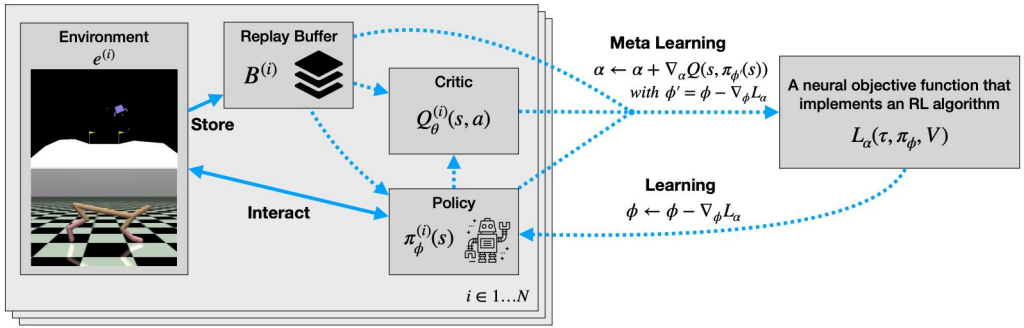
Faster training + generalization to out of distribution data.

Google Research

What are we learning?

Loss Functions | Neural Loss | Gradient Descent

[Bechtle et al. 2021]
[Kirsch et al., 2020]
[Oh et al., 2020]

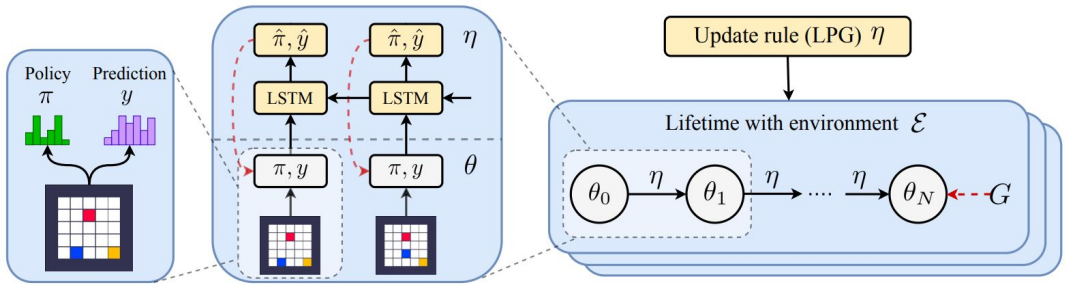


MetaGenRL: population of agents train a single meta-objective
[Kirsch et al., 2020]

Generalize to new different domains.

Require Jacobian vector product computation. Readily available.

Some cross domain generalization.



Learned Policy Gradient: what and how to predict
[Oh et al., 2020]

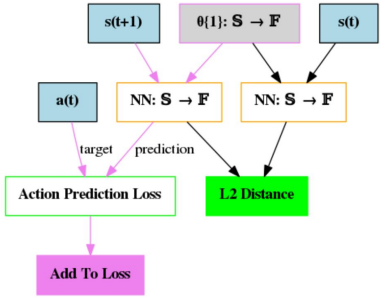
Google Research

What are we learning?

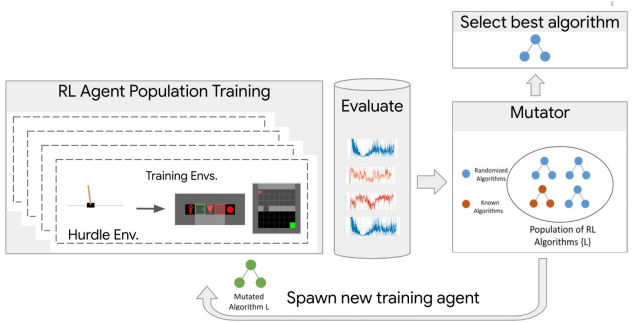
Loss Functions | Symbolic | Symbolic Loss Functions

[Laroche and Féraud 2018]

Loss function $\mathcal{L}(\theta; \zeta)$ defined with a domain specific language (DSL).



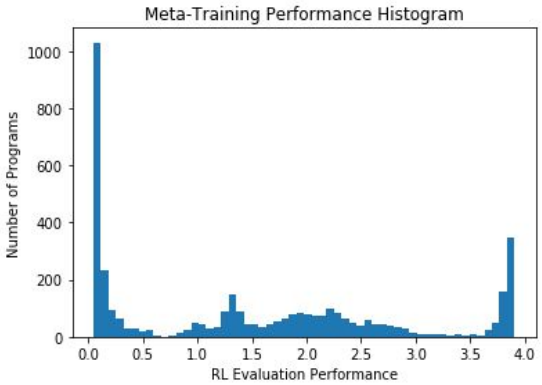
DSL for curiosity module trained w/ regressor. Search space combines NN, buffers, custom loss, etc. [Alet et al. 2020]



Loss function as a DAG. Lots of tricks due to size of search space. [Co-Reyes et al. 2021]

Interpretable loss functions.

First time large databases of loss functions.
github.com/jcoreyes/evolvingrl

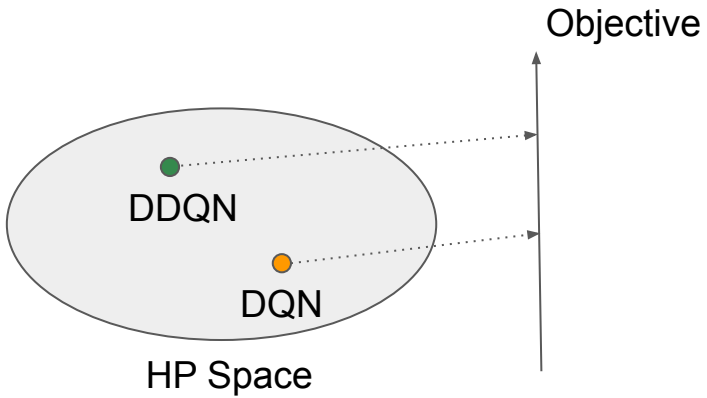


RL algorithm performance histogram.

Google Research

What are we learning? Algorithms!

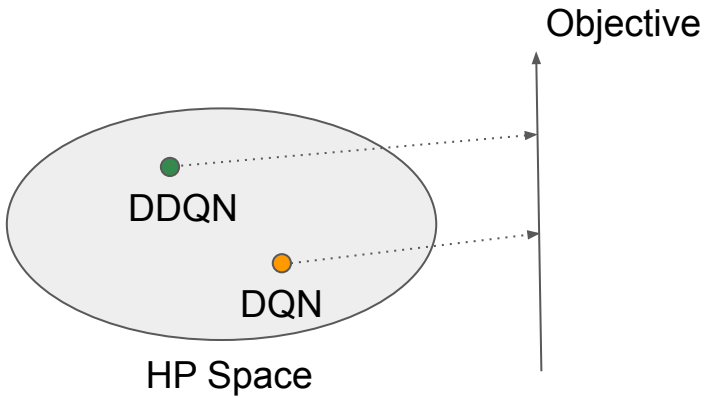
Main idea: View RL algorithms (e.g. DQN) as points in a hyperparameter space. To discover algorithms is to do HP tuning in that space.



What are we learning? Algorithms!

Main idea: View RL algorithms (e.g. DQN) as points in a hyperparameter space. To discover algorithms is to do HP tuning in that space.

Result: Co-Reyes et al. (2021) discovered new value-based RL algorithms that outperform baselines. These algorithms are interpretable, transferable, and implementable.



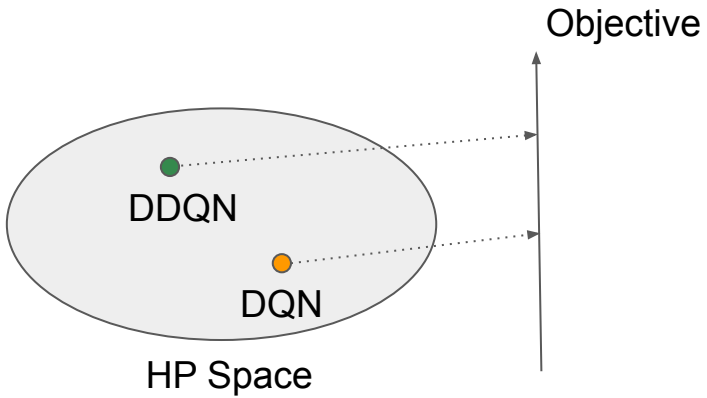
What are we learning? Algorithms!

Main idea: View RL algorithms (e.g. DQN) as points in a hyperparameter space. To discover algorithms is to do HP tuning in that space.

Result: Co-Reyes et al. (2021) discovered new value-based RL algorithms that outperform baselines. These algorithms are interpretable, transferable, and implementable.

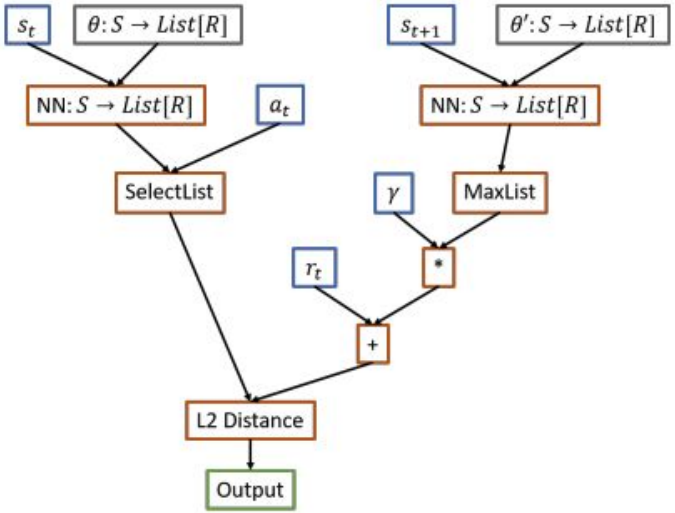
In this Tutorial:

- Algorithm representation and search space design
- Optimization method and tricks
- Code and colab example



Algorithm representation and the search space

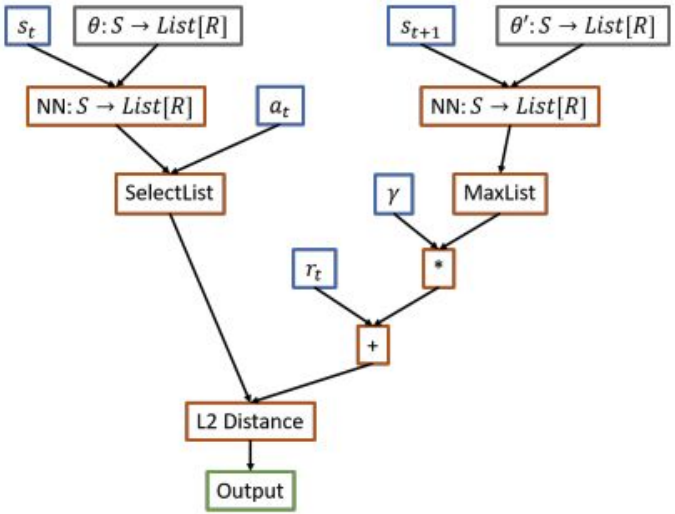
Algorithm (loss function) as a DAG



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

Algorithm representation and the search space

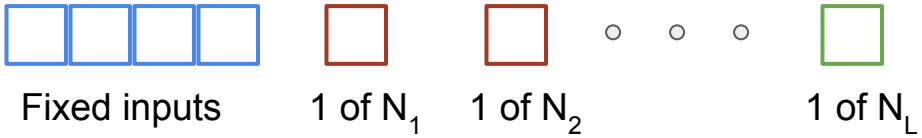
Algorithm (loss function) as a DAG



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

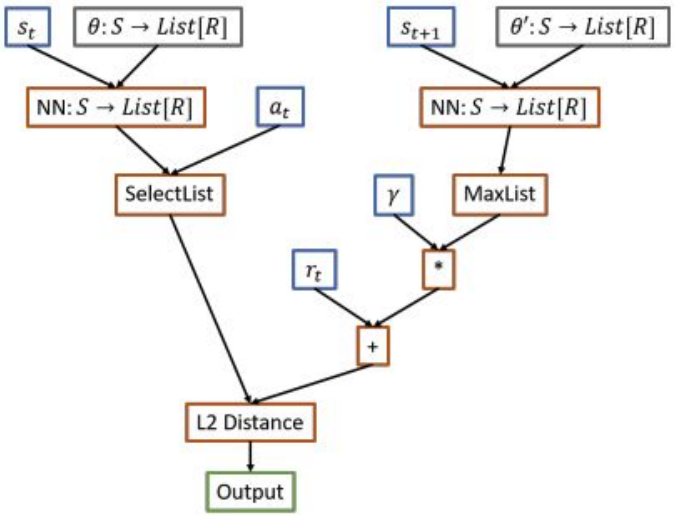
How to turn this into a HP and vice versa?

Each DAG can be viewed as **a sequence of categorical HPs**, one for each intermediate node (and the output node).



Algorithm representation and the search space

Algorithm (loss function) as a DAG



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

How to turn this into a HP and vice versa?

Each DAG can be viewed as **a sequence of categorical HPs**, one for each intermediate node (and the output node).

For each intermediate node: which op to use (+, -, …)?
Which *previous* nodes as inputs?

For the output node: which intermediate node to output?

Encoding-decoding logic is necessary for mapping between HPs and DAGs.

Search space detail

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)		$\mathbb{R}$

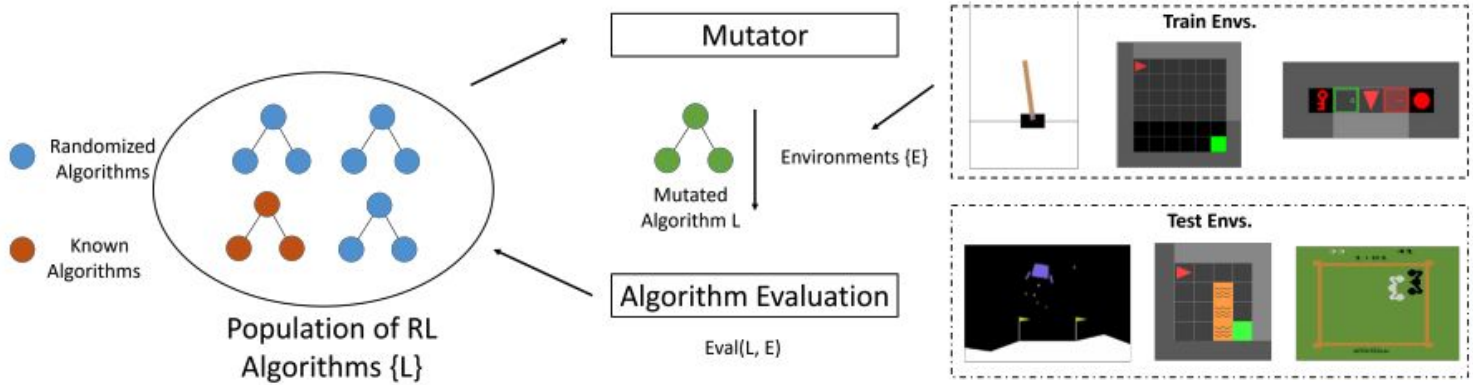
Operations:

- Covers many known value-based RL algorithms.
- Typed inputs/outputs help to prune the search space.
- In practice, need to set a max size of the DAG (~20 worked).

This completes the specification of the HP search space.
Next:

- Define an optimization objective.
- Plug in your favorite HP tuning algorithm!

Training overview



Optimization objective: sum of normalized returns on meta-training environments

Optimization method: Regularized Evolution

Distributed training is paramount (~300 CPUs for 3 days, ~20K candidates)

Speedup tricks:

- Hurdle environment (cartpole) – fail fast!
- Functional equivalence hashing and caching – dedup!

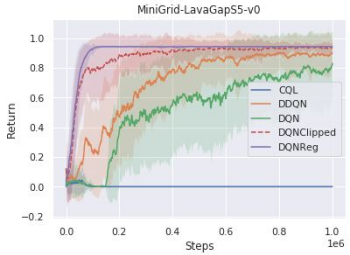
Discovered Algorithms

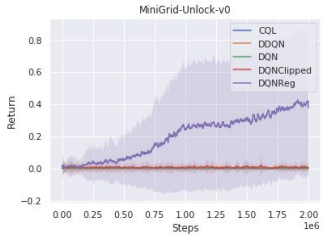
DQNReg: regular DQN + a regularization term

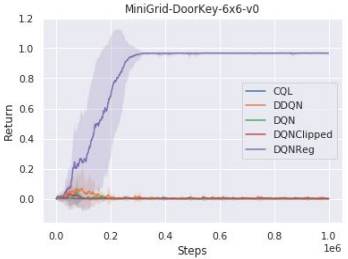
$$L_{\text{DQNReg}} = 0.1 * Q(s_t, a_t) + \delta^2$$

DQNReg generalizes to unseen classic Control, MiniGrids and Atari tasks.

Trained on non-image environments.
Not tuned to Atari games.

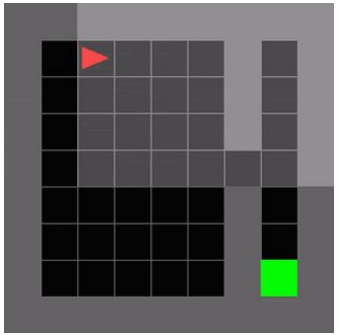




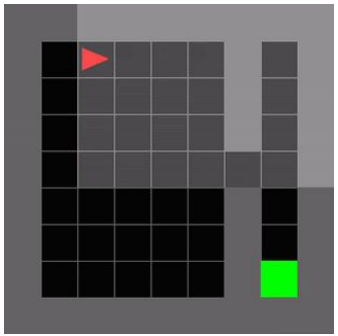


Env	DQN	DDQN	PPO	DQNReg
Asteroid	1364.5	734.7	2097.5	**2390.4**
Bowling	50.4	68.1	40.1	**80.5**
Boxing	88.0	91.6	94.6	**100.0**
RoadRunner	39544.0	44127.0	35466.0	**65516.0**

Baselines taken from reported numbers.



DQN



DQNReg

Google Research

In code and colab

PyGlove (AutoML library) Acme (RL library)

1. Search space definition

```
97    for i in range(search_program_length):
98      program_lst.append(
99          pg.oneof(
100             product_input_ops(inputs, existing_ops, i, operators, freeze_ops)))
```

PyGlove allows arbitrarily complex HP

A list of nodes, each node is a **tunable object**, as indicated by **pg.oneof(<list>)**

2. Search algorithm (Regularized Evolution)

```
282   return pg.evolution.Evolution(
283       reproduction=(
284           # Tournament selection and mutation.
285           pg.evolution.selectors.Random(tournament_size, seed=seed) >>
286           pg.evolution.selectors.Top(1) >> graph_mutator),
287       population_init=(graph_generator, population_size),
288       population_update=(
289           # Pop out oldest individual and update functional equivalence cache.
290           pg.evolution.selectors.Last(population_size) >> update_cache))
```

Google Research

In code and colab (continued)

3. Main training loop

(decoded) HP

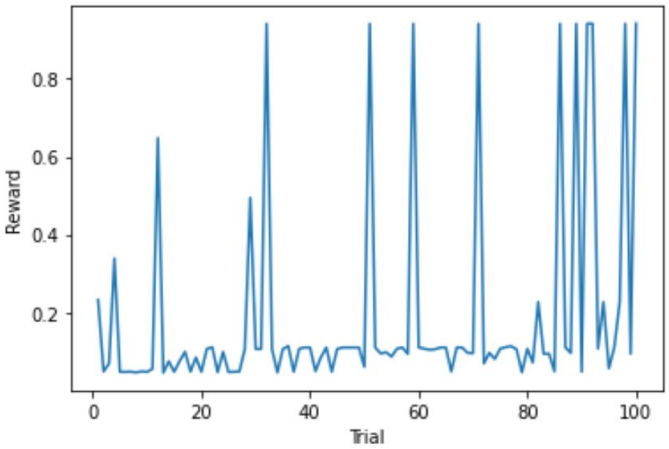
Evaluation result goes here

```
128     for program_spec, feedback in pg.sample(
129         search_space, generator, FLAGS.max_trials):
```

Search space

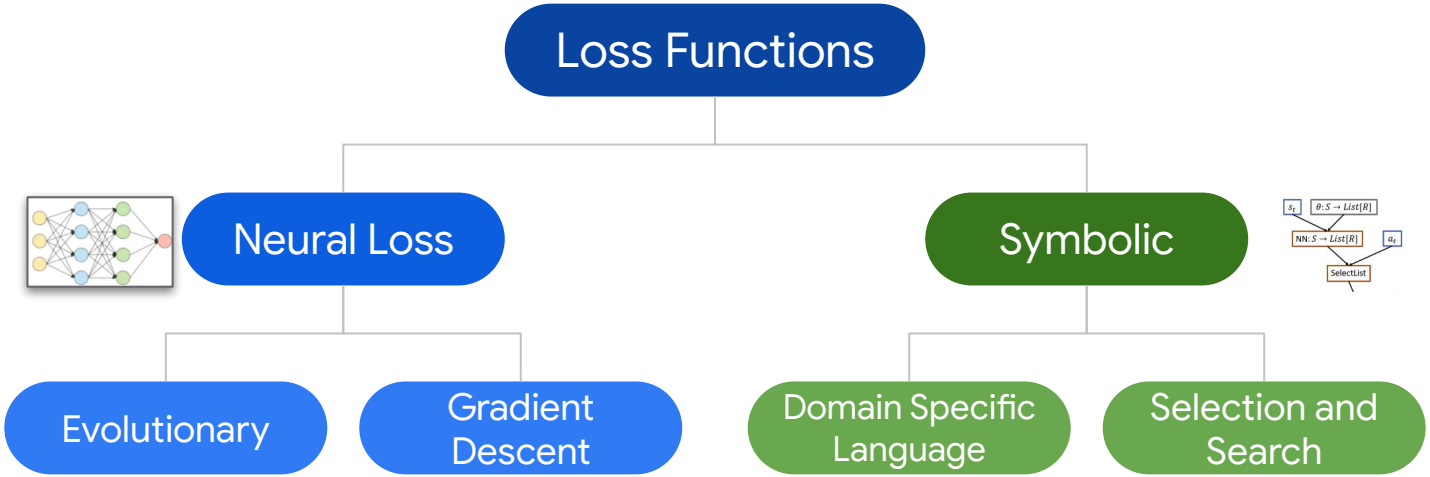
Regularized Evolution

Toy Training Curve



End-to-end colab:
https://github.com/google/brain_autorl/blob/main/evolving_rl/EvolvingRL_Demo.ipynb

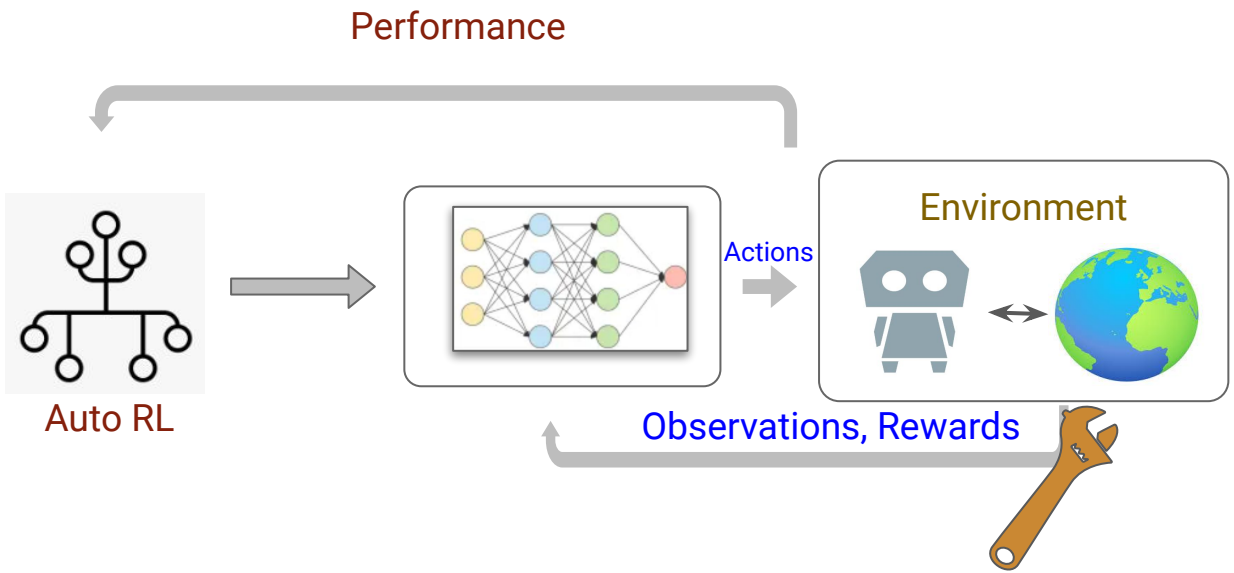
What are we learning? Algorithms!



Careful design + pruning + optimization.

Generalize across domains in discrete action spaces.
PG and Actor/Critic more challenging. [Garau-Luis et al., 2022]

What to Learn? Rewards

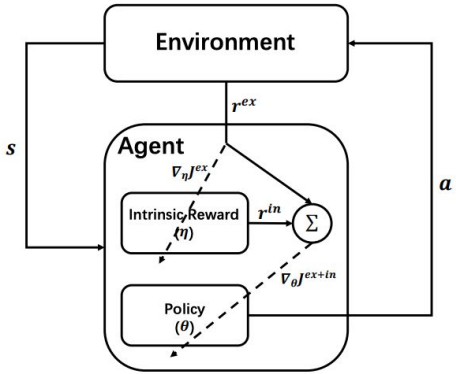


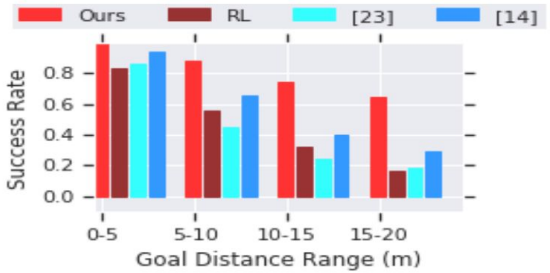
Main idea: Learn intrinsic rewards that maximize the excertic rewards.

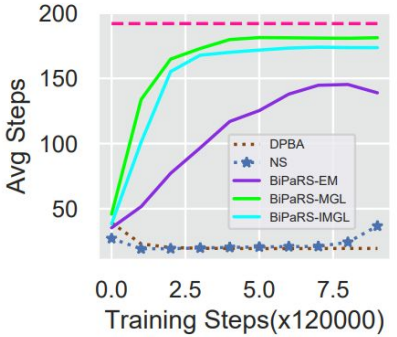
Reward tuning is difficult.

What to learn? Task Components. Rewards.

Sparse rewards. Difficult to design rewards.







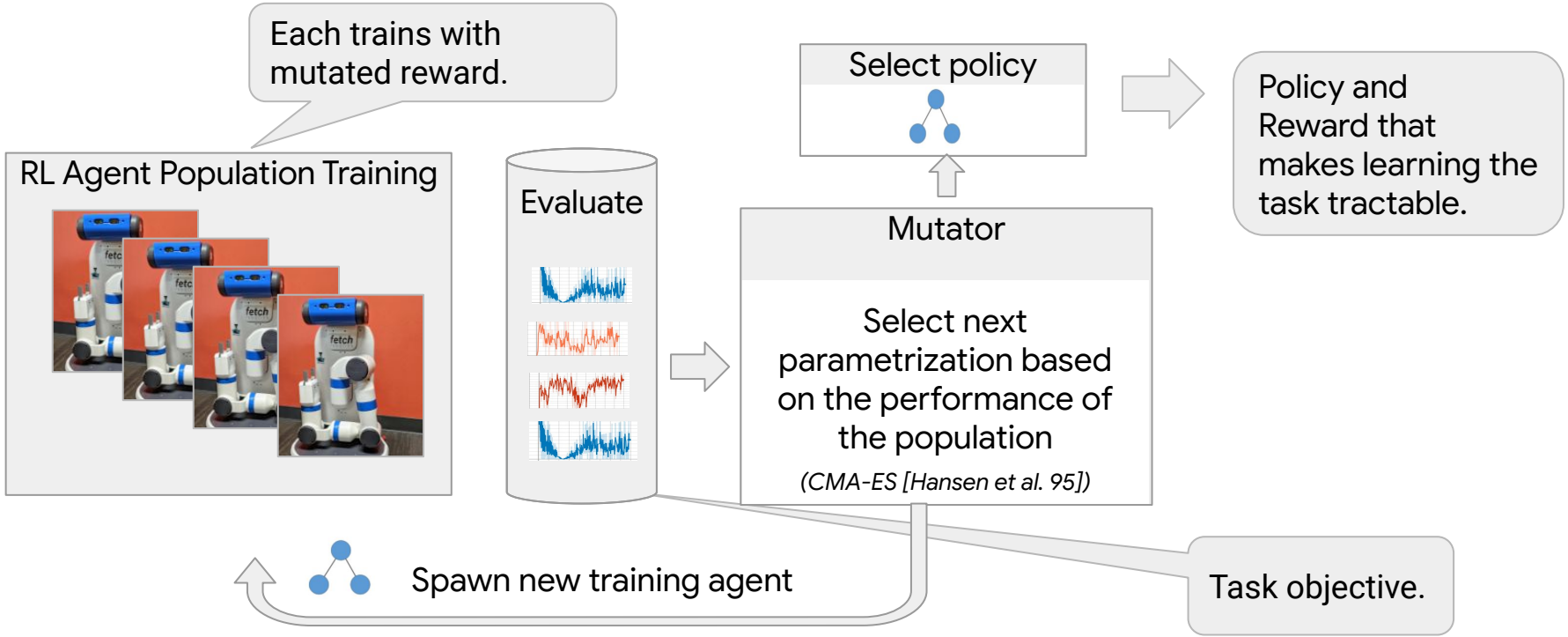
[Zhen et al. 2018] adds intrinsic reward neural network. Inner and outer: GD.

[Chiang et al. 2019] Feature-based intrinsic reward w/ ES.

BiPaRS [Hu et al. 2020] Feature-based reward shaping function w/ meta-learning.

Reward shaping helps for more complex tasks. Exact method is less important.

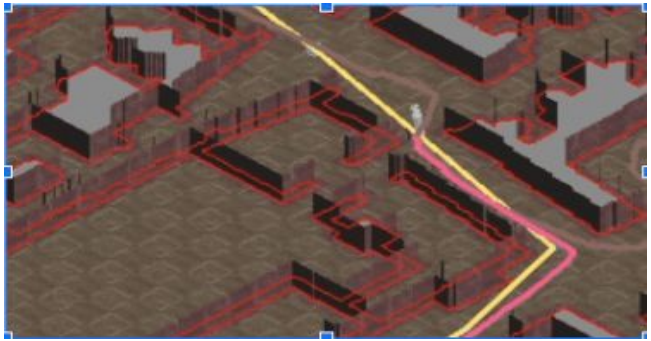
Learn the intrinsic reward that completes task.



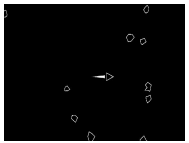
Single trial: Equivalent of 12 days of training. 12 hours wall time.
1000 trials: Equivalent of 32 years of training experience. 5 days wall time.

[Learning Navigation Behaviors End to End with AutoRL, Chiang*, Faust,* Fiser, Francis, RA-L/ICRA 2019] [Preprint, Video]

Google Research

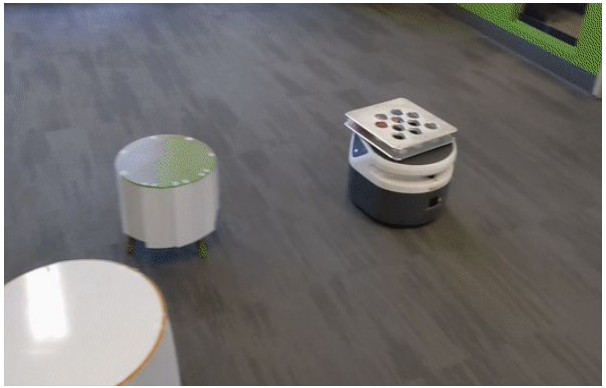


Zero-shot transfer



Differential drive.

Kinodynamic constraints.



Works on variety of robots.

Generalizes to real world, unseen environment.

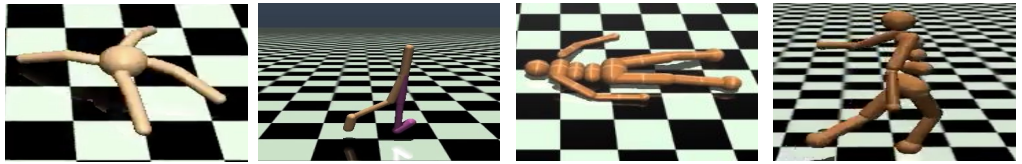
Adapts to changes on the go.

[Learning Navigation Behaviors End to End with AutoRL, Chiang*, Faust,* Fiser, Francis, RA-L/ICRA 2019 Preprint, Video]

Google Research

How general is intrinsic reward learning?

SAC [Haarnoja et al, 2018]
PPO [Schulman et al, 2017]
DDPG [Lillicrap et al, 2015]

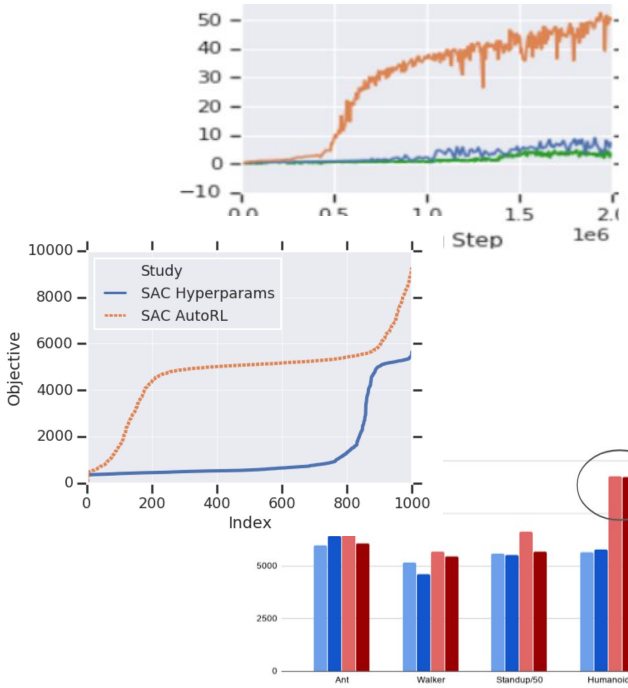


Intrinsic reward helps in more complex tasks.

Learn the rewards instead of hyper-parameter tuning, on a fixed training budget.

Codependency between RL algorithm and intrinsic reward.

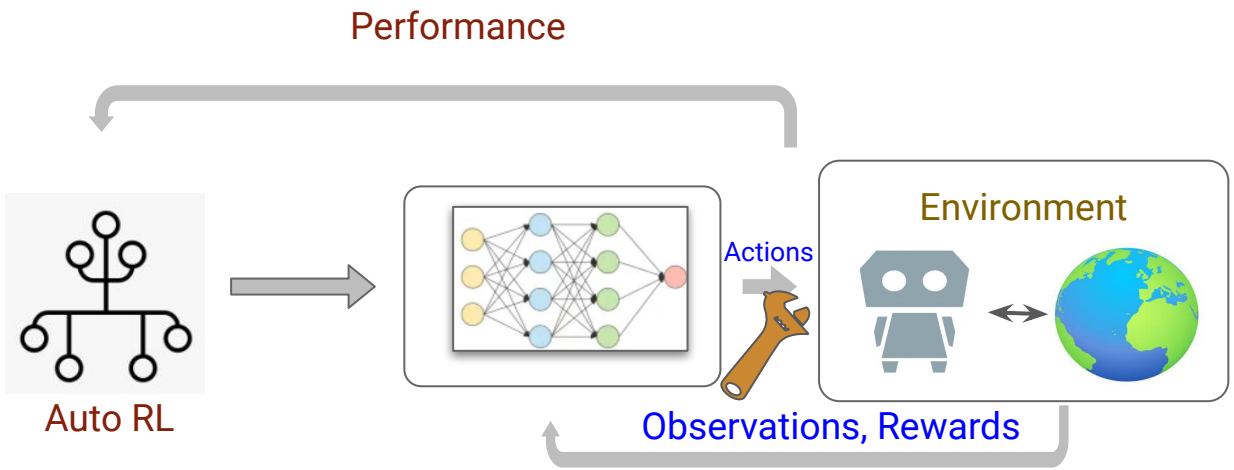
Simple objective performs almost as well -- without hand engineering.



[Evolving Rewards to Automate Reinforcement Learning, Faust, Francis, Mehta, 6th AutoML @ ICML 2019] [Preprint, Video]

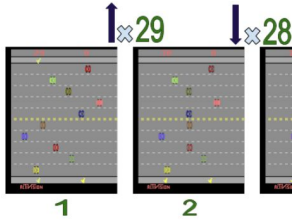
Google Research

What to Learn? Environment-based learning.

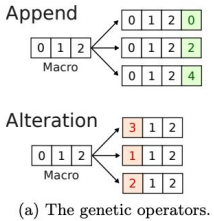


What to learn? Agent-based environment learning.

Action Space



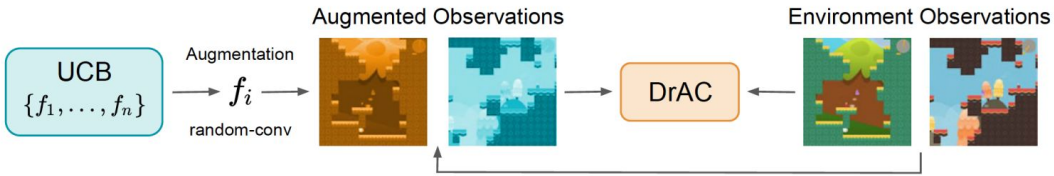
FiGAR learns action and its duration. [Sharma et al, 2017].



(a) The genetic operators.

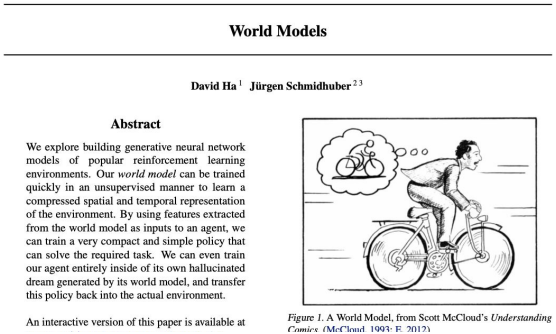
[Farahani and Mozayani 2019] combine actions into macro w/ GA

Observations



[Raileanu et al. 2021] UCB bandit selects image transformations for observations.

Dynamics



World Models

David Ha [1] Jürgen Schmidhuber [2 3]

Abstract

We explore building generative neural network models of popular reinforcement learning environments. Our *world model* can be trained quickly in an unsupervised manner to learn a compressed spatial and temporal representation of the environment. By using features extracted from the world model as inputs to an agent, we can train a very compact and simple policy that can solve the required task. We can even train our agent entirely inside of its own hallucinated dream generated by its world model, and transfer this policy back into the actual environment.

An interactive version of this paper is available at

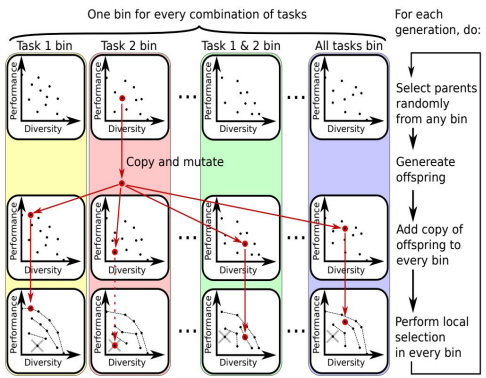
Figure 1. A World Model, from Scott McCloud's *Understanding Comics* (McCloud, 1993; F, 2012)

World Models learn vision, memory, and environment w/ VAE. [Ha and Schmidhuber, 2018]

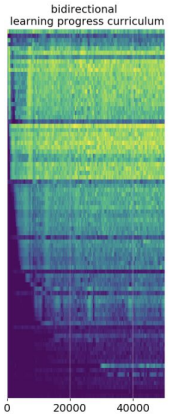
Google Research

What to Learn? Curriculum.

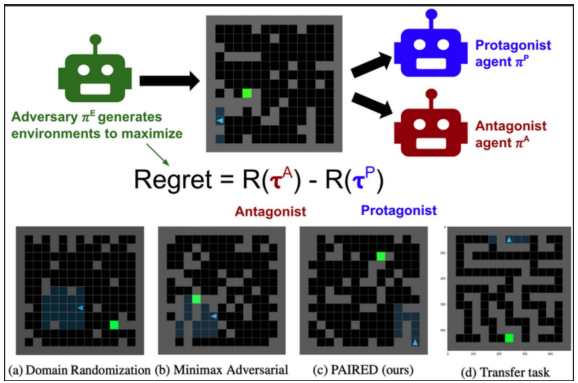
Order of tasks and environments for the agent to train on.



Combinatorial Multi-Objective Evolutionary Algorithm (CMOEA) [Huizinga and Clune, 2019]



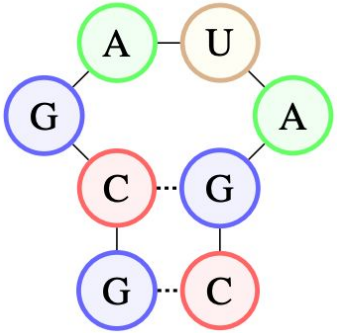
[Kanitscheider et al, 2019] Learning progress based multi-task curriculum



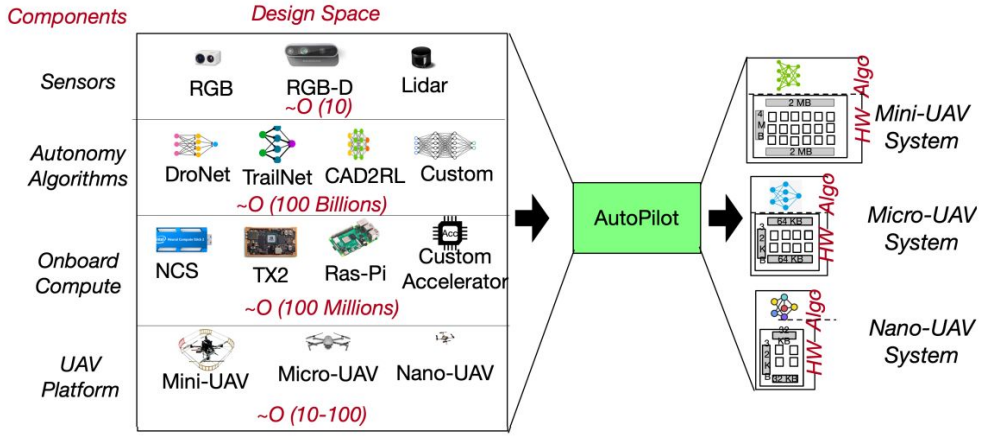
PAIRED constrained environment generation and RL agent. [Dennis et al, 2020]

Improves generalization. Sensitive to learning methods.

What to Learn? Combination of components.



LEARNA: reward + neural architecture + learning hyperparameters
[Runge et al, 2019]



Auto-pilot: NAS + accelerator
[Krishnan et al, 2022]

Summary of What to Learn?

What we know

Do learn learning HP.

Do learn intrinsic rewards for complex problems. Codependency with RL algorithm.

Do use curriculum for complex problems. Sensitive to task.

Loss function learning has promising generalization results in discrete action spaces. Requires careful design + pruning + optimization.

What we don't know

Very little work in NAS.

Very little work in agent learning.

What are important components to learn?

Which HP are nonstationary?

How to learn many components?

What are the best learning methods for each component?

Open Problems

Open Problems

- Theoretical understanding
  - HP co-dependencies.
  - Learning method appropriateness.
- Practical benchmarks
  - Non-trivial RL problem, and tractible outer loop.
  - Metrics.
- Nurture vs. nature trade off
- Integrative methods
  - Population-based training, multi-generational evolution + GD
  - Larger groups of HP, rewards, models, full algorithms.

Summary

TL;Dr;

Fundamental difference between RL and SL - more knobs and interdependencies. Stability of training and generalization due to non-stationarity and out of distribution.

Computational cost (AutoRL) = cost (AutoML) x cost (RL)

Hyperparameter taxonomy: numerical, symbolic, neural. Stationary or nonstationary.

Learn selected parts of the system: learning HP, NAS, loss, rewards, agent and environment components.

Future: Co-dependencies. Nonstationary hyperparameters. Scaling up.

Demos:
github.com/google/vizier/tree/main/demos
github.com/google/brain_autorl/blob/main/evolving_rl/EvolvingRL_Demo.ipynb
github.com/jcoreyes/evolvingrl

Ten+ years of full AutoRL training.
Joint curriculum and nonstationary hyperparameter training.

Acknowledgements

Yingjie Miao

John D. Co-Reyes

Richard Song

Jack Parker-Holder

Journal of Artificial Intelligence Research 74 (2022) 517-568 Submitted 01/2022; published 06/2022

Automated Reinforcement Learning (AutoRL): A Survey and Open Problems

Jack Parker-Holder JACKPH@ROBOTS.OX.AC.UK
University of Oxford

Raghu Rajan RAJANR@CS.UNI-FREIBURG.DE
University of Freiburg

Xingyou Song XINGYOUSONG@GOOGLE.COM
Google Research, Brain Team

André Biedenkapp BIEDENKA@CS.UNI-FREIBURG.DE
University of Freiburg

Yingjie Miao YINGJIEMIAO@GOOGLE.COM
Google Research, Brain Team

Theresa Eimer EIMER@TNT.UNI-HANNOVER.DE
Leibniz University Hannover

Baohe Zhang ZHANGB@CS.UNI-FREIBURG.DE
University of Freiburg

Vu Nguyen VUTNGN@AMAZON.COM
Amazon Australia

Roberto Calandra RCALANDRA@FB.COM
Meta AI

Aleksandra Faust SANDRAFAUST@GOOGLE.COM
Google Research, Brain Team

Frank Hutter FH@CS.UNI-FREIBURG.DE
University of Freiburg & Bosch Center for Artificial Intelligence

Marius Lindauer LINDAUER@TNT.UNI-HANNOVER.DE
Leibniz University Hannover

Abstract

The combination of Reinforcement Learning (RL) with deep learning has led to a series of impressive feats, with many believing (deep) RL provides a path towards generally capable agents. However, the success of RL agents is often highly sensitive to design choices in the training process, which may require tedious and error-prone manual tuning. This makes it challenging to use RL for new problems and also limits its full potential. In many other areas of machine learning, AutoML has shown that it is possible to automate such design choices, and AutoML has also yielded promising initial results when applied to RL. However, *Automated Reinforcement Learning* (AutoRL) involves not only standard applications of AutoML but also includes additional challenges unique to RL, that naturally produce a different set of methods. As such, AutoRL has been emerging as an important area of research in RL, providing promise in a variety of applications from RNA design to playing

Google Research

Thank You

Fundamental difference between RL and SL - more knobs and interdependencies. Stability of training and generalization due to non-stationarity and out of distribution.

Computational cost (AutoRL) = cost (AutoML) x cost (RL)

Hyperparameter taxonomy: numerical, symbolic, neural. Stationary or nonstationary.

Learn selected parts of the system: learning HP, NAS, loss, rewards, agent and environment components.

Future: Co-dependencies. Nonstationary hyperparameters. Scaling up.

Demos:
github.com/google/vizier/tree/main/demos
github.com/google/brain_autorl/blob/main/evolving_rl/EvolvingRL_Demo.ipynb
github.com/jcoreyes/evolvingrl