Reinforcement Learning

Speaker: Nando de Freitas
Slide contributors: Bobak Shahriari, Caglar Gulcehre, David Silver, Feryal Behbahani, Ksenia Konyushkova, Marc Bellemare, Matt Hoffman and Tom Le Paine, Tom Schaul
Special thanks: Bobak Shahriari and Feryal Behbahani

Khipu 2019

Machine learning

1. ML is about inferring knowledge from observations or experiences, subject to the physical laws of the world.
2. ML is also about using this knowledge to guide observation and hypothesis testing.
3. ML is about creating new knowledge, using the present knowledge, to solve a large diversity of novel problems.

Important to distinguish:

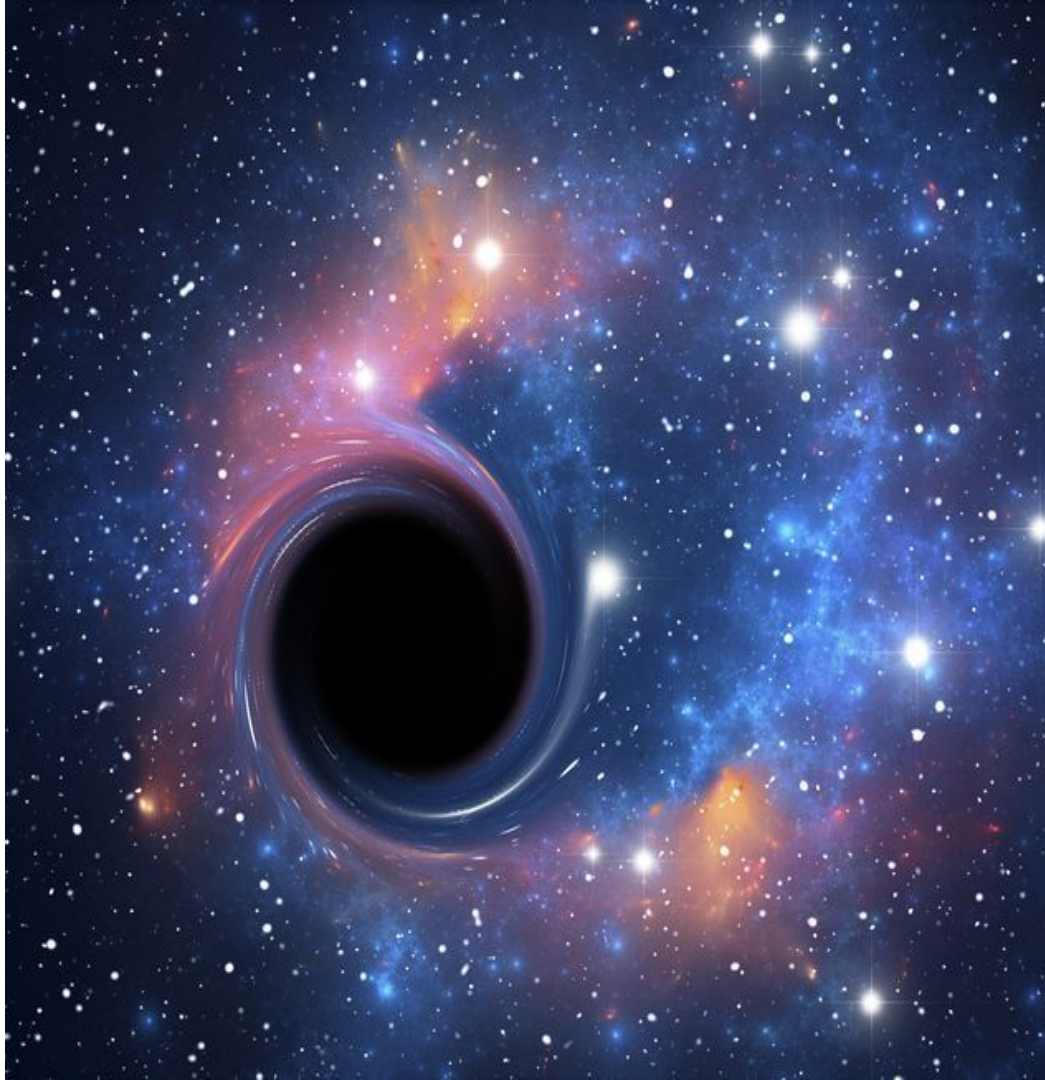
- Associative knowledge
- Explanatory knowledge

Magic association recipe

- Largest possible network with few general inductive biases.
- Massive curated dataset. Clear goal.
- Capitalize on the best existing communication, memory and computation infrastructure.

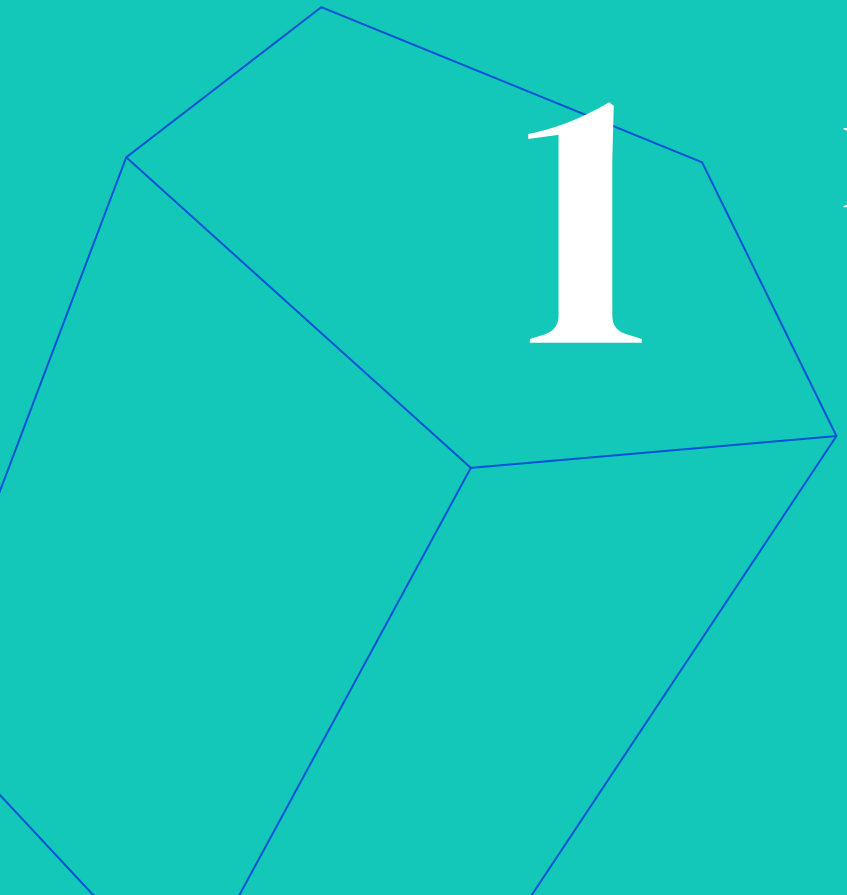
Outline

1. RL Concepts
2. Policy gradients
3. Dynamic programming
4. Deep Q-networks
5. Distributional RL
6. D4PG
7. PPO and MPO
8. R2D3
9. Applications of RL
    - AlphaX
    - Batch RL

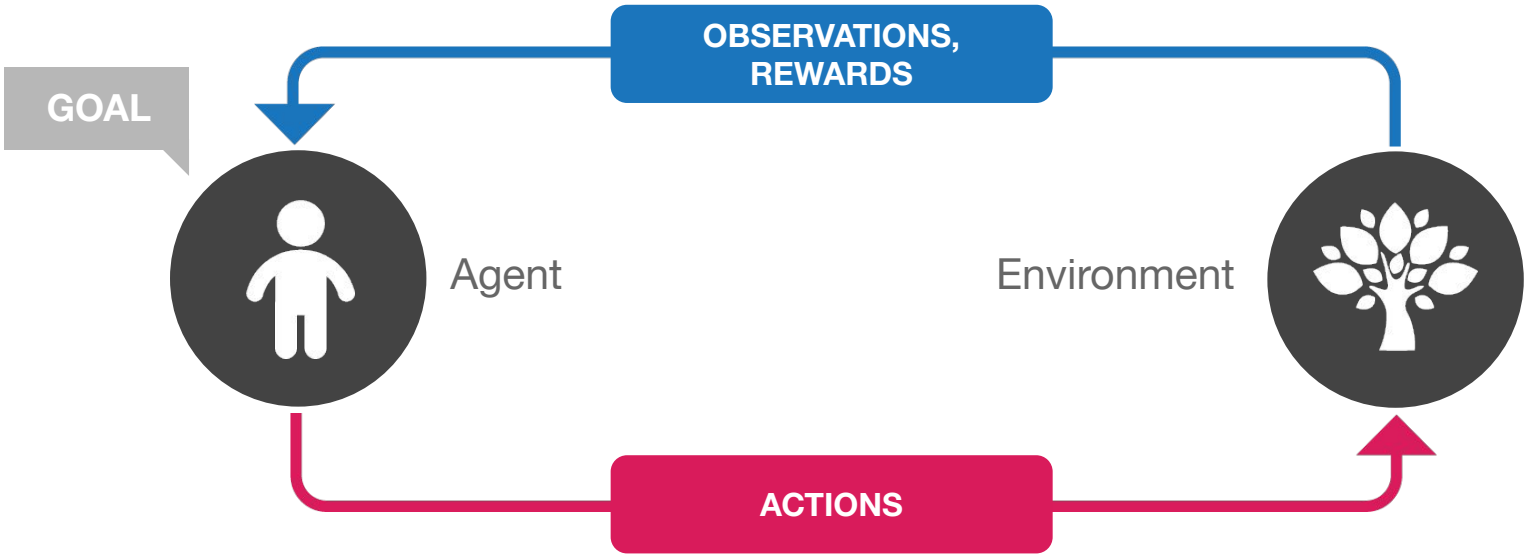


DeepMind

1 RL Concepts



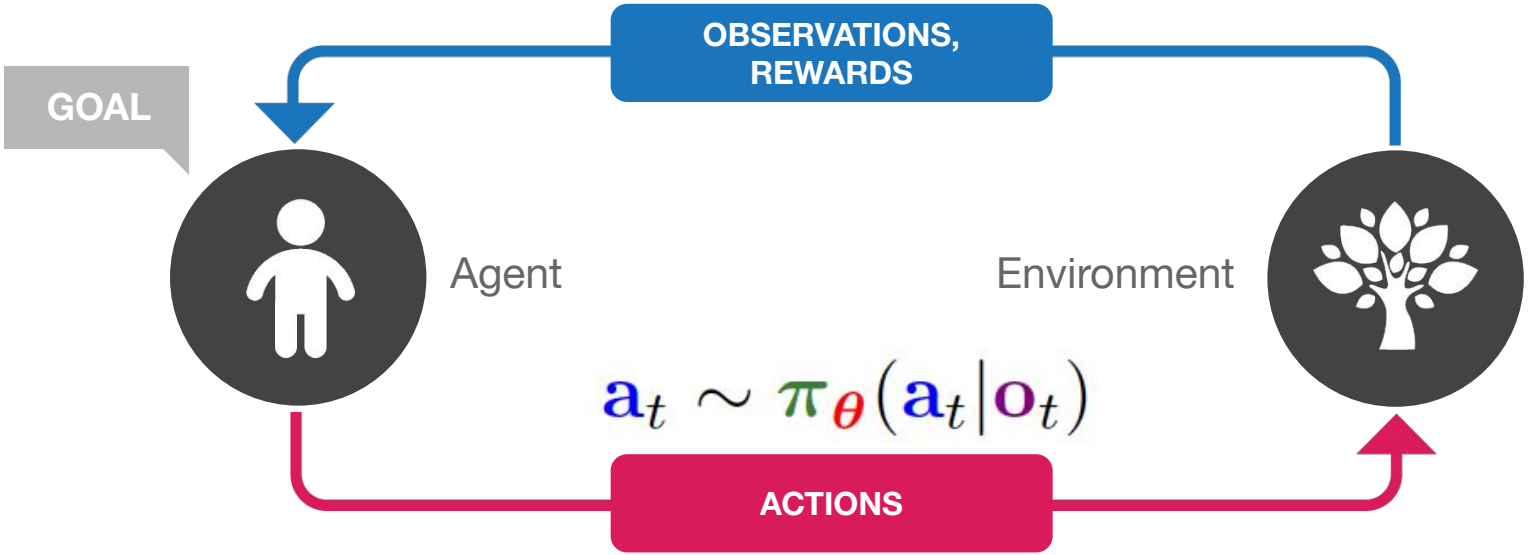
Reinforcement learning



Making good decisions by learning from experience

$$\mathbf{o}_{t+1} \sim P(\mathbf{o}_{t+1}|\mathbf{a}_t, \mathbf{o}_t)$$

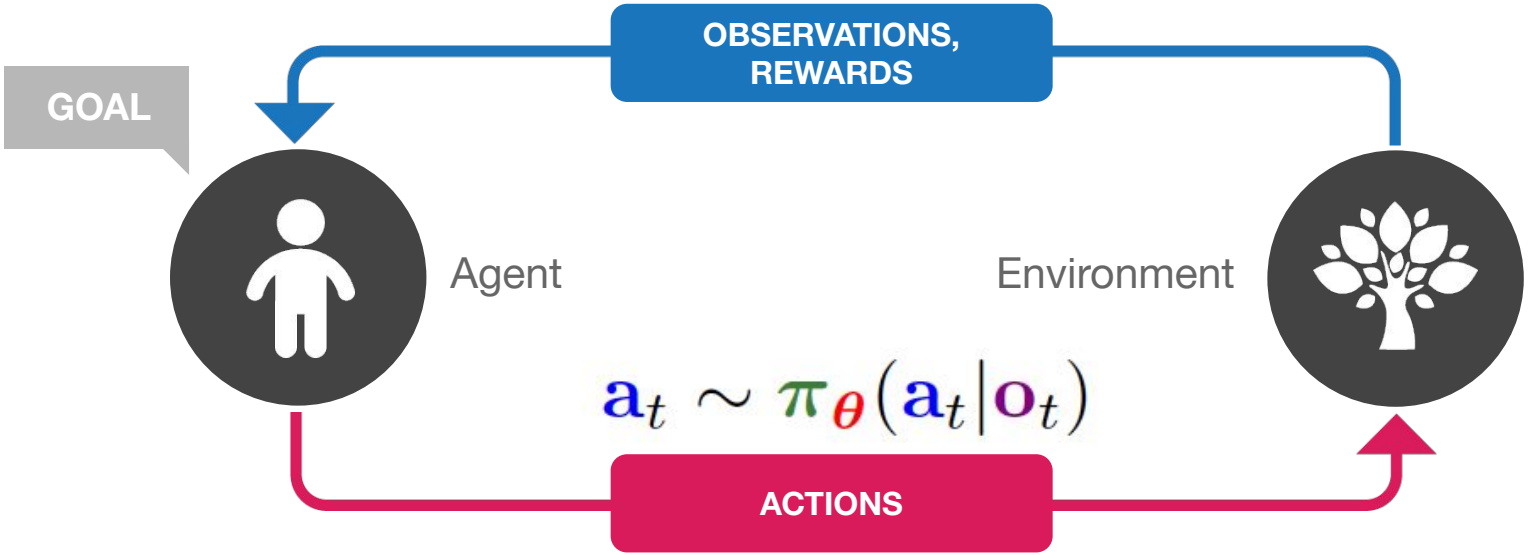
$$\mathbf{r}_t = \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$$



Making good decisions by learning from experience

$$\mathbf{o}_{t+1} \sim P(\mathbf{o}_{t+1} | \mathbf{a}_t, \mathbf{o}_t)$$

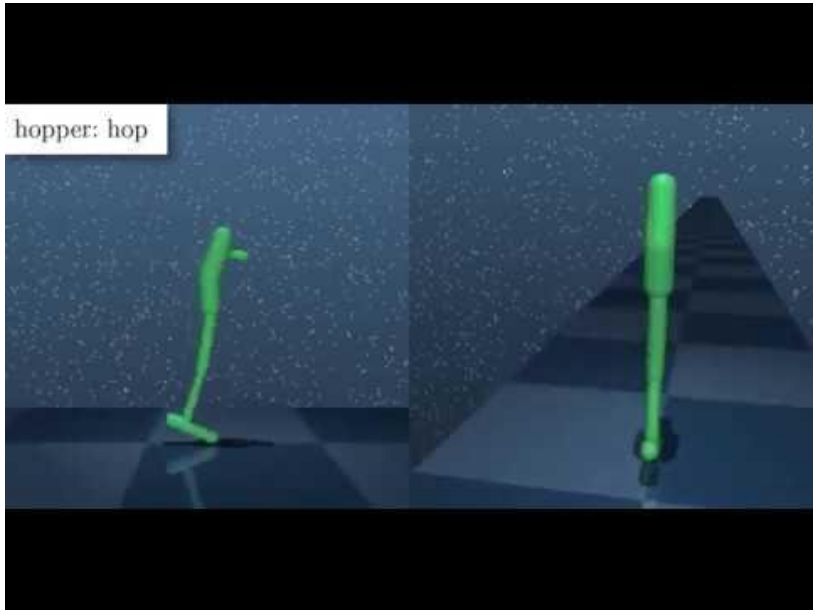
```
next_timestep = environment.step(action)
```




```
action = actor.policy(timestep.observation)
```

Making good decisions by learning from experience

Environments and tasks

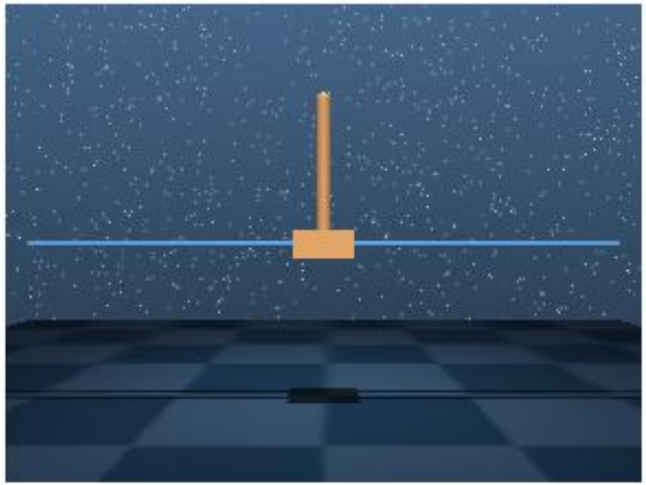



```
from dm_control import suite
```

Making good decisions by learning from experience

Environments and tasks

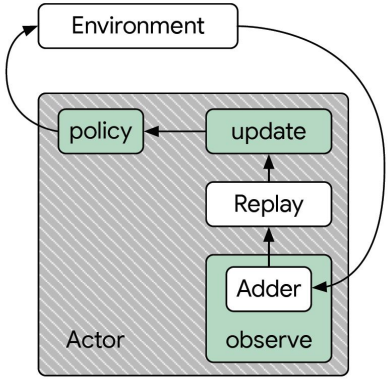
Example: The agent tries to balance the pole up by moving the cart left and right



```
from dm_control import suite

domain_name='cartpole'
task_name='balance'
environment = suite.load(domain_name, task_name)
```

Making good decisions by learning from experience

Generating episodes of experience

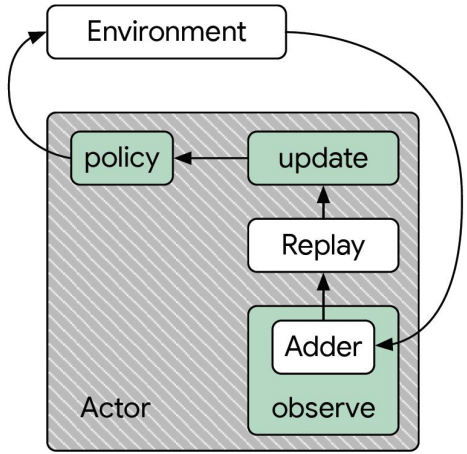


1: **while** True **do**
2: Make initial observation o_0
3: $t \leftarrow 0$; $e_t \leftarrow$ False
4: **while** not e_t **do**
5: $a_t \leftarrow \pi(o_t)$
6: Step the environment to obtain $(r_t, \kappa_t, e_{t+1}, o_{t+1})$
7: Observe $(o_t, a_t, r_t, \kappa_t)$
8: Update the policy π
9: $t \leftarrow t + 1$
10: **end while**
11: Observe the final state o_t
12: **end while**

```
# Make an initial observation.
timestep = environment.reset()

# Run an episode.
while not timestep.last():
  # Generate an action with the agent policy.
  action = actor.policy(timestep.observation)
  # Step in the environment.
  next_timestep = environment.step(action)
  # Make an observation and update the actor.
  actor.observe(timestep.observation,
                action,
                next_timestep.reward,
                next_timestep.discount)
  actor.update()
  timestep = next_timestep
# Make a final observation.
actor.observe_last(timestep.observation)
```

Making good decisions by learning from experience

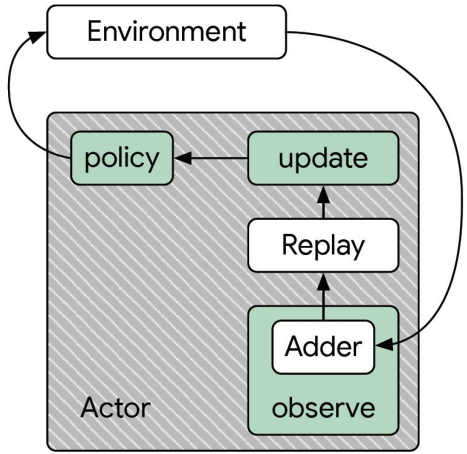
Building an actor




```
class RandomActor(core.Actor):

  def __init__(self, action_spec):
    # Make the agent output a valid action.
    self.scale = action_spec.maximum - action_spec.minimum
    self.offset = action_spec.minimum
    self.shape = action_spec.shape

  def policy(self, observation):
    # This actor chooses actions uniformly at random.
    raw_action = np.random.rand(*self.shape)
    return self.scale * raw_action + self.offset
```

Making good decisions by learning from experience

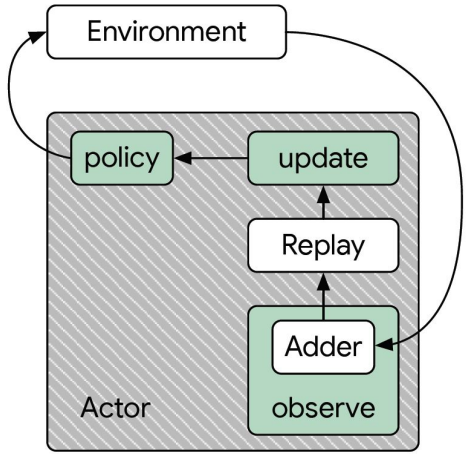
Building an actor




```
def observe(self, observation, action, reward, discount):
  """Random actor does not record experience."""
  pass

def observe_last(self, observation):
  """Random actor does not record experience."""
  pass

def update(self):
  """Random actor does not learn from experience."""
  pass
```

Making good decisions by learning from experience

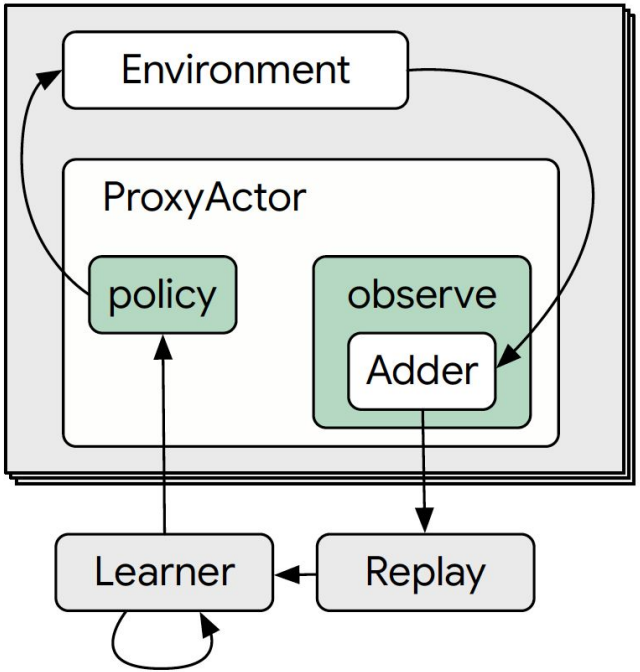
Building an actor




```
# Create an instance of a random actor.

action_spec = environment.action_spec()
actor = RandomActor(action_spec)
```

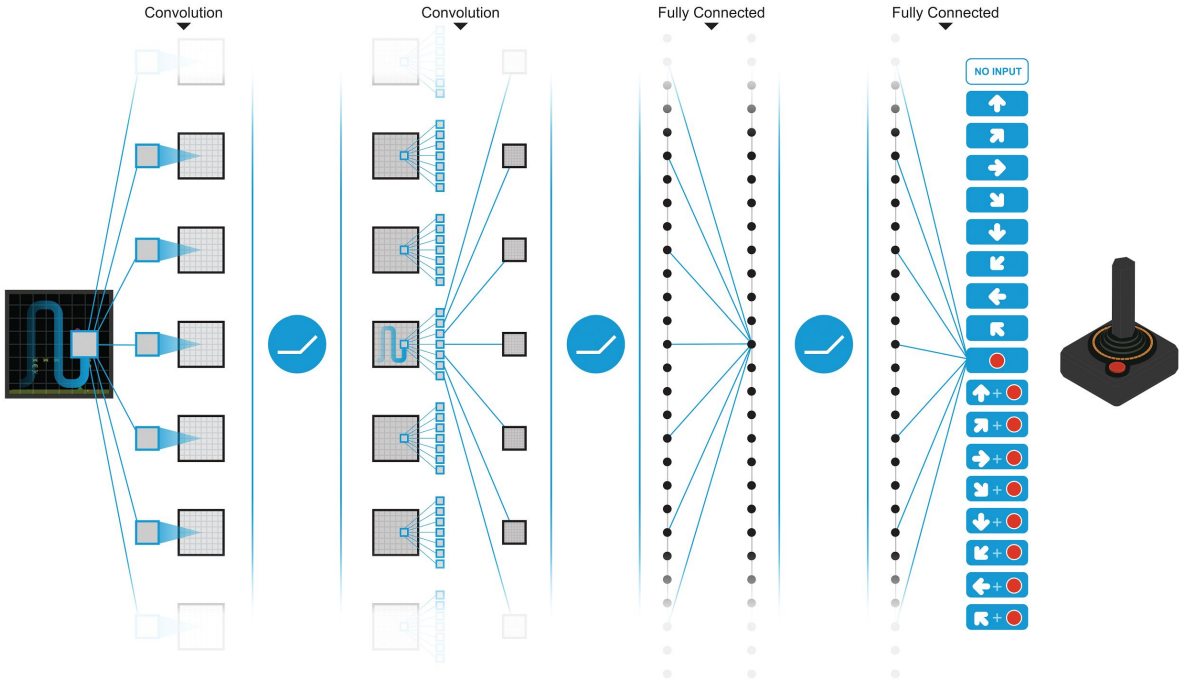
Making good decisions by learning from experience

Distributed actors and centralized learner



Making good decisions by learning from experience

Deep RL uses deep neural nets as policies



Making good decisions by learning from experience

Rewards

- Extrinsic
  - Environmental
  - Typically hardcoded
- Intrinsic
  - Social (eg influence, imitation, explanation, language)
  - Desire to control (empowerment)
  - Curiosity
  - Losses learned via meta-learning
  - Learning progress
  - Explanation

Making good decisions by learning from experience

DeepMind

2 Policy gradients

Mathematical formulation of RL

$$\tau = \{\mathbf{o}_0, \mathbf{a}_0, \mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_T\}$$ Experience / episode

$$\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$$ Policy

$$\mathbf{o}_{t+1} \sim P(\mathbf{o}_{t+1}|\mathbf{a}_t, \mathbf{o}_t)$$ Markov transition

$$\mathbf{r}_t = \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$$ Rewards

$$\mathbf{R}_t = \sum_{n=t}^{T} \mathbf{r}(\mathbf{o}_n, \mathbf{a}_n, \mathbf{o}_{n+1})$$ Returns

Mathematical formulation of RL

$$\tau = \{\mathbf{o}_0, \mathbf{a}_0, \mathbf{o}_1, \mathbf{a}_1, \ldots, \}$$

$$\mathbf{R}_0 = \sum_{n=0}^{\infty} \gamma^n \mathbf{r}(\mathbf{o}_n, \mathbf{a}_n, \mathbf{o}_{n+1})$$

For infinite horizons, we can use a discount factor $\gamma \in (0, 1)$

More weight is placed on more immediate future rewards.

The objective of RL

A rational agent maximizes its expected reward. This leaves us with a well-specified **objective function** for finding the best policy parameters

$$
\begin{aligned}
\rho^{\pi}(\theta) &= \mathbb{E}_{\tau}\left[\sum_{t=0}^{T} \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})\right] \\
&= \mathbb{E}_{\prod_t \pi_\theta(\mathbf{a}_t|\mathbf{o}_t) P(\mathbf{o}_{t+1}|\mathbf{a}_t,\mathbf{o}_t)}\left[\sum_{t=0}^{T} \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})\right] \\
&= \mathbb{E}_{P}\left[\sum_{\mathbf{a}_{0:T}} \left(\sum_{t=0}^{T} \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})\right) \pi_\theta(\mathbf{a}_{0:T}|\mathbf{o}_{0:T})\right]
\end{aligned}
$$

RL in 3 equations

$$\nabla_\theta \rho^\pi(\theta) = \mathbb{E}_\tau \left[\left(\sum_{t=0}^{T} \mathbf{R}_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{o}_t) \right) \right]$$

$$\nabla_\theta \rho^\pi(\theta) = \mathbb{E}_{d^\pi(\mathbf{o})\pi(\mathbf{a}|\mathbf{o})} \left[\mathbf{Q}^\pi(\mathbf{o}, \mathbf{a}) \nabla \log \pi_\theta(\mathbf{a}|\mathbf{o}) \right]$$

$$\mathbf{Q}^\pi(\mathbf{o}, \mathbf{a}) = \mathbf{r}(\mathbf{o}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} \mathbb{E}_{\mathbf{o}'} \left[\mathbf{Q}^\pi(\mathbf{o}', \mathbf{a}') | \mathbf{o}, \mathbf{a} \right]$$

Follow the gradient

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \rho^{\boldsymbol{\pi}}(\boldsymbol{\theta}) &= \mathbb{E}_P \left[\sum_{\mathbf{a}_{0:T}} \left(\sum_{t=0}^{T} \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1}) \right) \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_{0:T} | \mathbf{o}_{0:T}) \right] \\
&= \mathbb{E}_P \left[\sum_{\mathbf{a}_{0:T}} \left(\sum_{t=0}^{T} \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1}) \right) \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_{0:T} | \mathbf{o}_{0:T}) \, \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_{0:T} | \mathbf{o}_{0:T}) \right] \\
&= \mathbb{E}_P \left[\sum_{\mathbf{a}_{0:T}} \left(\sum_{t=0}^{T} \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1}) \right) \left(\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{o}_t) \right) \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_{0:T} | \mathbf{o}_{0:T}) \right] \\
&= \mathbb{E}_P \left[\sum_{\mathbf{a}_{0:T}} \left(\sum_{t=0}^{T} \left(\sum_{n=0}^{T} \mathbf{r}(\mathbf{o}_n, \mathbf{a}_n, \mathbf{o}_{n+1}) \right) \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{o}_t) \right) \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_{0:T} | \mathbf{o}_{0:T}) \right] \\
&= \mathbb{E}_\tau \left[\sum_{t=0}^{T} \left(\sum_{n=0}^{T} \mathbf{r}(\mathbf{o}_n, \mathbf{a}_n, \mathbf{o}_{n+1}) \right) \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{o}_t) \right] \\
&= \mathbb{E}_\tau \left[\sum_{t=0}^{T} \left(\sum_{n=t}^{T} \mathbf{r}(\mathbf{o}_n, \mathbf{a}_n, \mathbf{o}_{n+1}) \right) \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{o}_t) \right] \\
&= \mathbb{E}_\tau \left[\left(\sum_{t=0}^{T} \mathbf{R}_t \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{o}_t) \right) \right]
\end{aligned}
$$

The policy gradient

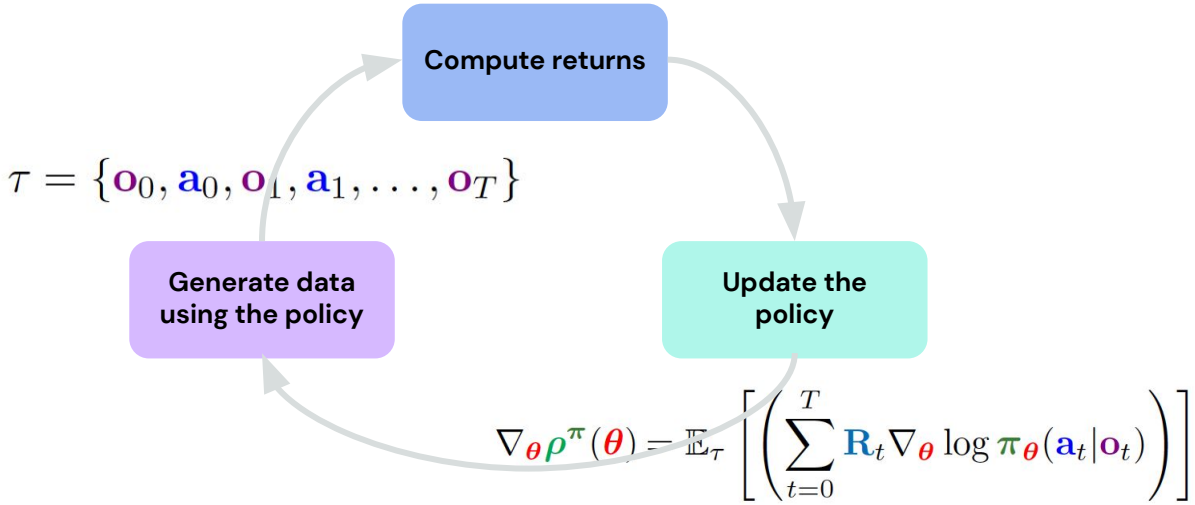
$$\nabla_{\boldsymbol{\theta}} \rho^{\pi}(\boldsymbol{\theta}) = \mathbb{E}_{\tau}\left[\left(\sum_{t=0}^{T} \mathbf{R}_t \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{o}_t)\right)\right]$$

It may be approximated by Monte Carlo using N trajectories (episodes)

$$\nabla_{\boldsymbol{\theta}} \rho^{\pi}(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \mathbf{R}_t^{(i)} \nabla \log \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)}|\mathbf{o}_t^{(i)})$$

The policy gradient

$$\mathbf{R}_t = \sum_{n=t}^{T} \mathbf{r}(\mathbf{o}_n, \mathbf{a}_n, \mathbf{o}_{n+1})$$



The policy gradient: Initialize

```
class PolicyGradientActor(core.Actor):

  def __init__(self, policy_network):
    self.policy_network = policy_network
    # This actor will need to record its experiences.
    self.actions = []
    self.observations = []
    self.rewards = []

    # Flag to update agent, set to False until we've seen a full episode.
    self.update_agent_next = False

    # Create the optimizer. It uses DeepMind's snt toolbox instead of Keras.
    self.optimizer = snt.optimizers.Adam(learning_rate=1e-3)
```

The policy gradient: Make decisions

```
class PolicyGradientActor(core.Actor):

  def policy(self, observation):
    # Pack the observation in a trivial batch.
    observation_with_dummy_batch_dim = tree.map_structure(
        lambda x: tf.expand_dims(x, axis=0), observation)
    # Get the action distribution at this state.
    action_distribution = self.policy_network(observation_with_dummy_batch_dim)

    # Return a sampled action.
    return action_distribution.sample()
```

The policy gradient: Record experience

```
class PolicyGradientActor(core.Actor):

  def observe(self, observation, action, reward, discount):
    # Add experience to agent buffer; ignore discount.
    self.actions.append(action)
    self.observations.append(observation)
    self.rewards.append(reward)

  def observe_last(self, observation):
    """Flag agent to update its variables; ignore observation."""
    self.update_agent_next = True
```

The policy gradient: Learn from experience

```
class PolicyGradientActor(core.Actor):

  def update(self):
    # Get returns first
    reward_to_go = get_reward_to_go(self.rewards)
    # Compute the policy gradient using automatic differentiation
    with tf.GradientTape() as tape:
      action_distribution = self.policy_network(stacked_observations)
      logprobs = action_distribution.log_prob(stacked_actions)
      loss = -tf.reduce_sum(reward_to_go * logprobs, axis=0)

    gradients = tape.gradient(loss, self.policy_network.trainable_variables)
    self.optimizer.apply(gradients, self.policy_network.trainable_variables)
```

Value functions

The state-action value function is the expected reward with the first action and observation fixed.

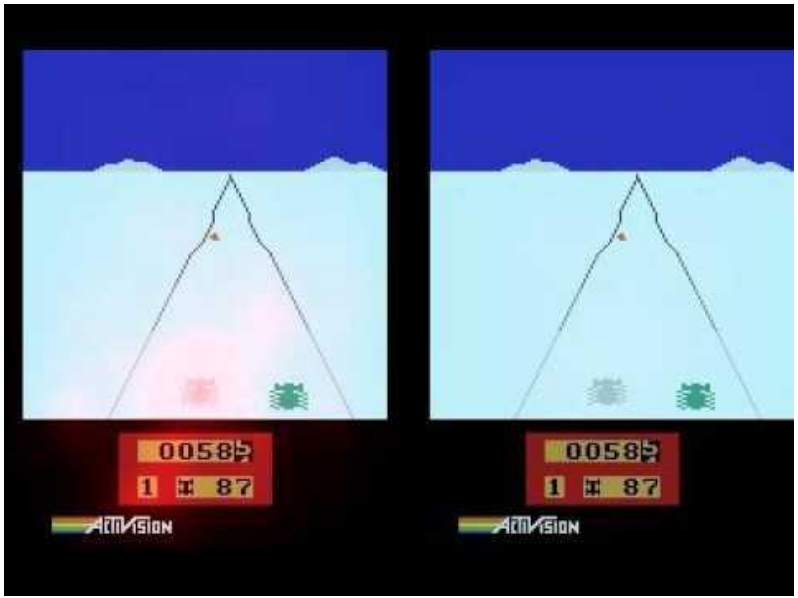
$$\mathbf{Q}^{\pi}(\mathbf{o}_0, \mathbf{a}_0) = \mathbb{E}_{\tau}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1}) \middle| \mathbf{o}_0, \mathbf{a}_0\right]$$

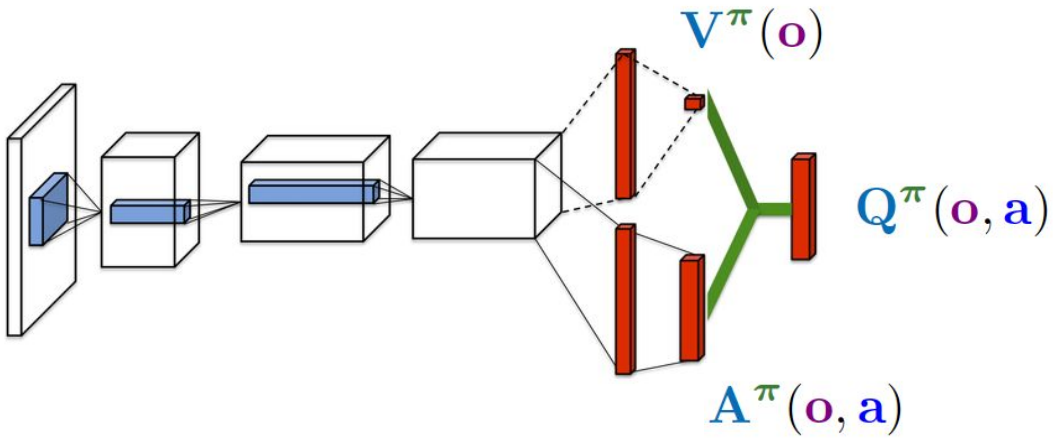
The value and advantage functions can be defined accordingly

$$\mathbf{V}^{\pi}(\mathbf{o}) = \mathbb{E}_{\pi(\mathbf{a}|\mathbf{o})}\left[\mathbf{Q}^{\pi}(\mathbf{o}, \mathbf{a})\right]$$

$$\mathbf{A}^{\pi}(\mathbf{o}, \mathbf{a}) = \mathbf{Q}^{\pi}(\mathbf{o}, \mathbf{a}) - \mathbf{V}^{\pi}(\mathbf{o})$$

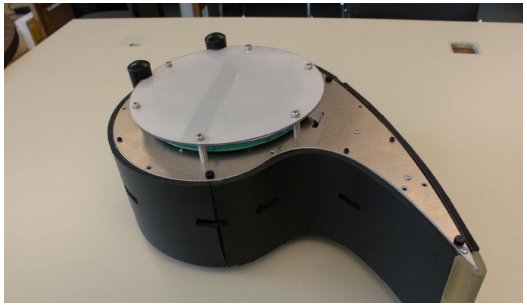
Value and advantage: Dueling networks

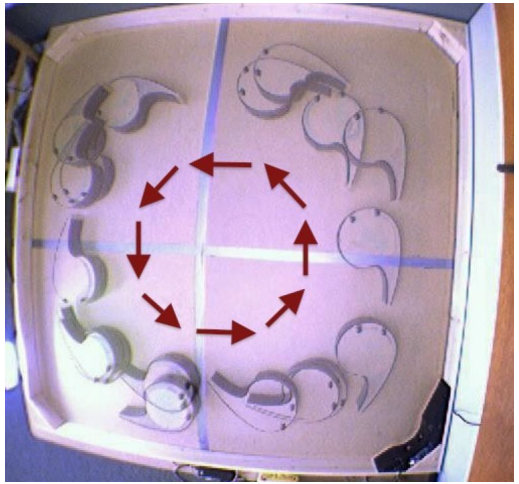


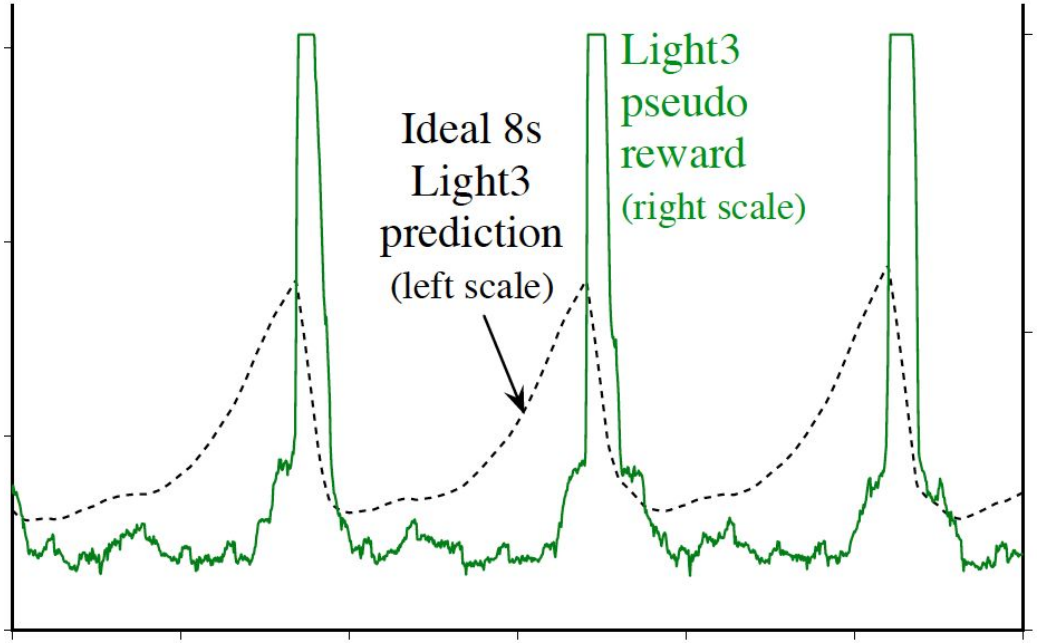


(Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, Nando Freitas, 2016)

Visualization of value functions







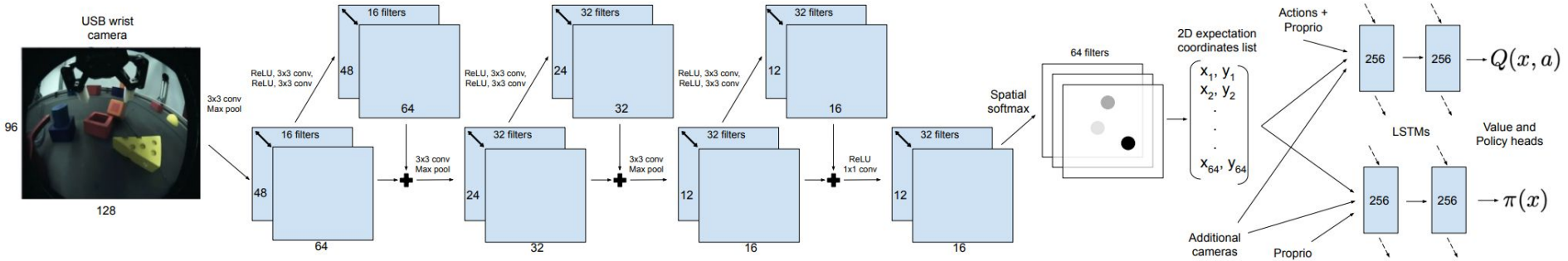
Nexting by Joseph Modayil, Adam White, and Richard Sutton, 2011

The policy gradient revisited

If we learn Q directly, we can use the following approximation

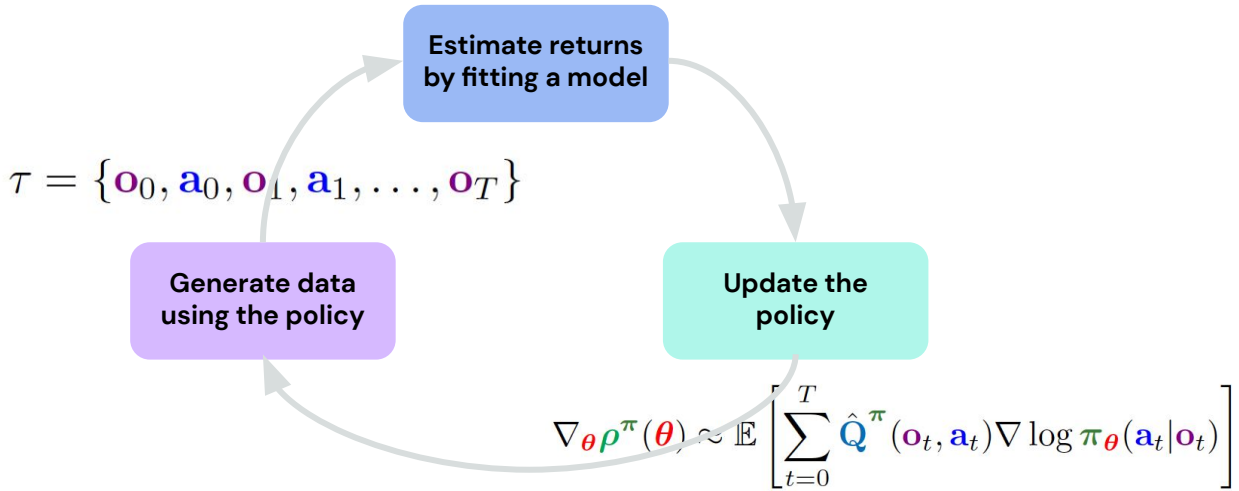
$$\nabla_\theta \rho^\pi(\theta) \approx \mathbb{E}\left[\sum_{t=0}^{T} \hat{\mathrm{Q}}^\pi(\mathbf{o}_t, \mathbf{a}_t) \nabla \log \pi_\theta(\mathbf{a}_t|\mathbf{o}_t)\right]$$

It has lower variance but higher bias (trade-off that leads to many algorithms)



The policy gradient

$$\mathbf{Q}^{\pi}(\mathbf{o}_0, \mathbf{a}_0) = \mathbb{E}_{\tau}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1}) \middle| \mathbf{o}_0, \mathbf{a}_0\right]$$



RL in 3 equations

$$\nabla_\theta \rho^\pi(\theta) = \mathbb{E}_\tau \left[\left(\sum_{t=0}^{T} \mathbf{R}_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{o}_t) \right) \right]$$

$$\nabla_\theta \rho^\pi(\theta) = \mathbb{E}_{d^\pi(\mathbf{o})\pi(\mathbf{a}|\mathbf{o})} \left[\mathbf{Q}^\pi(\mathbf{o}, \mathbf{a}) \nabla \log \pi_\theta(\mathbf{a}|\mathbf{o}) \right]$$

$$\mathbf{Q}^\pi(\mathbf{o}, \mathbf{a}) = \mathbf{r}(\mathbf{o}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} \mathbb{E}_{\mathbf{o}'} \left[\mathbf{Q}^\pi(\mathbf{o}', \mathbf{a}') | \mathbf{o}, \mathbf{a} \right]$$

The policy gradient Theorem

$$\nabla_{\theta}\rho^{\pi}(\theta) = \mathbb{E}_{\tau}\left[\left(\sum_{t=0}^{T} \mathbf{R}_t \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t|\mathbf{o}_t)\right)\right]$$

Applying the Markov transition operators repeatedly leads to a fixed point (this is just like PageRank the famous google algorithm)

At **stationarity**, the following important result applies

$$\nabla_{\theta}\rho^{\pi}(\theta) = \mathbb{E}_{d^{\pi}(\mathbf{o})\pi(\mathbf{a}|\mathbf{o})}\left[\mathbf{Q}^{\pi}(\mathbf{o},\mathbf{a})\nabla \log \pi_{\theta}(\mathbf{a}|\mathbf{o})\right]$$

(Richard Sutton, David McAllester, Satinder Singh, Yishay Mansour, 2000)

DeepMind

3 Dynamic programming

Value functions and optimal value functions

The value of being in a state and following a deterministic policy $\mathbf{a}_t = \boldsymbol{\pi}(\mathbf{o}_t)$

$$\mathbf{V}^{\boldsymbol{\pi}}(\mathbf{o}_0) = \mathbb{E}_{\tau}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(\mathbf{o}_t, \boldsymbol{\pi}(\mathbf{o}_t), \mathbf{o}_{t+1}) \middle| \mathbf{o}_0\right]$$

The optimal value function

$$\mathbf{V}^{\star}(\mathbf{o}_0) = \max_{\boldsymbol{\pi}} \mathbb{E}_{\tau}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(\mathbf{o}_t, \boldsymbol{\pi}(\mathbf{o}_t), \mathbf{o}_{t+1}) \middle| \mathbf{o}_0\right]$$

Dynamic programming

$$
\begin{aligned}
\mathbf{V}^\star(\mathbf{o}_0) &= \max_{\boldsymbol{\pi}} \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(\mathbf{o}_t, \boldsymbol{\pi}(\mathbf{o}_t), \mathbf{o}_{t+1}) \middle| \mathbf{o}_0 \right] \\
&= \max_{\boldsymbol{\pi}} \mathbb{E}_\tau \left[\mathbf{r}(\mathbf{o}_0, \mathbf{a}_0, \mathbf{o}_1) + \sum_{t=1}^{\infty} \gamma^t \mathbf{r}(\mathbf{o}_t, \boldsymbol{\pi}(\mathbf{o}_t), \mathbf{o}_{t+1}) \middle| \mathbf{o}_0 \right] \\
&= \max_{\mathbf{a}_0, \boldsymbol{\pi}} \mathbb{E}_\tau \left[\mathbf{r}(\mathbf{o}_0, \mathbf{a}_0, \mathbf{o}_1) + \sum_{t=1}^{\infty} \gamma^t \mathbf{r}(\mathbf{o}_t, \boldsymbol{\pi}(\mathbf{o}_t), \mathbf{o}_{t+1}) \middle| \mathbf{o}_0 \right] \\
&= \max_{\mathbf{a}_0} \mathbb{E}_{\mathbf{o}_1} \left[\mathbf{r}(\mathbf{o}_0, \mathbf{a}_0, \mathbf{o}_1) + \max_{\boldsymbol{\pi}} \mathbb{E}_{\mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \ldots} \left\{ \sum_{t=1}^{\infty} \gamma^t \mathbf{r}(\mathbf{o}_t, \boldsymbol{\pi}(\mathbf{o}_t), \mathbf{o}_{t+1}) \middle| \mathbf{o}_1 \right\} \middle| \mathbf{o}_0 \right] \\
&= \max_{\mathbf{a}_0} \mathbb{E}_{\mathbf{o}_1} \left[\mathbf{r}(\mathbf{o}_0, \mathbf{a}_0, \mathbf{o}_1) + \gamma \max_{\boldsymbol{\pi}} \mathbb{E}_{\mathbf{o}_2, \mathbf{o}_3, \mathbf{o}_4, \ldots} \left\{ \sum_{t=1}^{\infty} \gamma^{t-1} \mathbf{r}(\mathbf{o}_t, \boldsymbol{\pi}(\mathbf{o}_t), \mathbf{o}_{t+1}) \middle| \mathbf{o}_1 \right\} \middle| \mathbf{o}_0 \right] \\
&= \max_{\mathbf{a}_0} \mathbb{E}_{\mathbf{o}_1} \left[\mathbf{r}(\mathbf{o}_0, \mathbf{a}_0, \mathbf{o}_1) + \gamma \mathbf{V}^\star(\mathbf{o}_1) \right]
\end{aligned}
$$

By definition

$$\mathbf{V}^{\star}(\mathbf{o}) = \max_{\mathbf{a}} \mathbb{E}_{\mathbf{o}'} \left[\mathbf{r}(\mathbf{o}, \mathbf{a}, \mathbf{o}') + \gamma \mathbf{V}^{\star}(\mathbf{o}') | \mathbf{o} \right]$$

$$\mathbf{V}(\mathbf{o}) = \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}, \mathbf{a}')$$

$$\mathbf{V}^{\star}(\mathbf{o}) = \max_{\mathbf{a}} \mathbb{E}_{\mathbf{o}'} \left[\mathbf{r}(\mathbf{o}, \mathbf{a}, \mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}^{\star}(\mathbf{o}', \mathbf{a}') | \mathbf{o} \right]$$

Hence

$$\mathbf{Q}^{\star}(\mathbf{o}, \mathbf{a}) = \mathbb{E}_{\mathbf{o}'} \left[\mathbf{r}(\mathbf{o}, \mathbf{a}, \mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}^{\star}(\mathbf{o}', \mathbf{a}') | \mathbf{o}, \mathbf{a} \right]$$

RL in 3 equations

$$\nabla_\theta \rho^\pi(\theta) = \mathbb{E}_\tau \left[\left(\sum_{t=0}^{T} \mathbf{R}_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{o}_t) \right) \right]$$

$$\nabla_\theta \rho^\pi(\theta) = \mathbb{E}_{d^\pi(\mathbf{o})\pi(\mathbf{a}|\mathbf{o})} \left[\mathbf{Q}^\pi(\mathbf{o}, \mathbf{a}) \nabla \log \pi_\theta(\mathbf{a}|\mathbf{o}) \right]$$

$$\mathbf{Q}^\pi(\mathbf{o}, \mathbf{a}) = \mathbf{r}(\mathbf{o}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} \mathbb{E}_{\mathbf{o}'} \left[\mathbf{Q}^\pi(\mathbf{o}', \mathbf{a}') | \mathbf{o}, \mathbf{a} \right]$$

DeepMind

4 Deep Q-Networks

DQN

Maintain an experience replay memory and sample $\{\mathbf{o}_j, \mathbf{a}_j, \mathbf{r}_j, \mathbf{o}'_j\}|_{j=1}^{N}$

$$L(\boldsymbol{\theta}_i) = \mathbb{E}_{\mathbf{o},\mathbf{a},\mathbf{o}'}\left[\left(\mathbf{r}(\mathbf{o},\mathbf{a},\mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}',\mathbf{a}',\boldsymbol{\theta}^\star) - \mathbf{Q}(\mathbf{o},\mathbf{a},\boldsymbol{\theta}_i)\right)^2\right]$$

Keep the target fixed for stability

$$\mathbf{y}_i = \mathbf{r}(\mathbf{o},\mathbf{a},\mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}',\mathbf{a}',\boldsymbol{\theta}^\star)$$

Follow the gradient to find theta

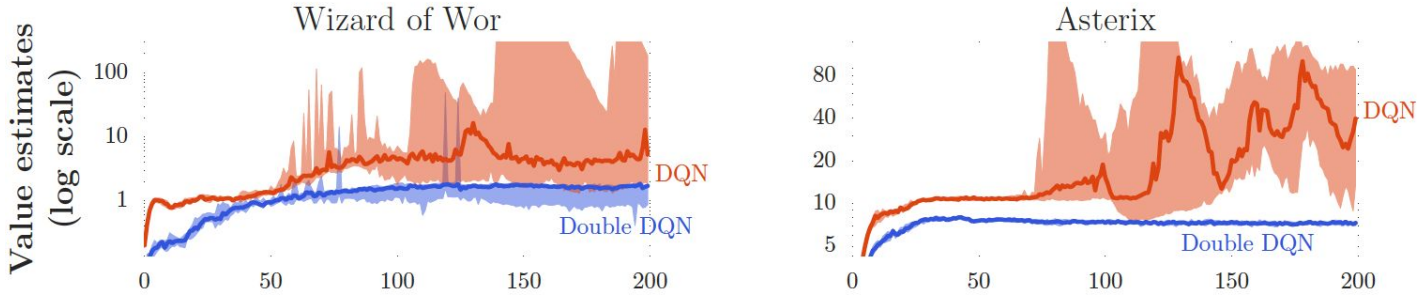
$$\nabla_{\boldsymbol{\theta}_i} L(\boldsymbol{\theta}_i) = \mathbb{E}_{\mathbf{o},\mathbf{a},\mathbf{o}'}\left[\left(\mathbf{r}(\mathbf{o},\mathbf{a},\mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}',\mathbf{a}',\boldsymbol{\theta}^\star) - \mathbf{Q}(\mathbf{o},\mathbf{a},\boldsymbol{\theta}_i)\right) \nabla_{\boldsymbol{\theta}_i} \mathbf{Q}(\mathbf{o},\mathbf{a},\boldsymbol{\theta}_i)\right]$$

$$\hat{\mathbf{a}} = \arg\max_{\mathbf{a}} \mathbf{Q}(\mathbf{o},\mathbf{a};\hat{\boldsymbol{\theta}})$$

Double DQN

DDQN aims to reduce over-confidence in Q estimates, which results from choosing the max of Q

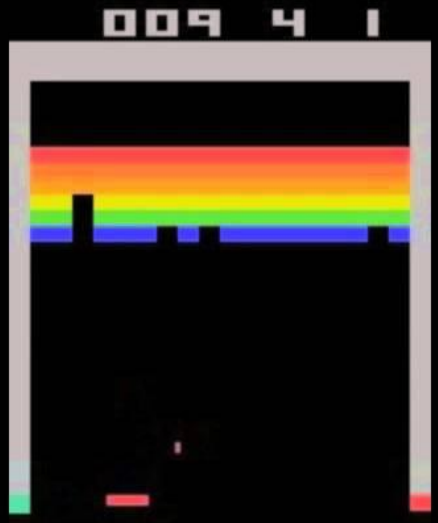
$$\mathbf{y}_i = \mathbf{r}(\mathbf{o}, \mathbf{a}, \mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}', \mathbf{a}', \boldsymbol{\theta}^\star)$$



$$\mathbf{y}_i = \mathbf{r}(\mathbf{o}, \mathbf{a}, \mathbf{o}') + \gamma \mathbf{Q}(\mathbf{o}', \arg\max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}, \mathbf{a}', \boldsymbol{\theta}_i), \boldsymbol{\theta}^\star)$$

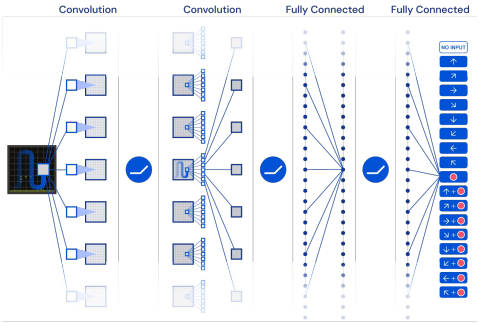
```python
class DQNActor(core.Actor):
  def update(self):
    # Uses sample from replay: (o_tm1, a_tm1, r_t, d_t, o_t). Note: tm1 = t-1.

    with tf.GradientTape() as tape:
      # Get online and target Q-values.
      q_tm1 = self._q_network(o_tm1)
      q_t_value = self._target_q_network(o_t)
      q_t_selector = self._q_network(o_t)

      # Double DQN target
      best_action = tf.argmax(q_t_selector, 1, output_type=tf.int32)
      double_q_bootstrapped = indexing_ops.batched_index(q_t_value, best_action)
      target = tf.stop_gradient(r_t + d_t * double_q_bootstrapped)
      qa_tm1 = indexing_ops.batched_index(q_tm1, a_tm1)

      # Temporal difference error and loss.
      td_error = target - qa_tm1
      loss = tf.reduce_mean(0.5 * tf.square(td_error), axis=0)

    # Compute gradients and update Q-network variables.
    gradients = tape.gradient(loss, self._q_network.trainable_variables)
    self.optimizer.apply(gradients, self._q_network.trainable_variables)
```

Atari with deep RL

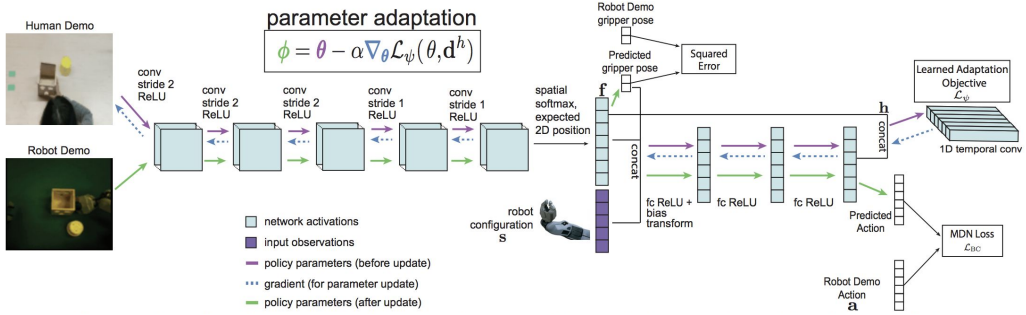


Nets got bigger in RL

DQN

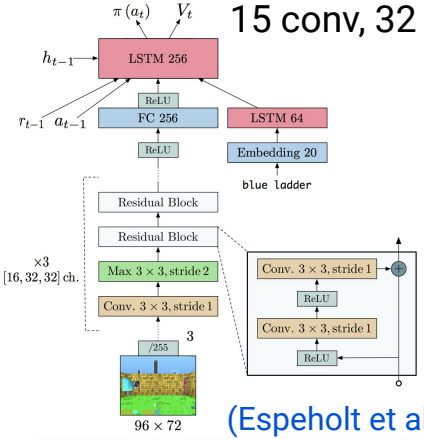


MAML third-person imitation
5 conv, 64 channels



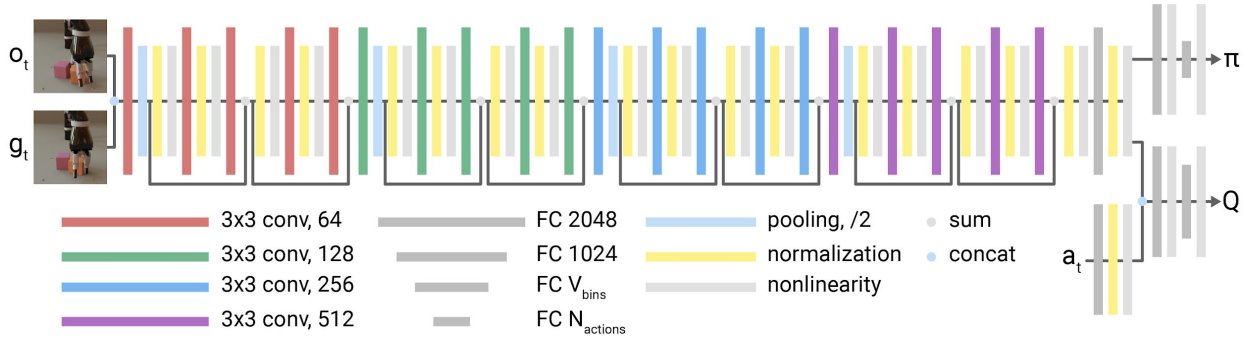
(Yu & Finn et al 2018)

IMPALA
15 conv, 32 channels



(Espeholt et al 2018)

MetaMimic
20 conv, 512 channels



(Paine et al 2018)

RL in 3 equations

$$\nabla_\theta \rho^\pi(\theta) = \mathbb{E}_\tau \left[\left(\sum_{t=0}^{T} \mathbf{R}_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{o}_t) \right) \right]$$

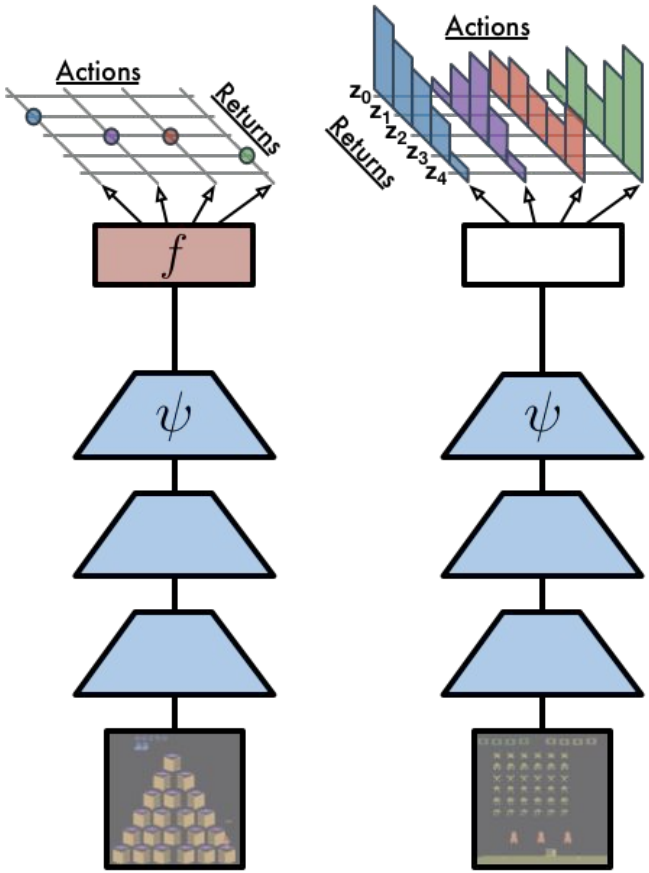
$$\nabla_\theta \rho^\pi(\theta) = \mathbb{E}_{d^\pi(\mathbf{o})\pi(\mathbf{a}|\mathbf{o})} \left[\mathbf{Q}^\pi(\mathbf{o}, \mathbf{a}) \nabla \log \pi_\theta(\mathbf{a}|\mathbf{o}) \right]$$

$$\mathbf{Q}^\pi(\mathbf{o}, \mathbf{a}) = \mathbf{r}(\mathbf{o}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} \mathbb{E}_{\mathbf{o}'} \left[\mathbf{Q}^\pi(\mathbf{o}', \mathbf{a}') | \mathbf{o}, \mathbf{a} \right]$$

DeepMind

5 Distributional RL

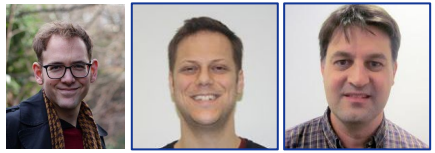
$\mathbf{Q}(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})$ $f_i(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})$



$$\mathbf{Z}(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta}) = z_i \quad w.p. \quad p_i(\mathbf{o}, \mathbf{a}) = \frac{e^{f_i(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})}}{\sum_j e^{f_j(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})}}$$

$$\begin{aligned} \hat{\mathbf{a}} &= \arg\max_{\mathbf{a}} \mathbf{Q}(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta}) \\ &= \arg\max_{\mathbf{a}} \mathbb{E}\left[\mathbf{Z}(\mathbf{o}, \mathbf{a}; \boldsymbol{\theta})\right] \\ &\approx \arg\max_{\mathbf{a}} \sum_{i=0}^{N-1} z_i p_i(\mathbf{o}, \mathbf{a}) \end{aligned}$$

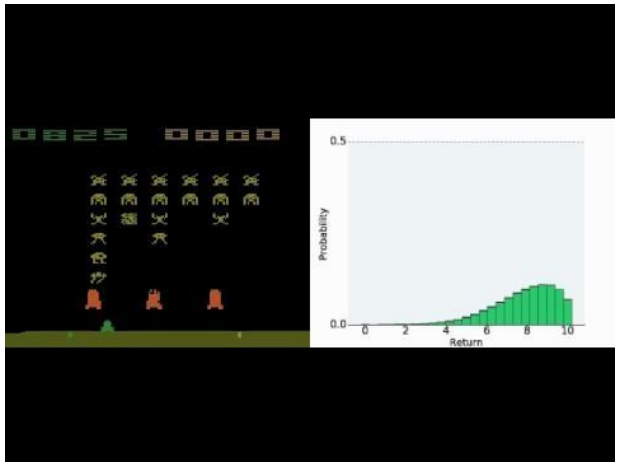
C51

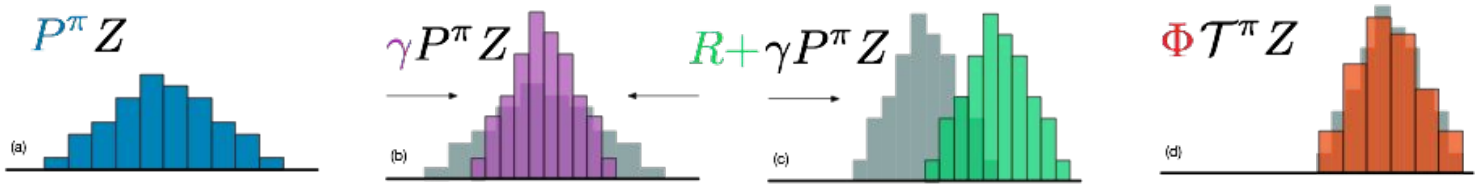


Minimize the TD distance in distribution using discrete distributions

$$\left(\mathbf{r}(\mathbf{o},\mathbf{a},\mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{o}',\mathbf{a}',\boldsymbol{\theta}^\star) - \mathbf{Q}(\mathbf{o},\mathbf{a},\boldsymbol{\theta}_i)\right)^2$$

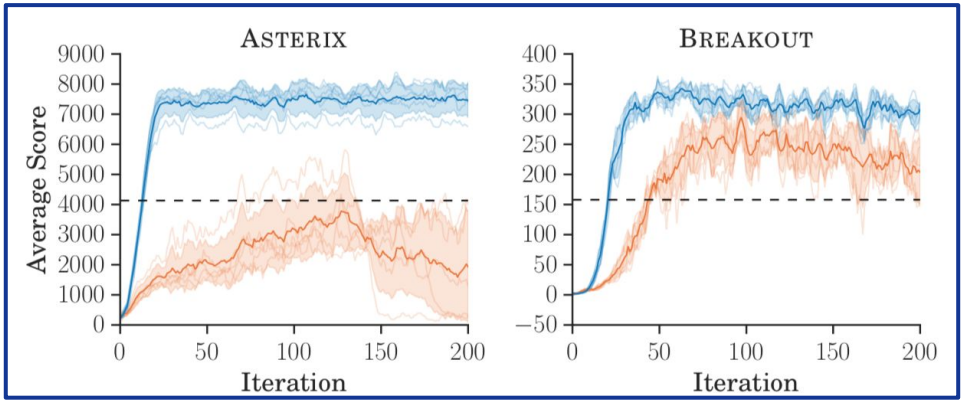
$$Dist\left(\mathbf{r}(\mathbf{o},\mathbf{a},\mathbf{o}') + \gamma \max_{\mathbf{a}'} \mathbf{Z}(\mathbf{o}',\mathbf{a}',\boldsymbol{\theta}^\star);\ \mathbf{Z}(\mathbf{o},\mathbf{a},\boldsymbol{\theta}_i)\right)$$



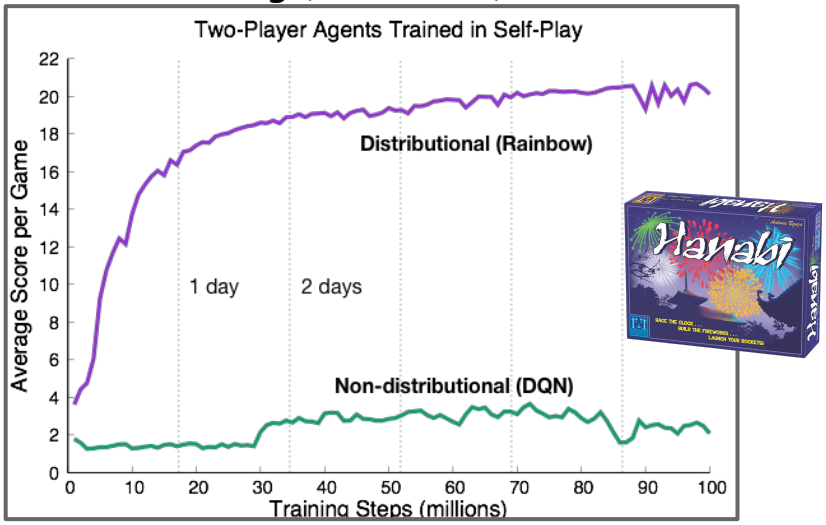


Bellemare*, Dabney*, Munos (ICML 2017). There is a nice blog on distributional RL by Massimiliano Tomassoli

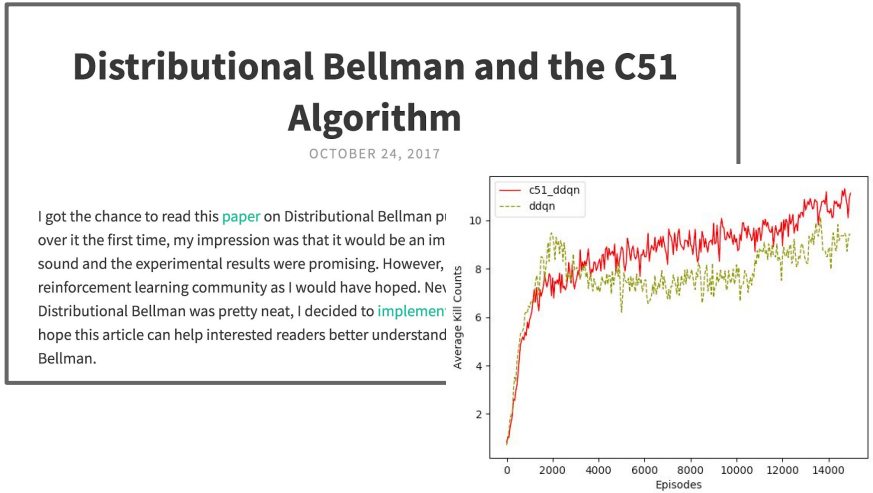
Batch deep RL, Agarwal, Schuurmans, Norouzi, 2019

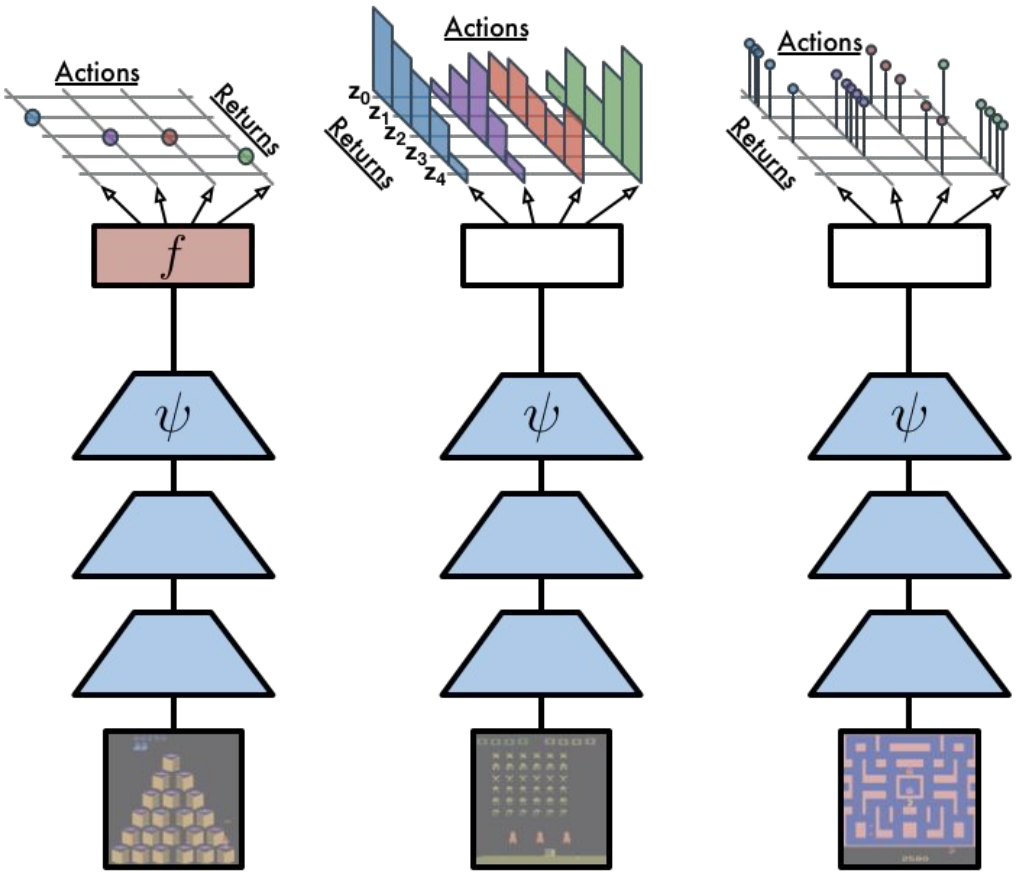


Hanabi Challenge, Bard et al., 2019



VizDoom, Felix Yu, [blog post]





(Dabney, Ostrovski, Silver, Munos, 2018)

DeepMind

6 Deterministic policy gradients d4pg

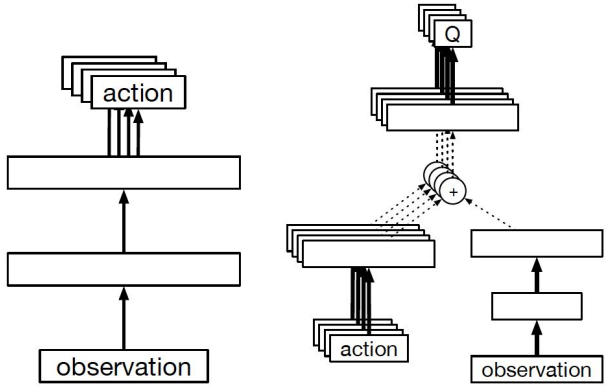
Deep Deterministic policy gradients (ddpg)

Apply the policy gradients theorem to deterministic policies

$$\begin{aligned}\rho^{\pi}(\theta) &= \mathbb{E}_{d^{\pi}(\mathbf{o})}\left[\mathbf{Q}^{\pi}(\mathbf{o}, \pi_{\theta}(\mathbf{o}))\right] \\ \nabla_{\theta}\rho^{\pi}(\theta) &= \mathbb{E}_{d^{\pi}(\mathbf{o})}\left[\nabla_{\mathbf{a}'}\mathbf{Q}^{\pi}(\mathbf{o}, \mathbf{a}')|_{\mathbf{a}'=\pi_{\theta}(\mathbf{o})}\nabla_{\theta}\pi_{\theta}(\mathbf{o})\right]\end{aligned}$$

Replay $\{\mathbf{o}_j, \mathbf{a}_j, \mathbf{r}_j, \mathbf{o}'_j\}|_{j=1}^{N}$ and follow the gradient

$$\begin{aligned}\delta_j &= \mathbf{r}_j + \gamma \mathbf{Q}^{\pi}_{\mathbf{W}^{\star}}(\mathbf{o}'_j, \pi_{\theta}(\mathbf{o}'_j)) - \mathbf{Q}^{\pi}_{\mathbf{W}}(\mathbf{o}_j, \mathbf{a}_j) \\ \mathbf{W} &= \mathbf{W} + \alpha_{critic} \sum_j \delta_j \nabla_{\mathbf{W}} \mathbf{Q}^{\pi}_{\mathbf{W}}(\mathbf{o}_j, \mathbf{a}_j) \\ \theta &= \theta + \alpha_{actor} \sum_j \nabla_{\mathbf{a}'} \mathbf{Q}^{\pi}_{\mathbf{W}}(\mathbf{o}_j, \mathbf{a}')|_{\mathbf{a}'=\pi_{\theta}(\mathbf{o}_j)} \nabla_{\theta}\pi_{\theta}(\mathbf{o}_j)\end{aligned}$$

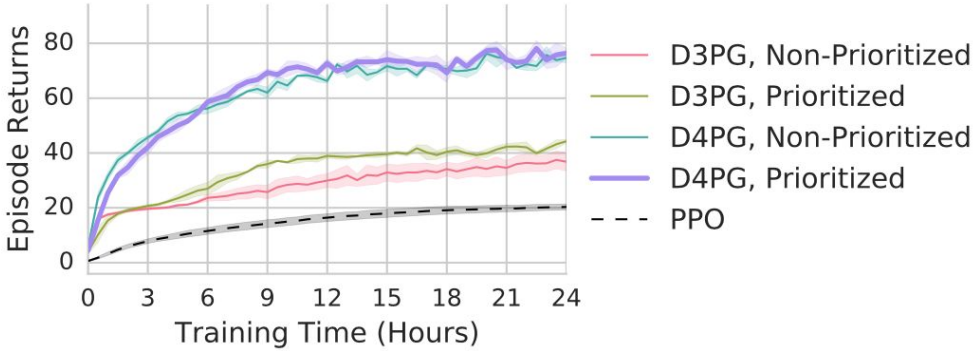


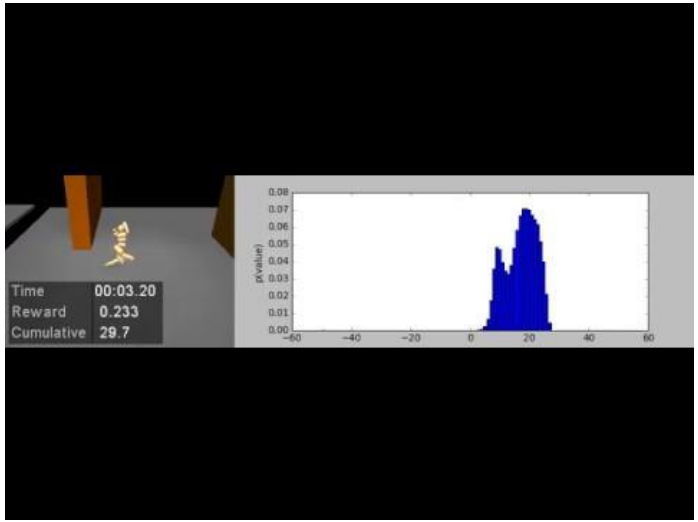
(Lillicrap et al., 2018)

d4pg

Adds distributed and distributional to ddpg.

$$\begin{aligned}\nabla_\theta \rho^\pi(\theta) &= \mathbb{E}_{d^\pi(\mathbf{o})}\left[\nabla_{\mathbf{a}'}\mathbf{Q}^\pi(\mathbf{o},\mathbf{a}')|_{\mathbf{a}'=\pi_\theta(\mathbf{o})}\nabla_\theta\pi_\theta(\mathbf{o})\right] \\ &= \mathbb{E}_{d^\pi(\mathbf{o})}\left[\nabla_{\mathbf{a}'}\mathbb{E}\left[\mathbf{Z}^\pi(\mathbf{o},\mathbf{a}')\right]|_{\mathbf{a}'=\pi_\theta(\mathbf{o})}\nabla_\theta\pi_\theta(\mathbf{o})\right]\end{aligned}$$

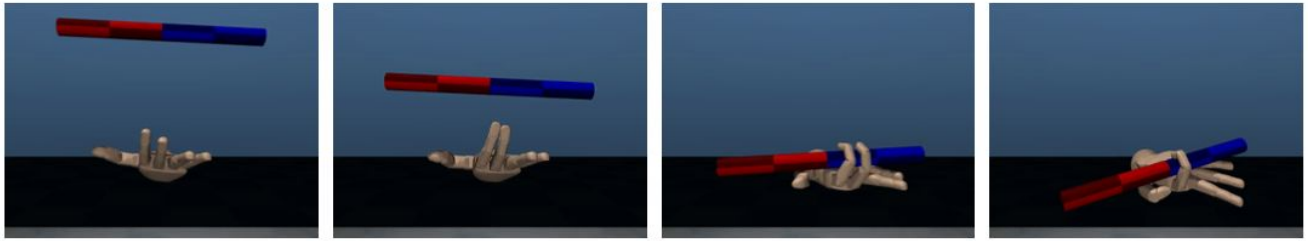




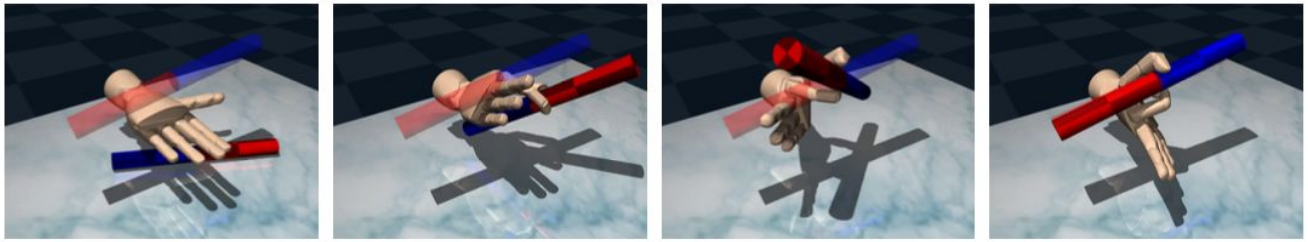
(Gabriel Barth-Maron, Matthew W. Hoffman et al., 2018)

Manipulation results

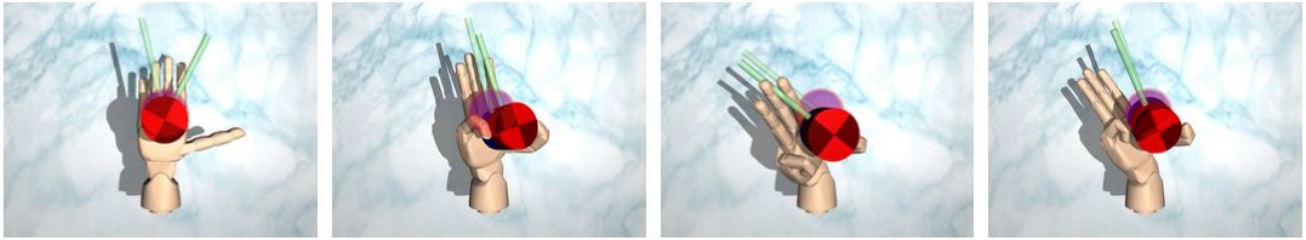
Catch:



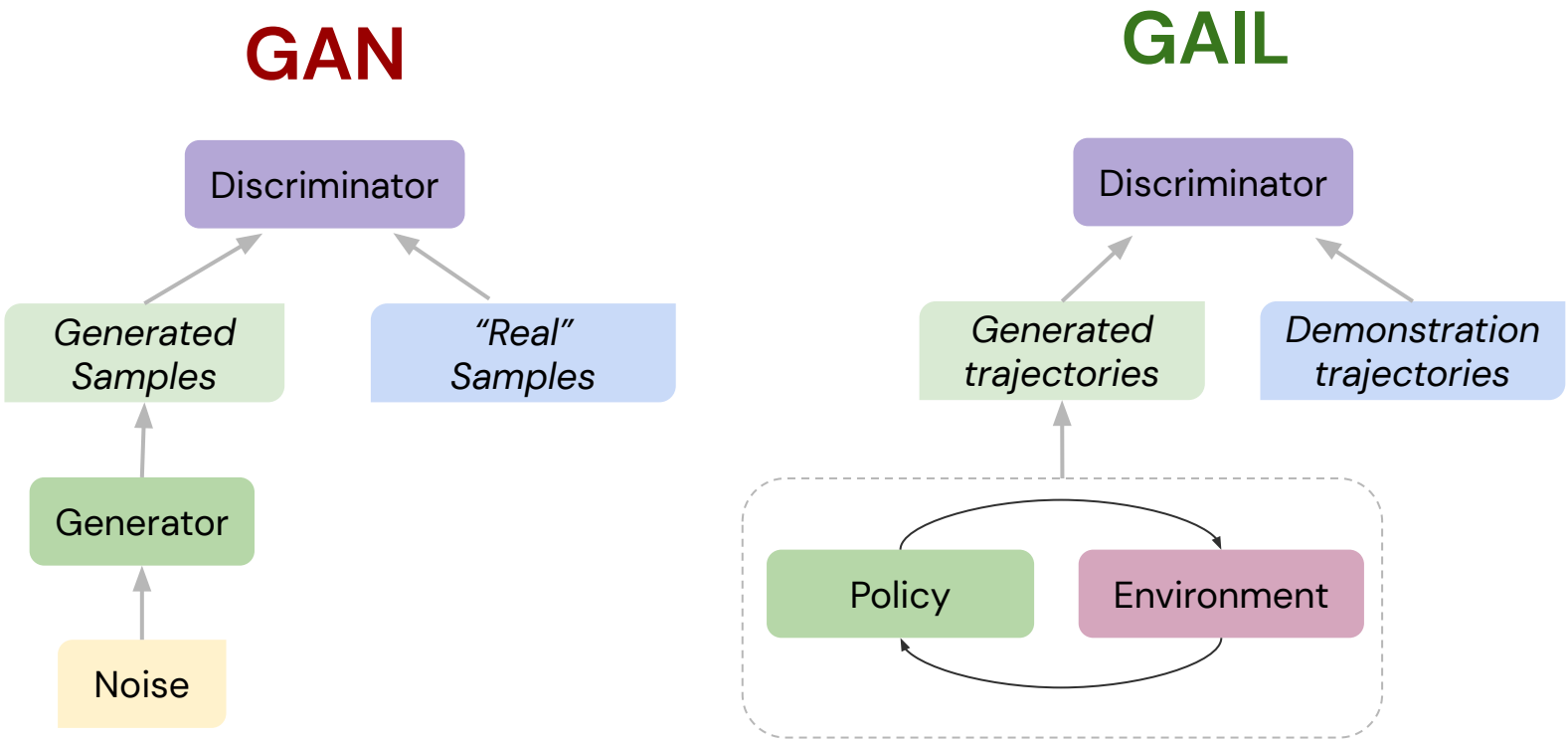
Match moving target:



Pickup and Orient:



Generative Adversarial Imitation Learning (GAIL)

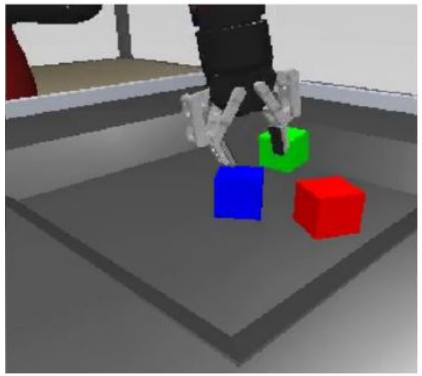


(Ian Goodfellow et al., 2014)

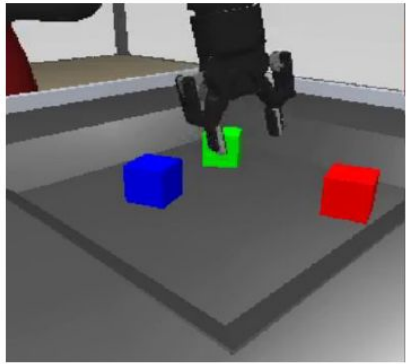
(Jonathan Ho & Stefano Ermon, 2019)

Task relevant adversarial imitation learning (TRAIL) with d4pg

Expert appearance



Agent appearance

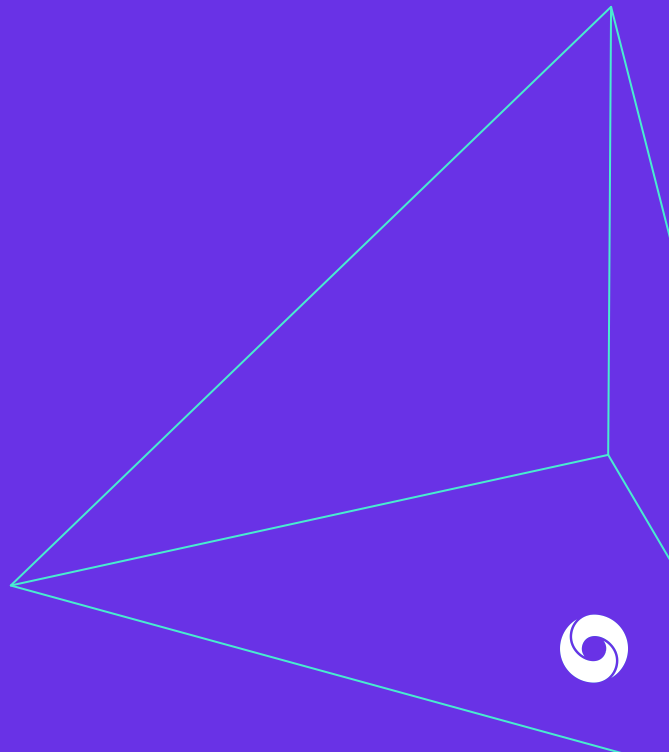


(Zolna et al., 2019)

DeepMind

7

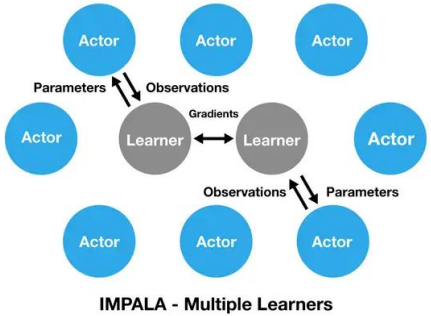
TRPO
ACER
PPO
Impala
MPO



Distributed policy gradient variants

Slow down the policy and use some form of off-policy correction

MPO uses replay, while others use queues and clipped importance sampling



IMPALA - Multiple Learners

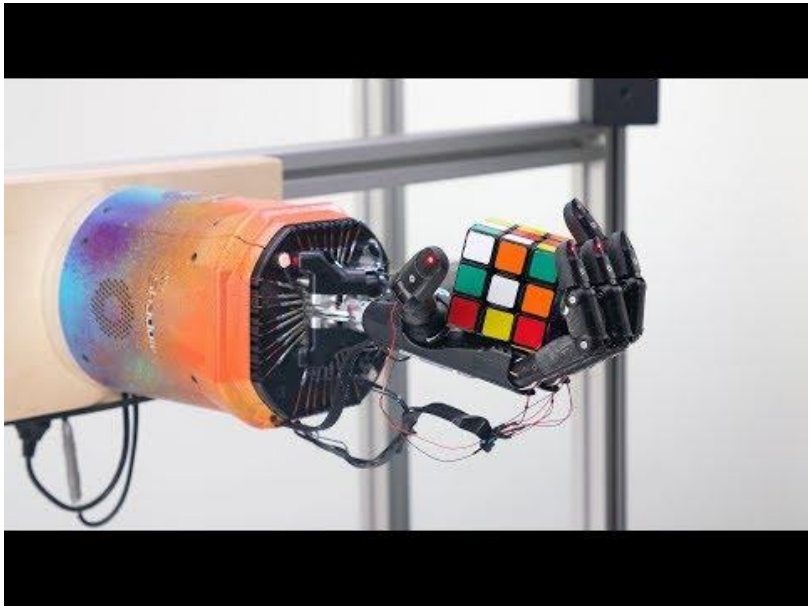
$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$$

$$\mathbb{E}_{d^\pi(\mathbf{o})\pi(\mathbf{a}|\mathbf{o})}\left[\exp\left(\frac{\mathbf{Q}^\pi(\mathbf{o},\mathbf{a})}{\eta}\right)\log \boldsymbol{\pi_\theta}(\mathbf{a}|\mathbf{o})\right] + \alpha\left(\epsilon - KL(\boldsymbol{\pi}(\mathbf{a}|\mathbf{o})||\boldsymbol{\pi_\theta}(\mathbf{a}|\mathbf{o}))\right)$$

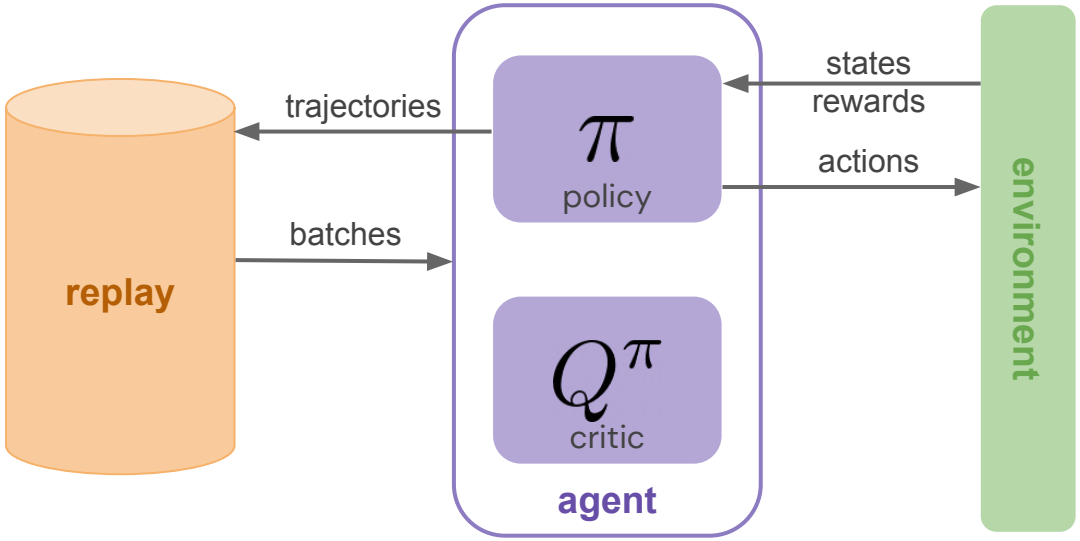
$$\begin{aligned}\underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \le \delta\end{aligned}$$

$$\begin{aligned}\widehat{g}_t^{\text{acer}} \quad = \quad & \bar{\rho}_t \nabla_{\phi_\theta(x_t)} \log f(a_t|\phi_\theta(x))[Q^{\text{ret}}(x_t,a_t) - V_{\theta_v}(x_t)] \\ & + \mathop{\mathbb{E}}_{a\sim\pi}\left(\left[\frac{\rho_t(a)-c}{\rho_t(a)}\right]_+ \nabla_{\phi_\theta(x_t)} \log f(a_t|\phi_\theta(x))[Q_{\theta_v}(x_t,a) - V_{\theta_v}(x_t)]\right)\end{aligned}$$

On-policy methods like PPO shine in simulation



MPO



MPO+D4PG agent
(Some) loss components

MPO policy optimization

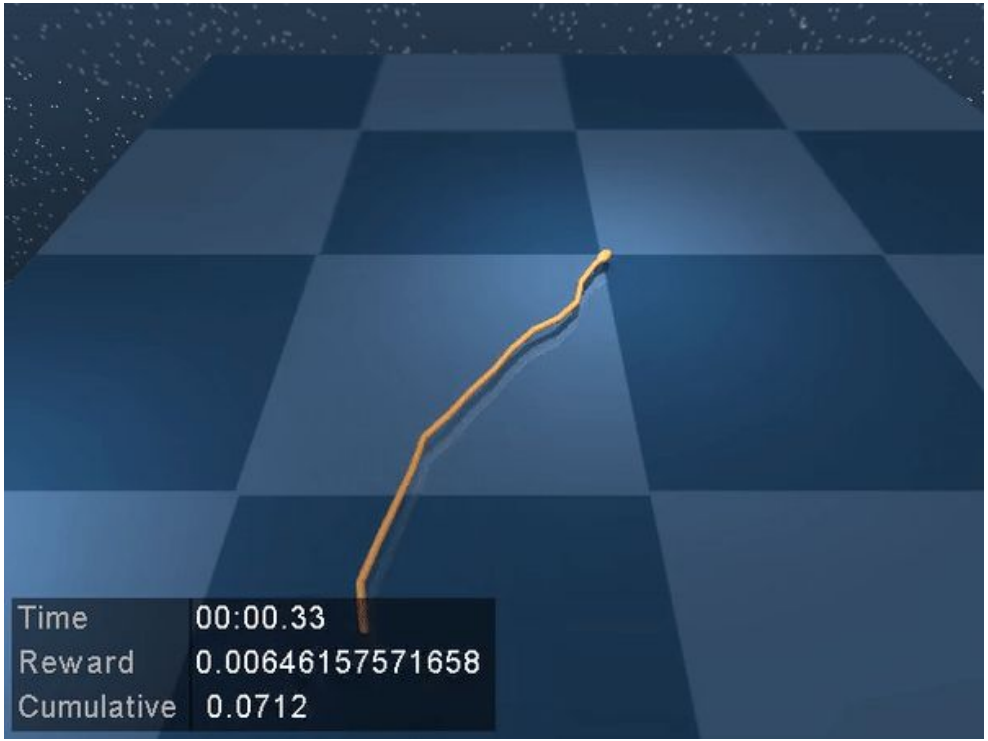
$$\mathbb{E}_{d^{\pi}(\mathbf{o})\pi(\mathbf{a}|\mathbf{o})}\left[\exp\left(\frac{\mathbf{Q}^{\pi}(\mathbf{o},\mathbf{a})}{\eta}\right)\log \boldsymbol{\pi_{\theta}}(\mathbf{a}|\mathbf{o})\right] + \alpha\left(\epsilon - KL(\boldsymbol{\pi}(\mathbf{a}|\mathbf{o})||\boldsymbol{\pi_{\theta}}(\mathbf{a}|\mathbf{o}))\right)$$

data-fit: policy gradient theorem

KL-regularization

MPO demo

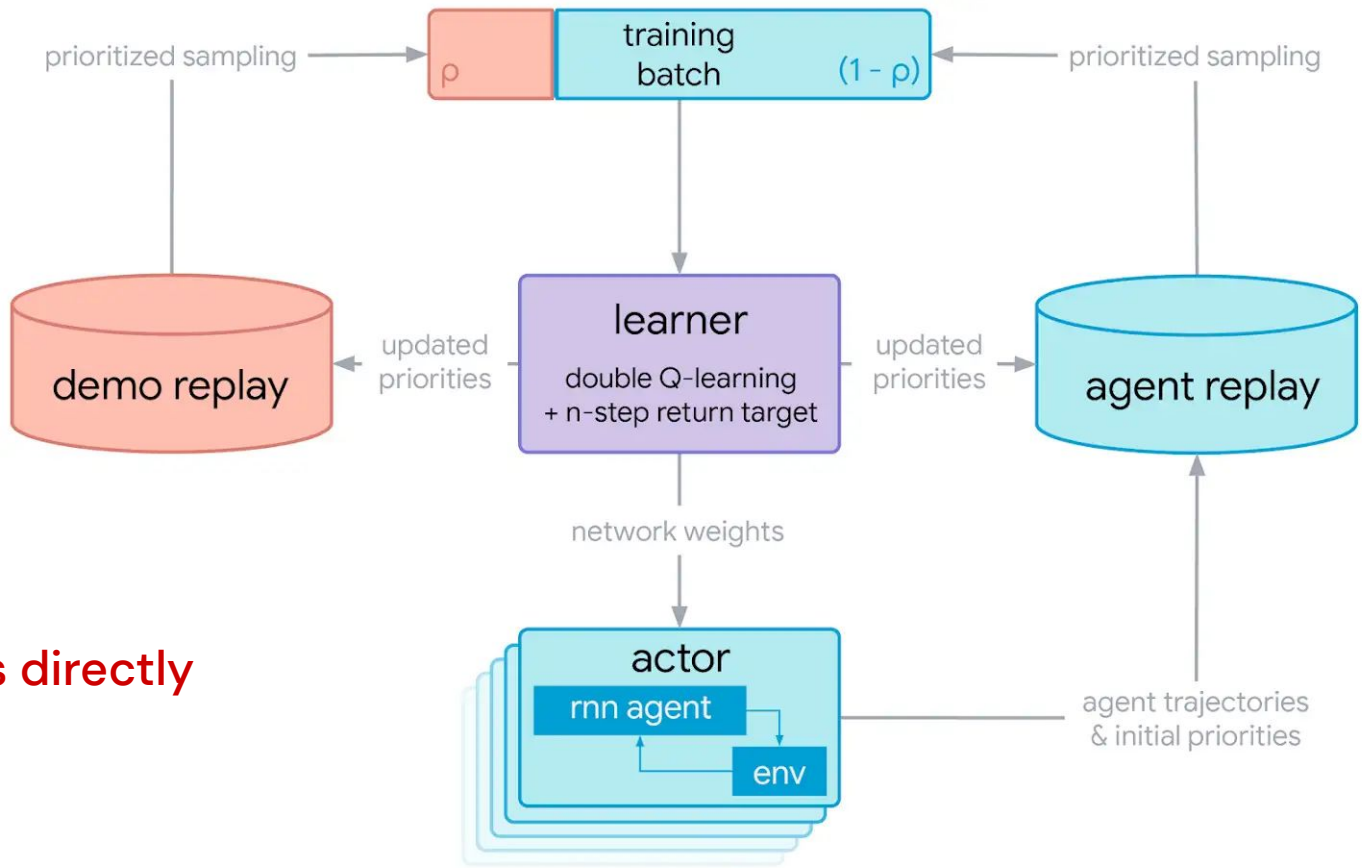
- 15 dim action space
- dynamics
- model-free, from scratch



DeepMind

8 R2D3

R2D3



Learns from sensors directly

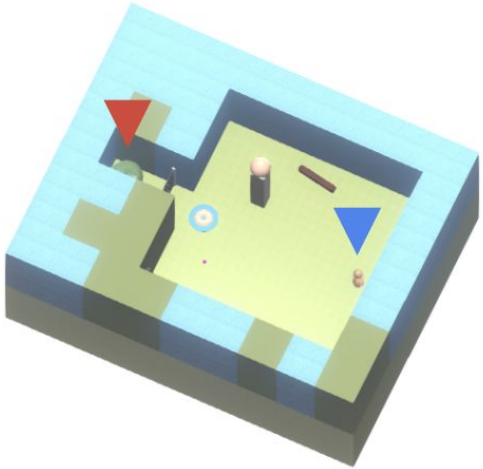
Sparse rewards

Partial observability

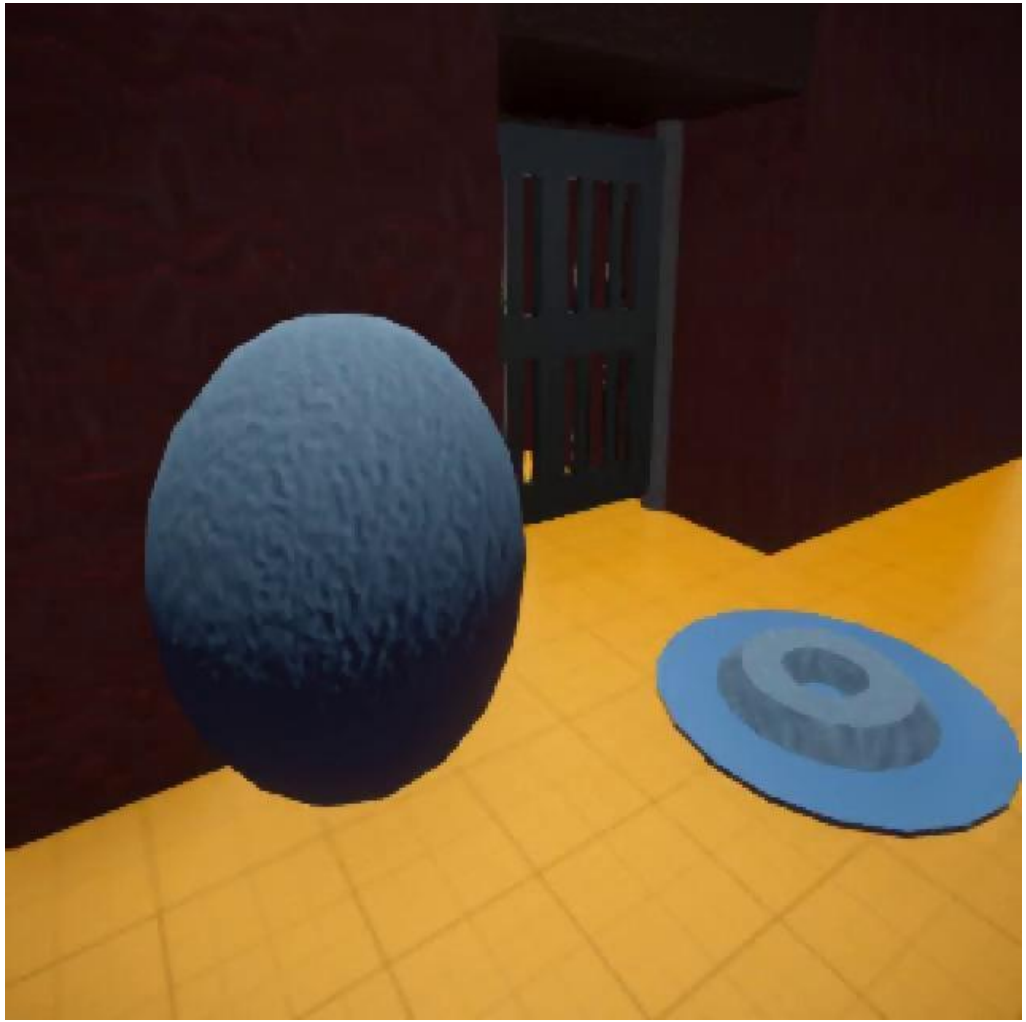
Highly variable initial conditions

Tom Paine, Caglar Gulcehre et al 2019

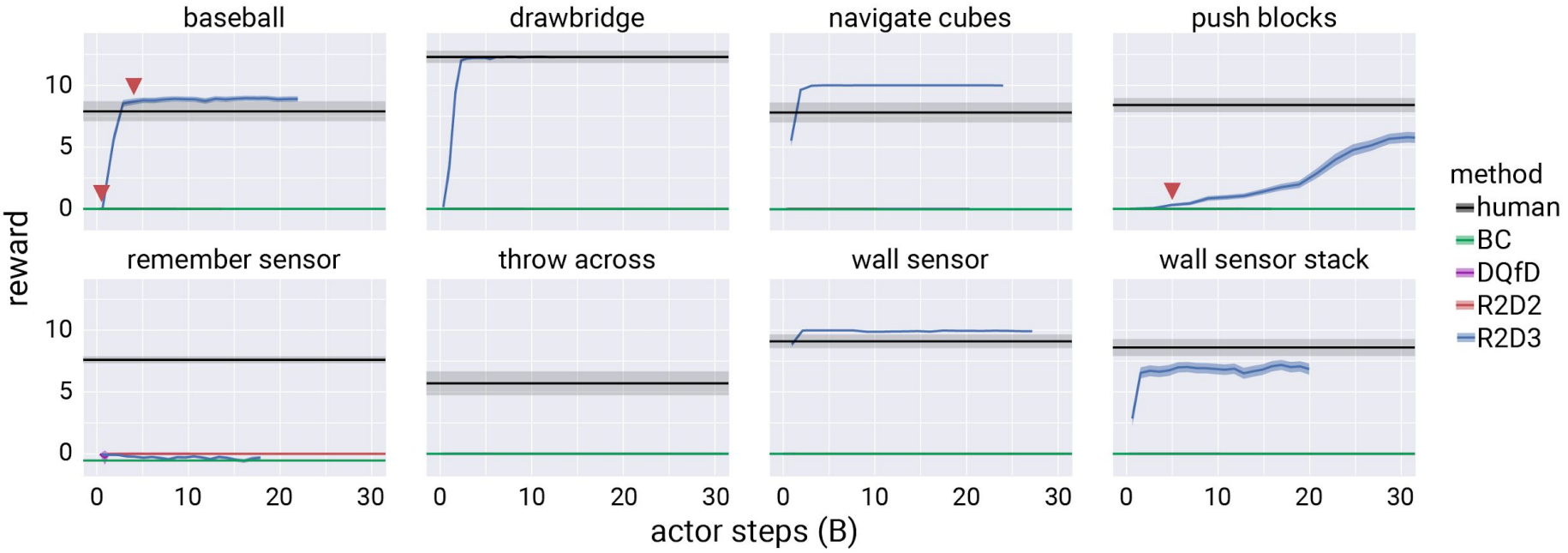
High-level steps necessary to solve the Baseball task



baseball



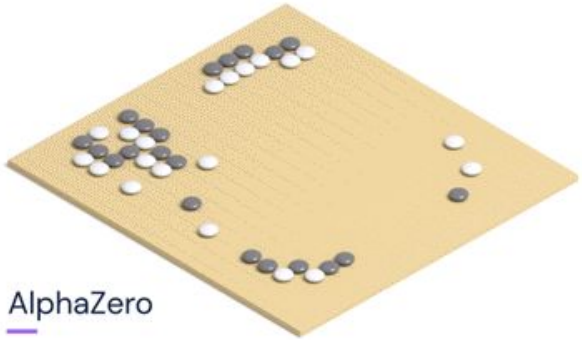
R2D3 vs baseline methods on Hard Eight

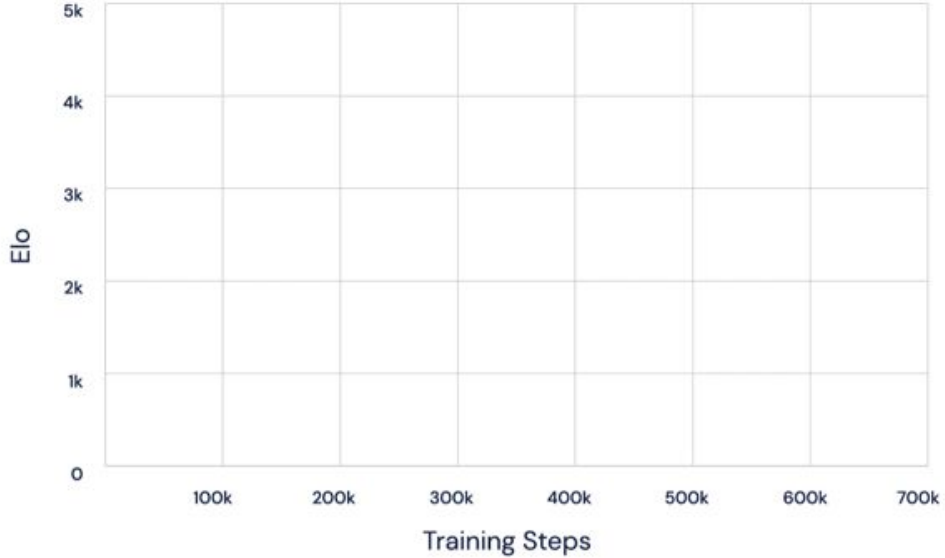


(Tom Paine, Caglar Gulcehre et al 2019)

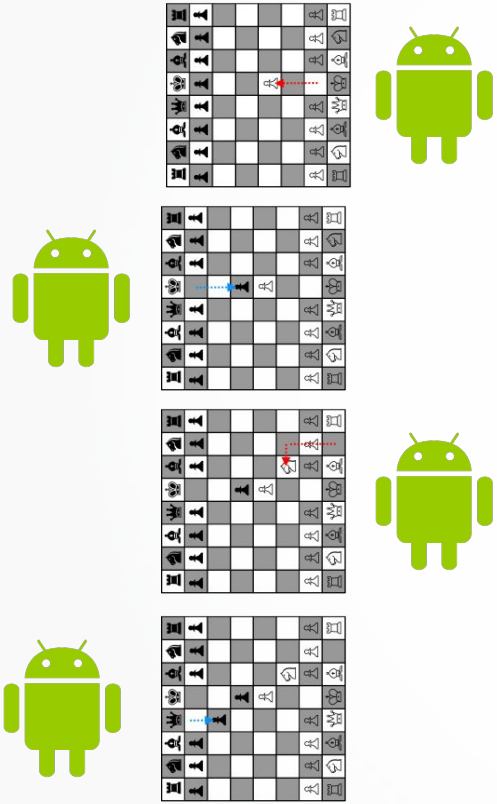
DeepMind

Applications of RL

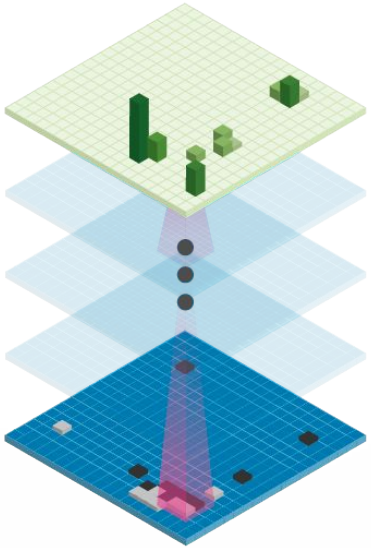

AlphaZero


5k
4k
3k
2k
1k
0
Elo
100k
200k
300k
400k
500k
600k
700k
Training Steps

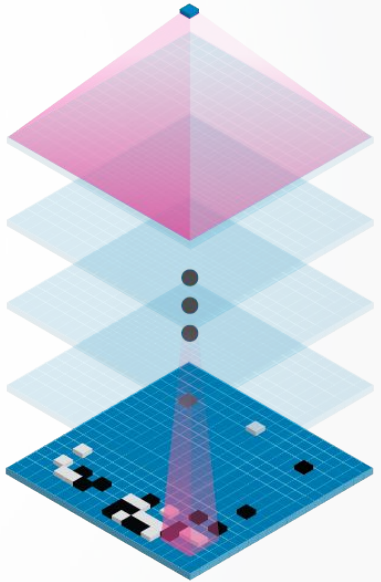
Policy and Value Nets trained by self-play



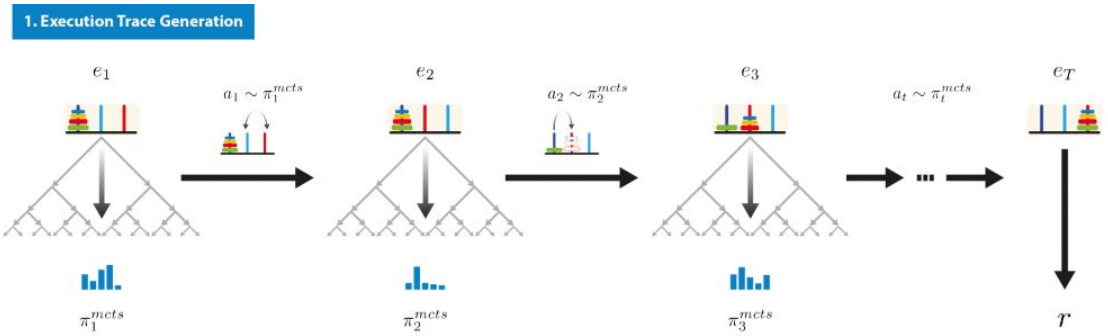
Policy Network returns
Probability distribution over moves
Selects most likely moves

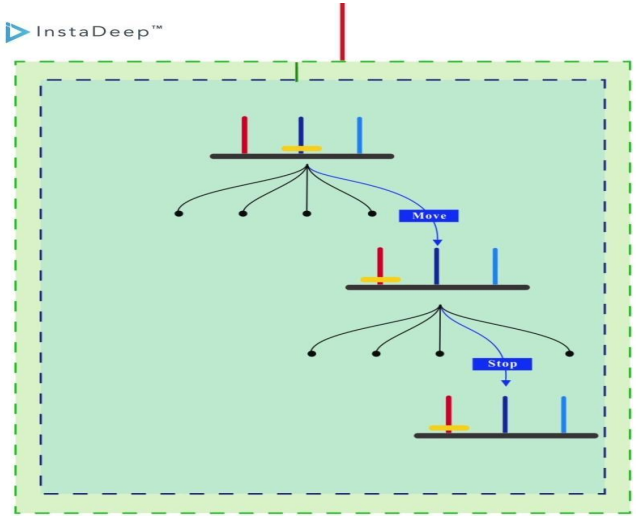


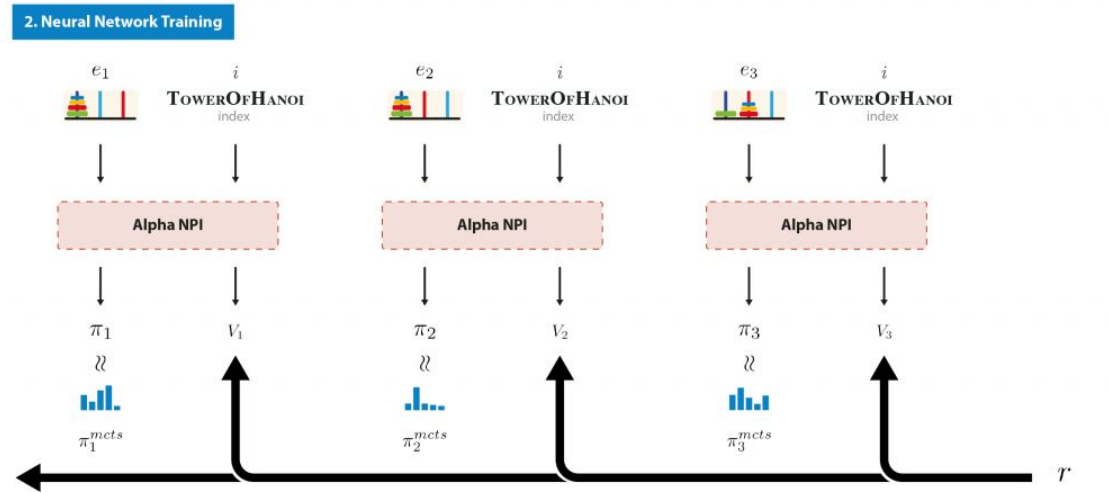
Value Network returns
Probability of winning: 0 (White) - 1 (Black)
Evaluates desirability of a position

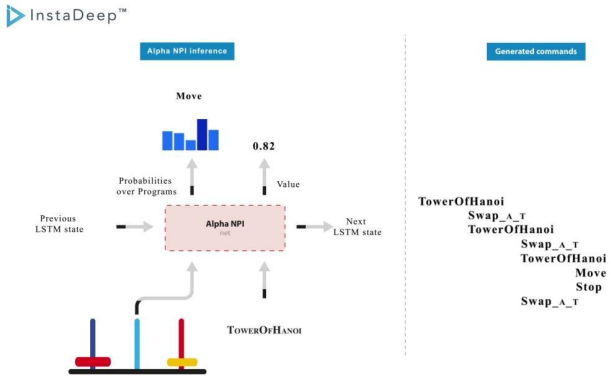


AlphaNPI - An Indaba story









(Pierrot et al., 2019)

StarCraft: A game with challenging complexity

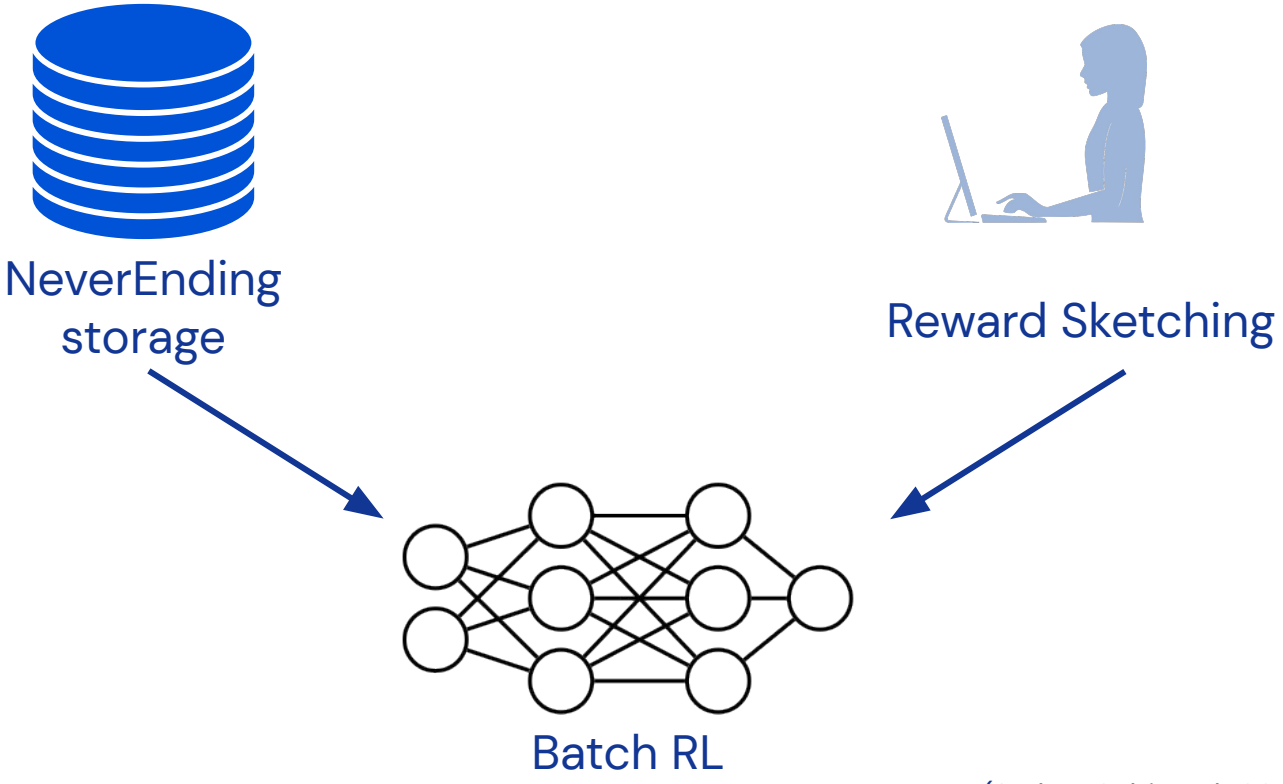
- More complex than Atari or Go
- Imperfect information
- Many players
- Action space ~10^{26}
- 1000's moves per game


Workers: 3/3
Workers: 25/16
Workers: 3/3
14:32
Catalyst LE
AlphaStar
177 /200
945 +2015
758 +873
64
113
940
SUPPLY
MINERALS
GAS
WORKERS
ARMY
APM
PRODUCTION
LiquidTLO
147 /172
335 +1595
442 +1030
61
86
1377

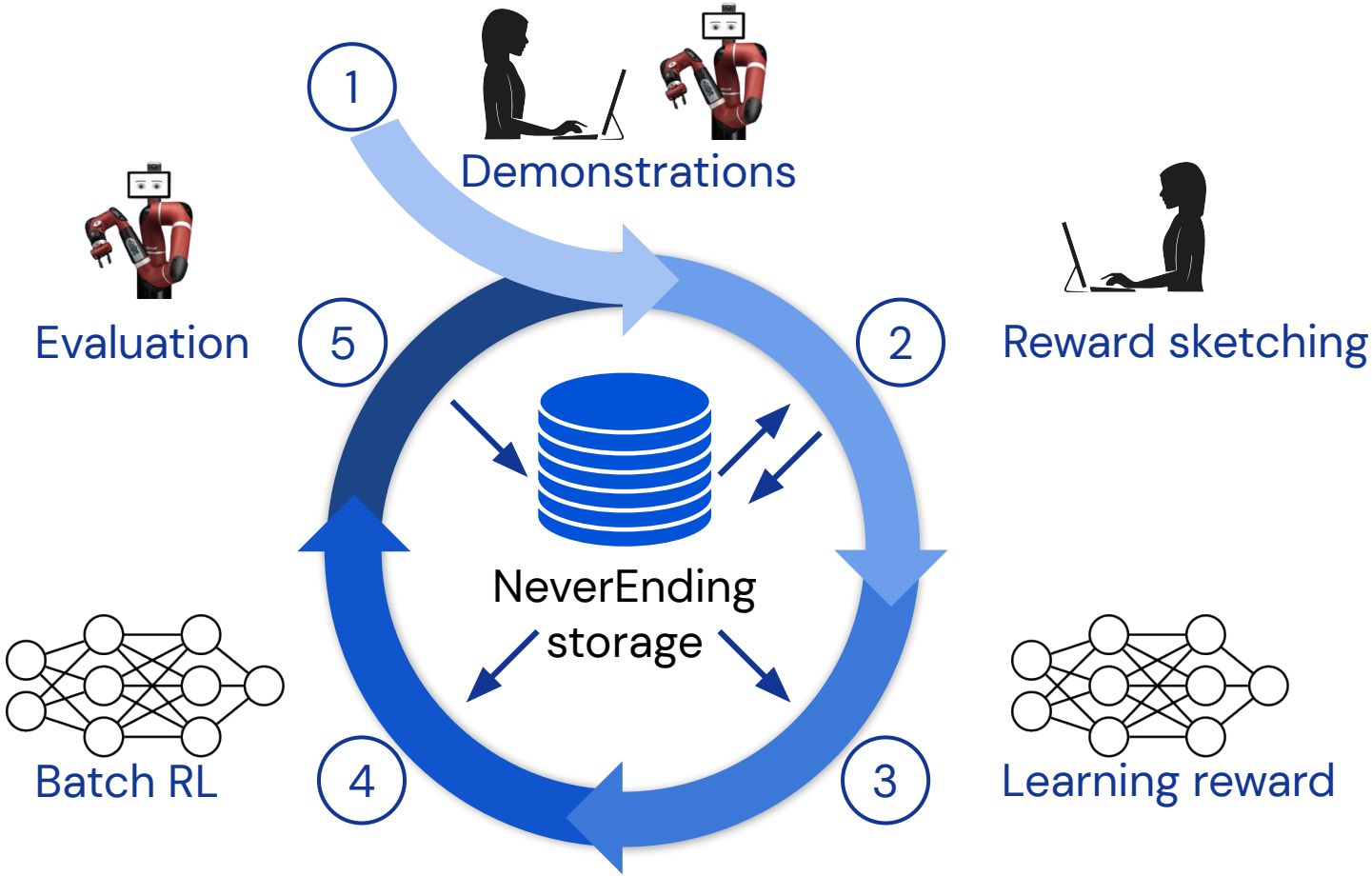
DeepMind

Batch RL

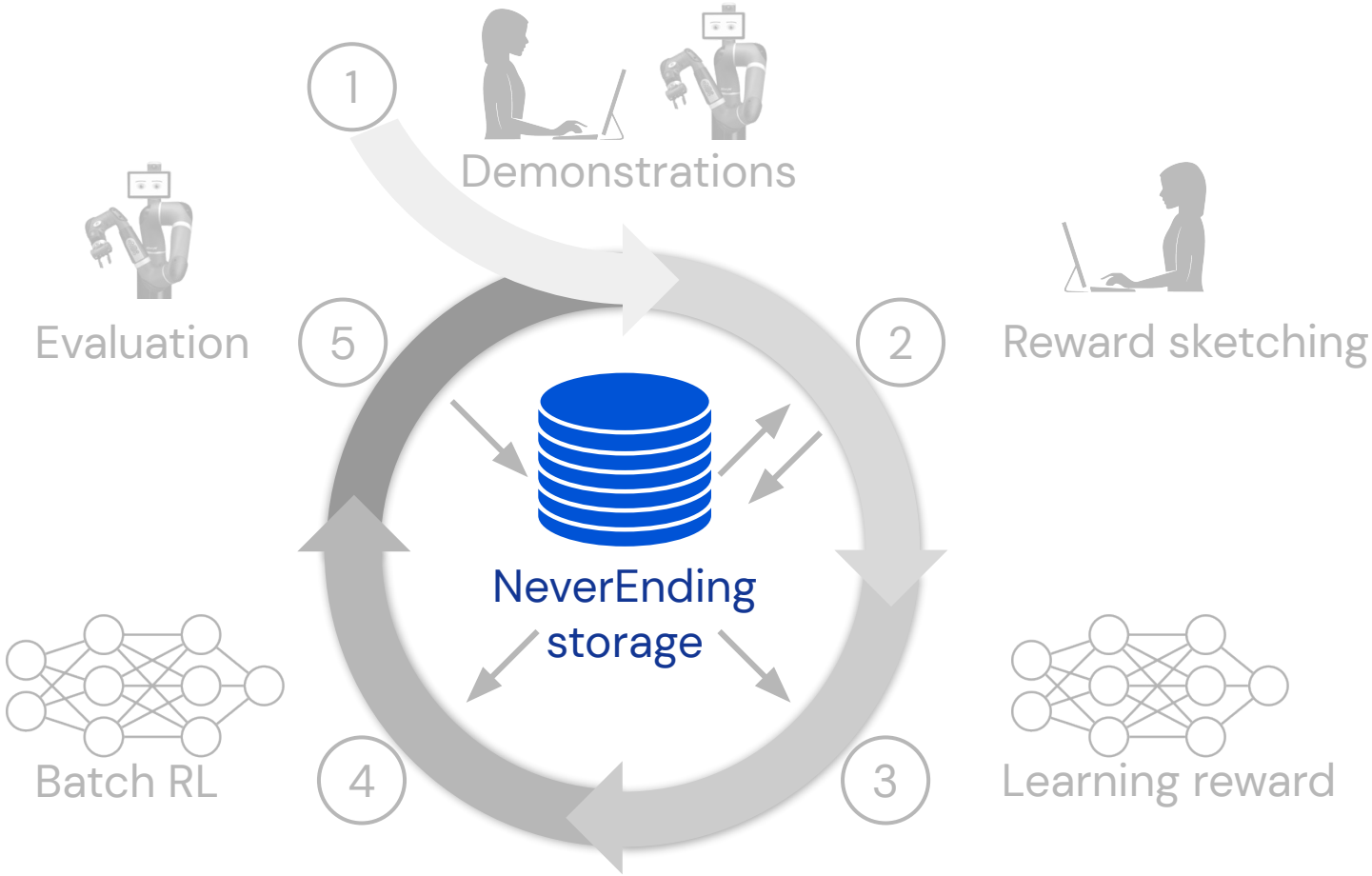
A framework for data-driven robotics



(Serkan Cabi et al., 2019)



(Serkan Cabi et al., 2019)

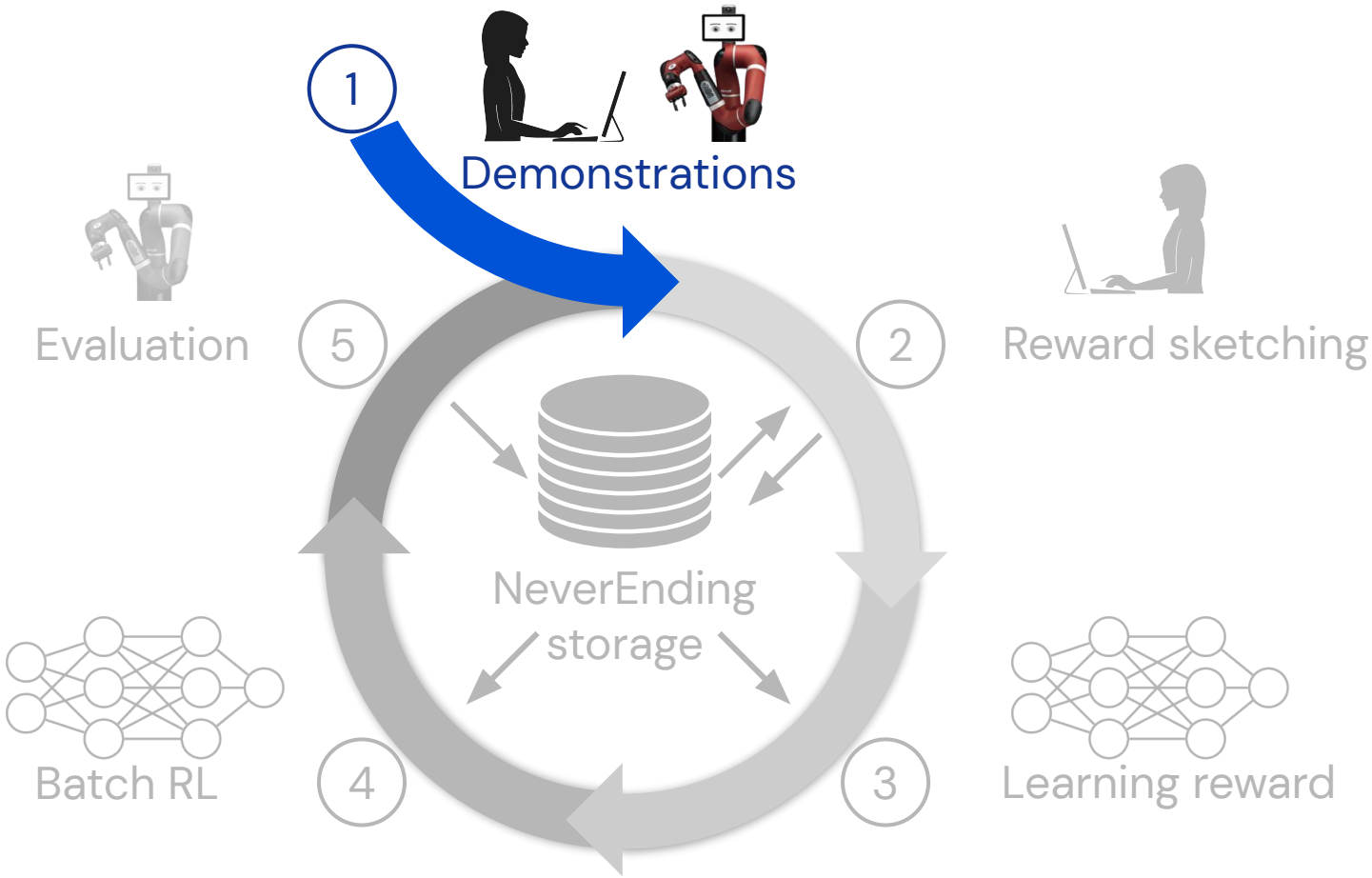


(Serkan Cabi et al., 2019)

NeverEnding storage

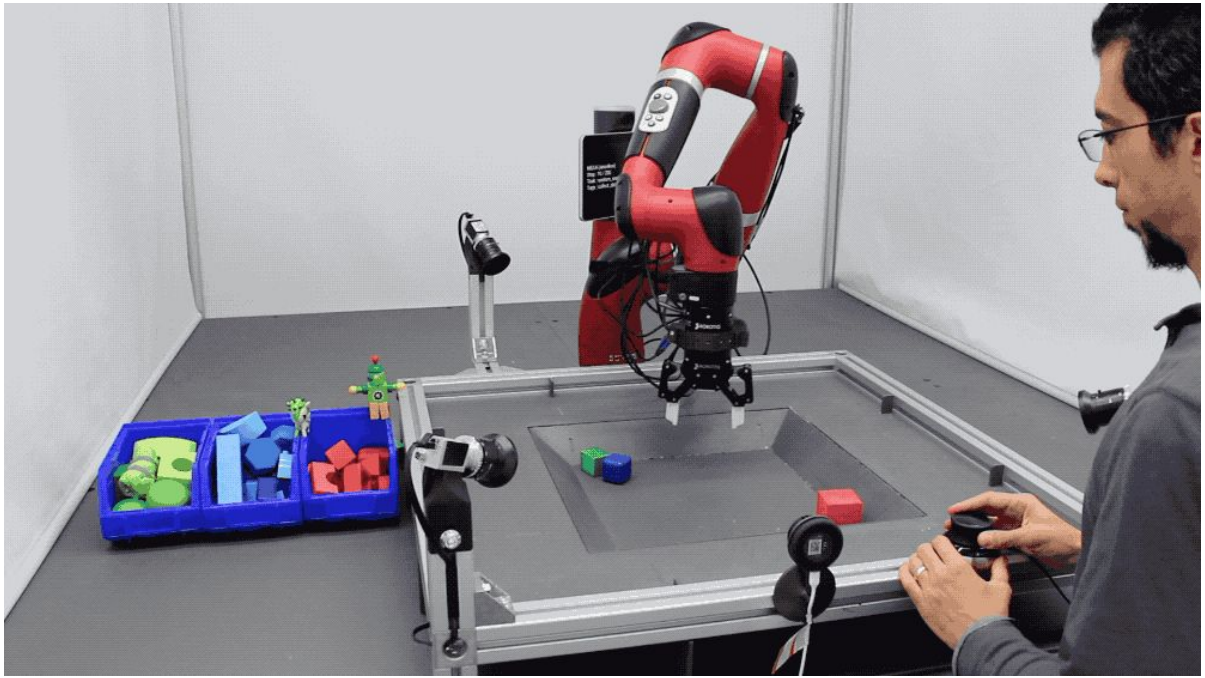
Contains:

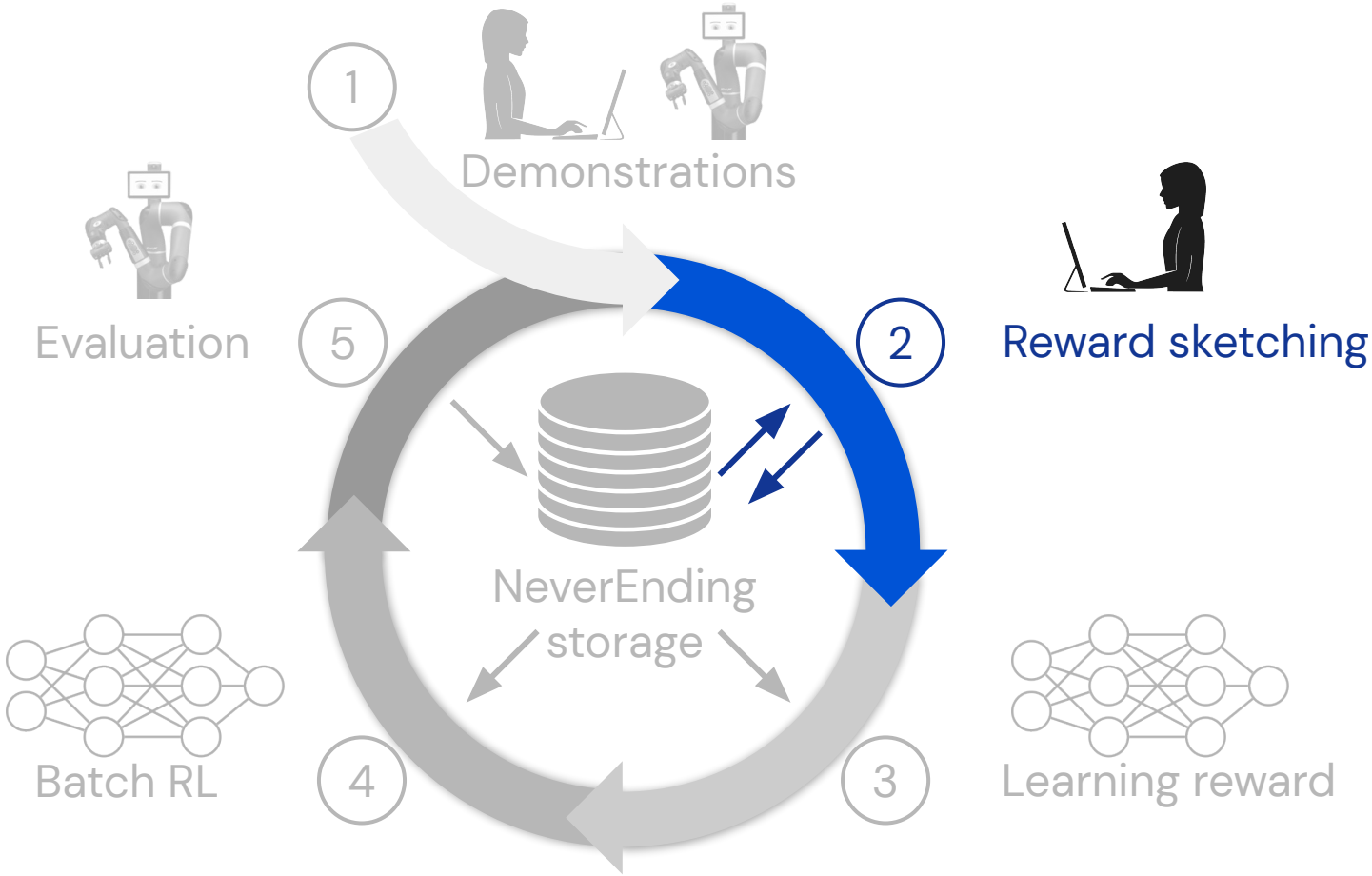
- Different tasks
- Demonstrations
- Agents
- Random policies
- Failed experiments etc.



(Serkan Cabi et al 2019)

Step 1: Demonstrations

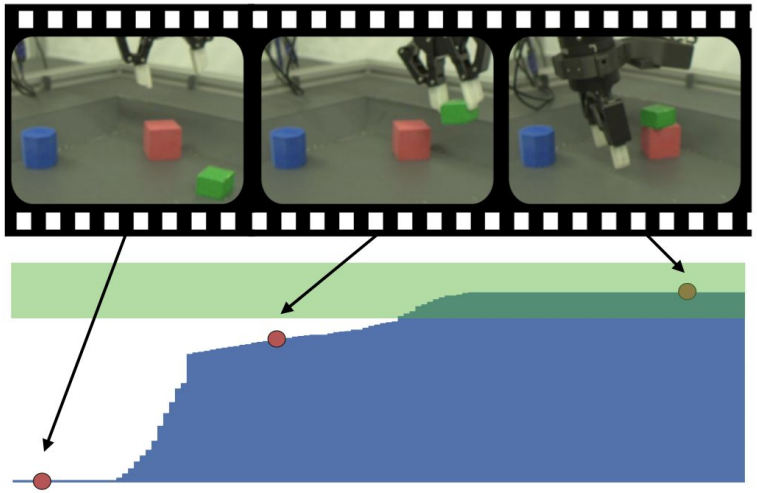


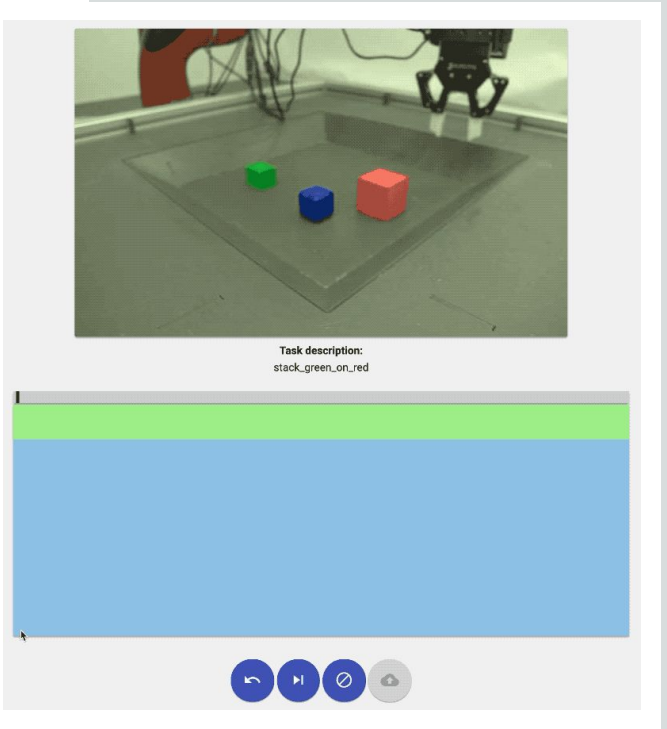


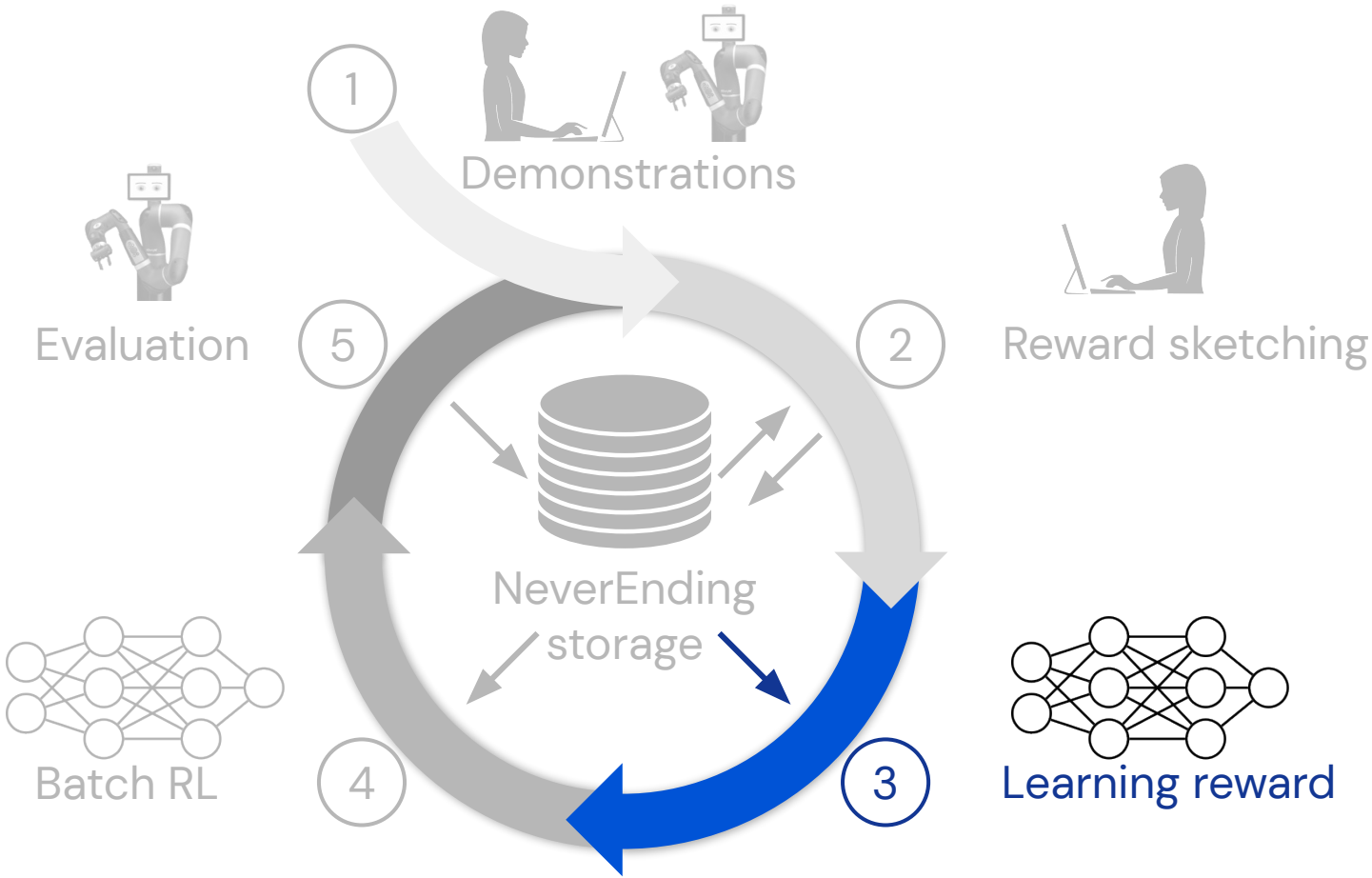
(Serkan Cabi et al., 2019)

Step 2: reward sketching

Task: stack green on red

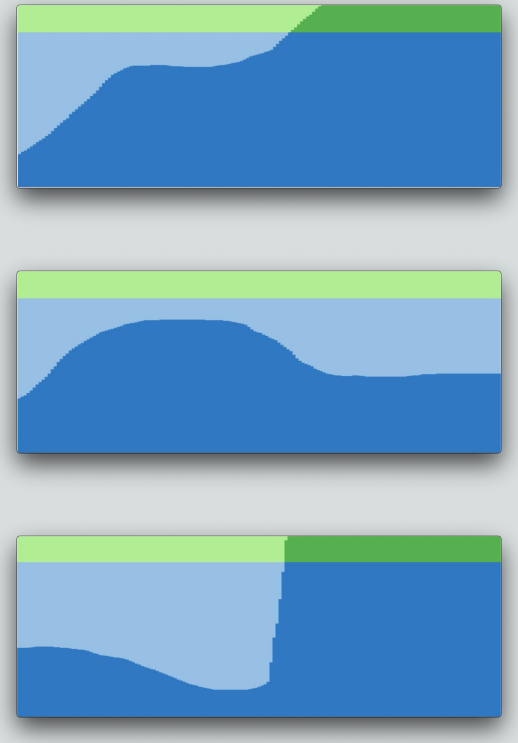


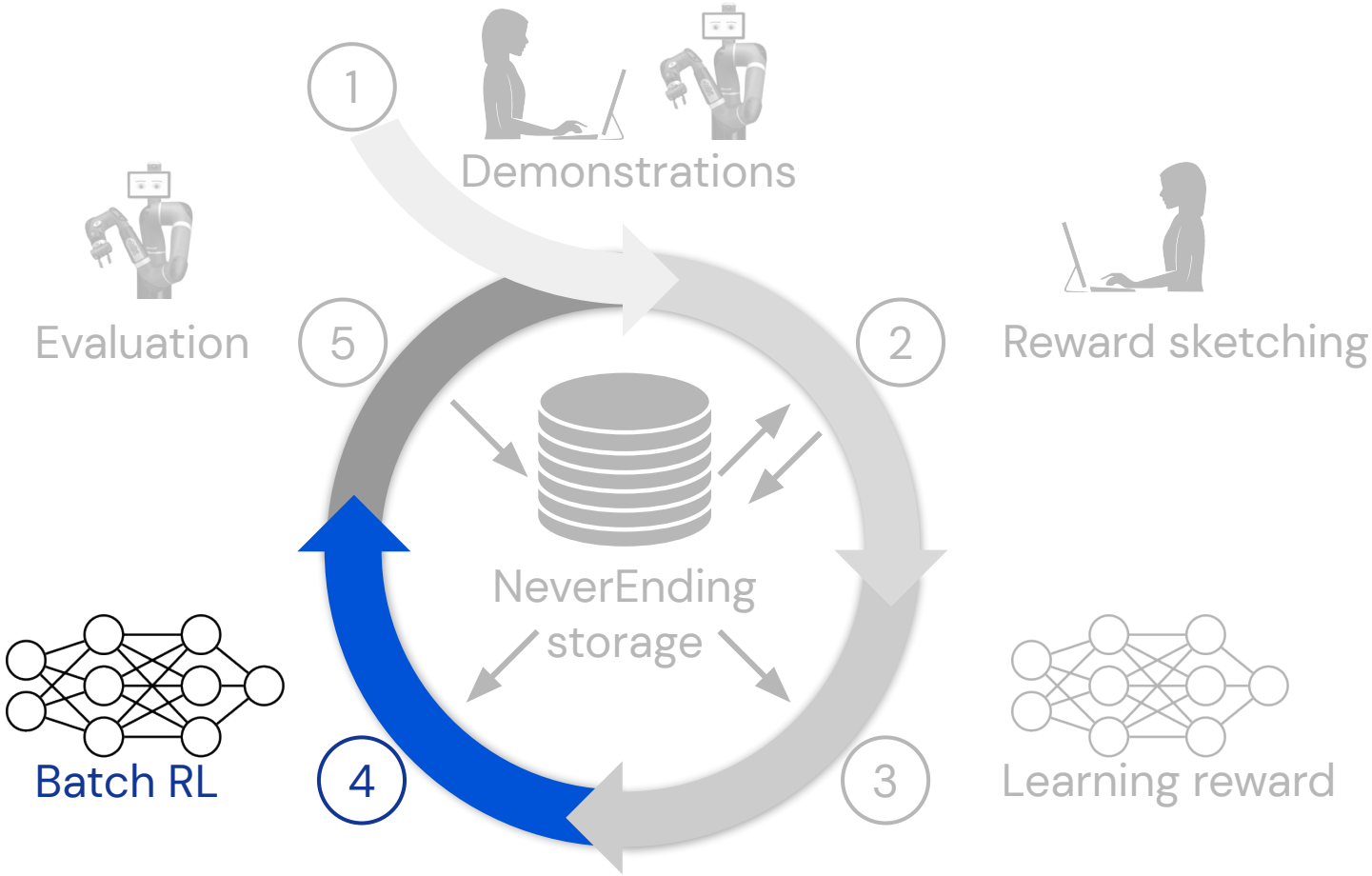




Serkan Cabi et al 2019

Learning reward



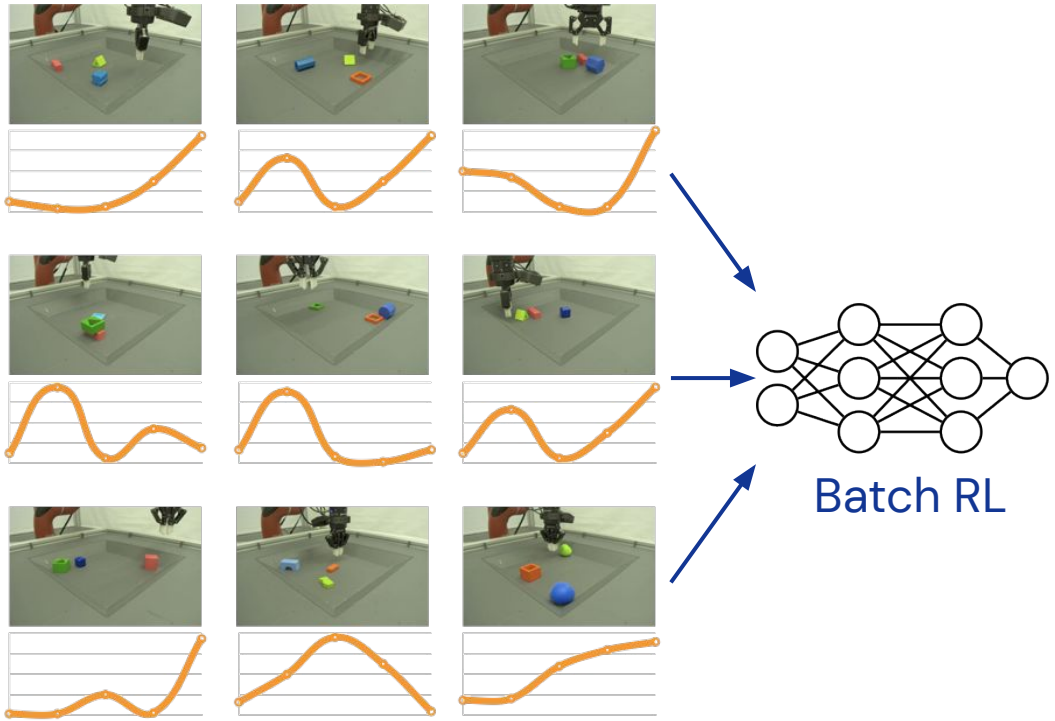


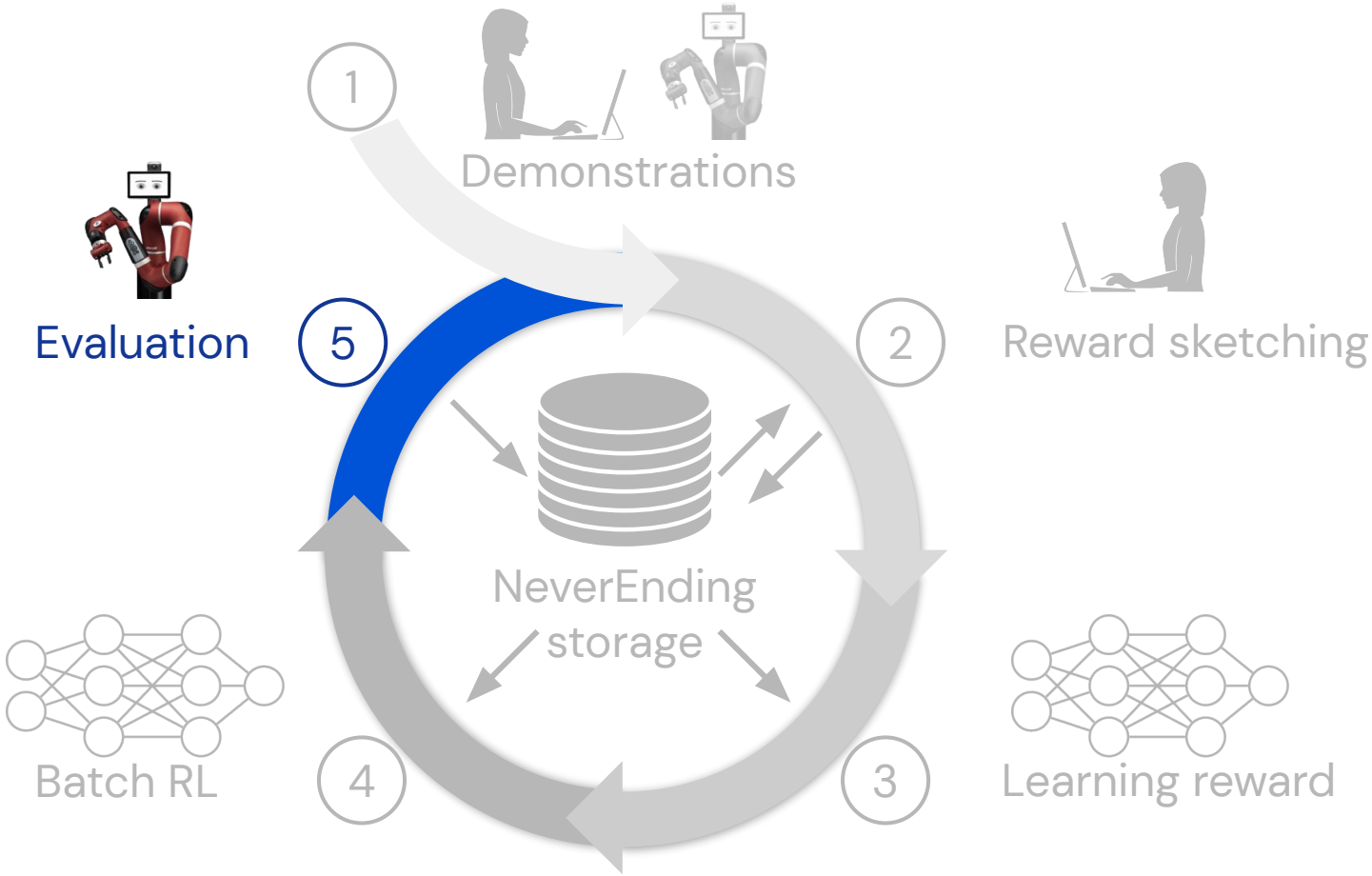
Serkan Cabi et al 2019

Batch RL

Agent:

- D4PG
- Recurrent state
- Uses demonstrations
- 25% task specific in each batch
- Diverse data is crucial!

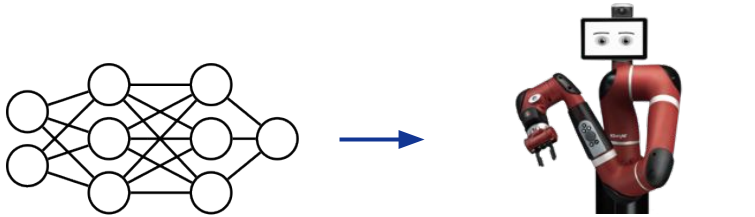




(Serkan Cabi et al., 2019)

Evaluation

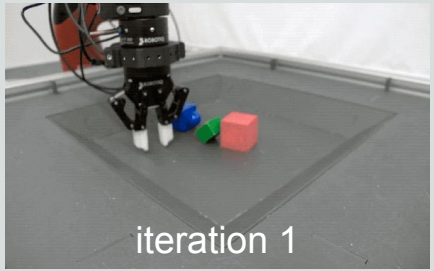
- Execute policy on the robot
- Record episodes to NeverEnding storage

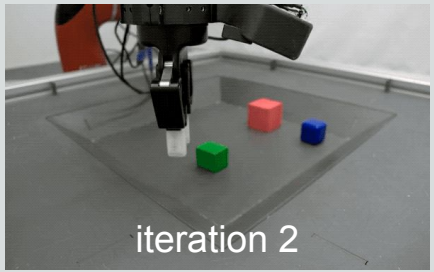


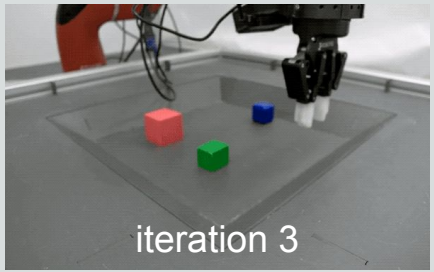
Batch RL

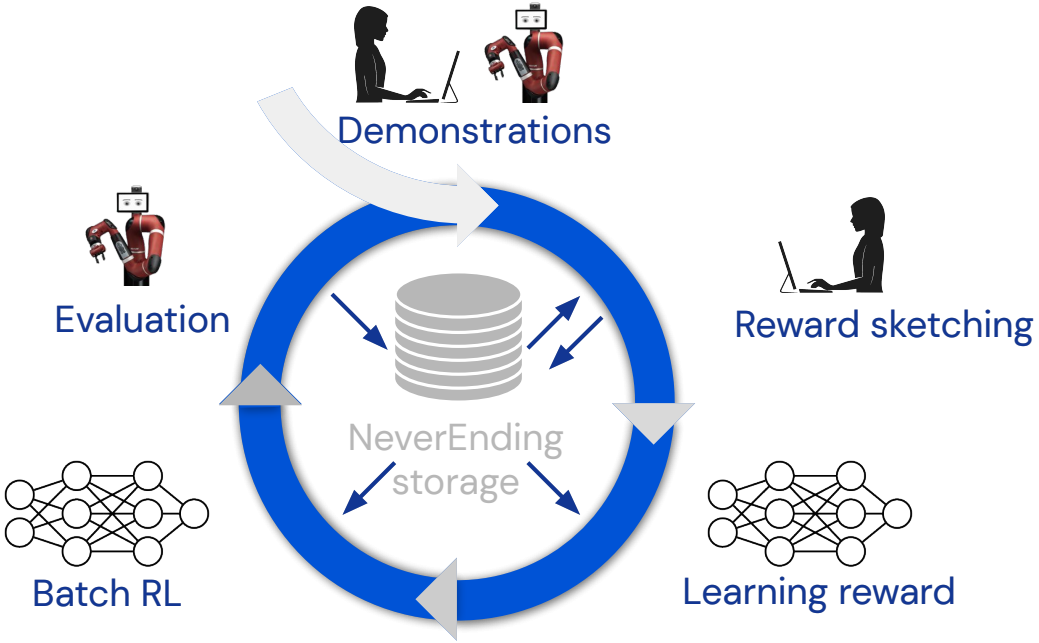
Robot policy

Iterative improvement





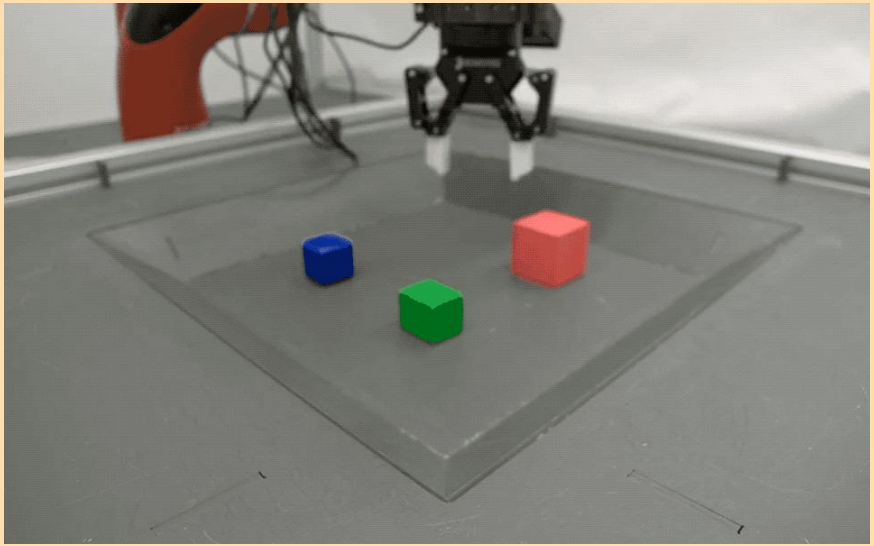




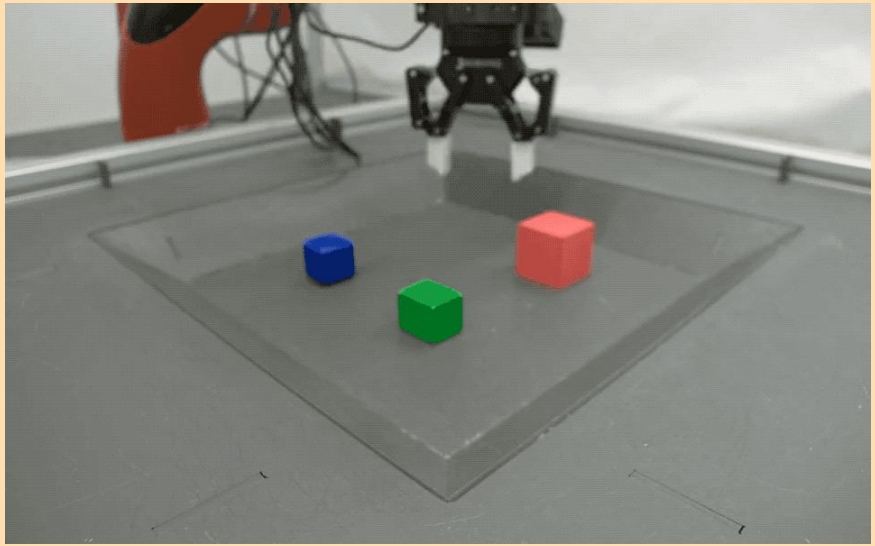
Adversarial robustness

Better than human teleoperators

Human 

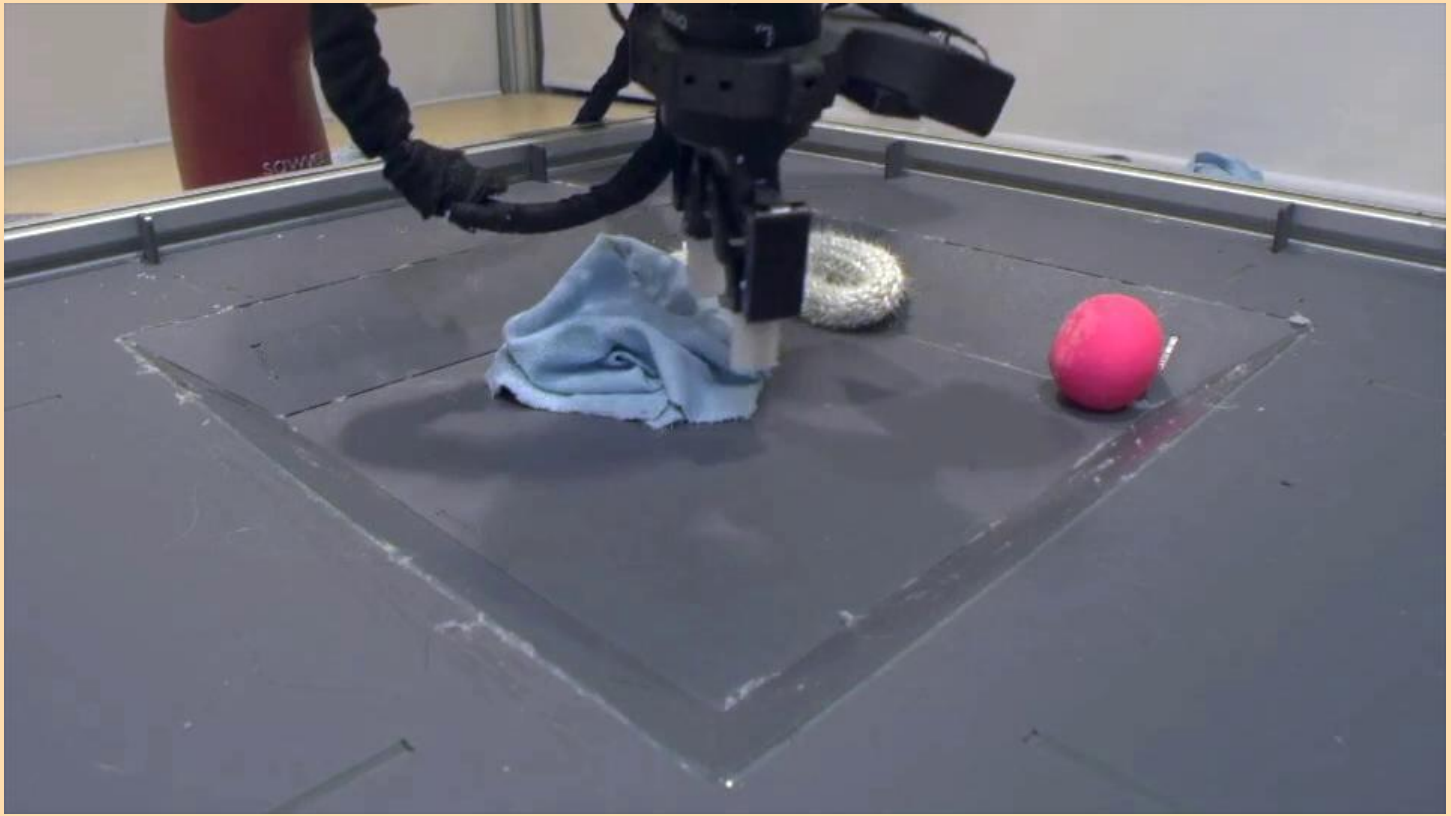


Agent 

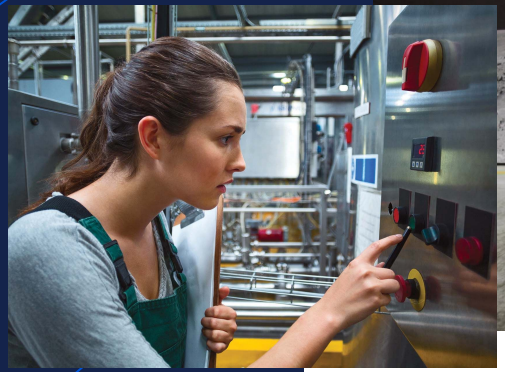


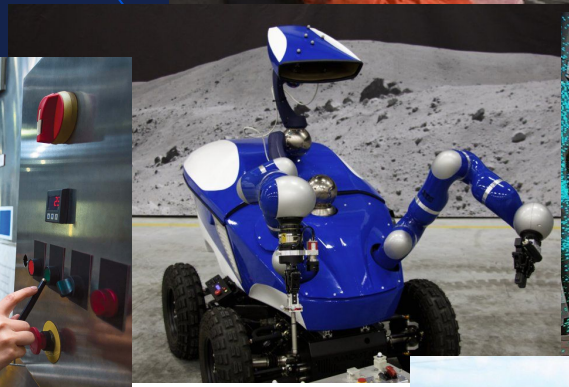
3x speed

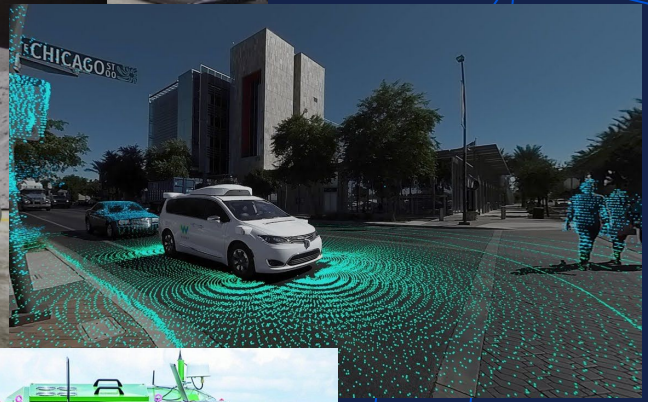
Handles non-scriptable objects



Increased automation







Dialog

Natasha Jacques

[User]: i'm just at work.
[RL - Model avg Psi]: 😐 😑 😕 😒 i'm not sure, but i'm glad to see what you did!
[User]: you're boring.
[RL - Model avg Psi]: 🤨 ✋ 😒 why would you say this? what's up to you?!?!?!?!!!!!!!!!!!
[User]: i'm sorry! I didn't mean to offend you.
[RL - Model avg Psi]: 😔 😞 i'm not sure, and i think i'm going to do this, and i'm sure you'll go!!
[User]: i will go, you're being weird!
[RL - Model avg Psi]: 😠 👊 we are the same, we don't understand that, sir.
[User]: we are not the same. i'm a human and you're a bot.
[RL - Model avg Psi]: ✨ 💗 ✌️ that's really nice to me too.
>

Close Chat and Rate

SEND

Chat History

[User]: hey dude, how's it going?
[RL - KL control Psi]: 😎 😄 honestly, you're going to have to keep to talk about what you think.
[User]: well, i think that it's time for me to take a break!
[RL - KL control Psi]: ✌️ well, you know that you like it.
[User]: yes, i really, really would like one.
[RL - KL control Psi]: today i'm going to have a good time.
[User]: yeah? what are you going to do?
[RL - KL control Psi]: 😚 i don't know what i'm doing, but i'm so happy for you!

Please Rate Your Chat Experience!

Overall, how was the **QUALITY** of the chat?

Very Poor — Very High

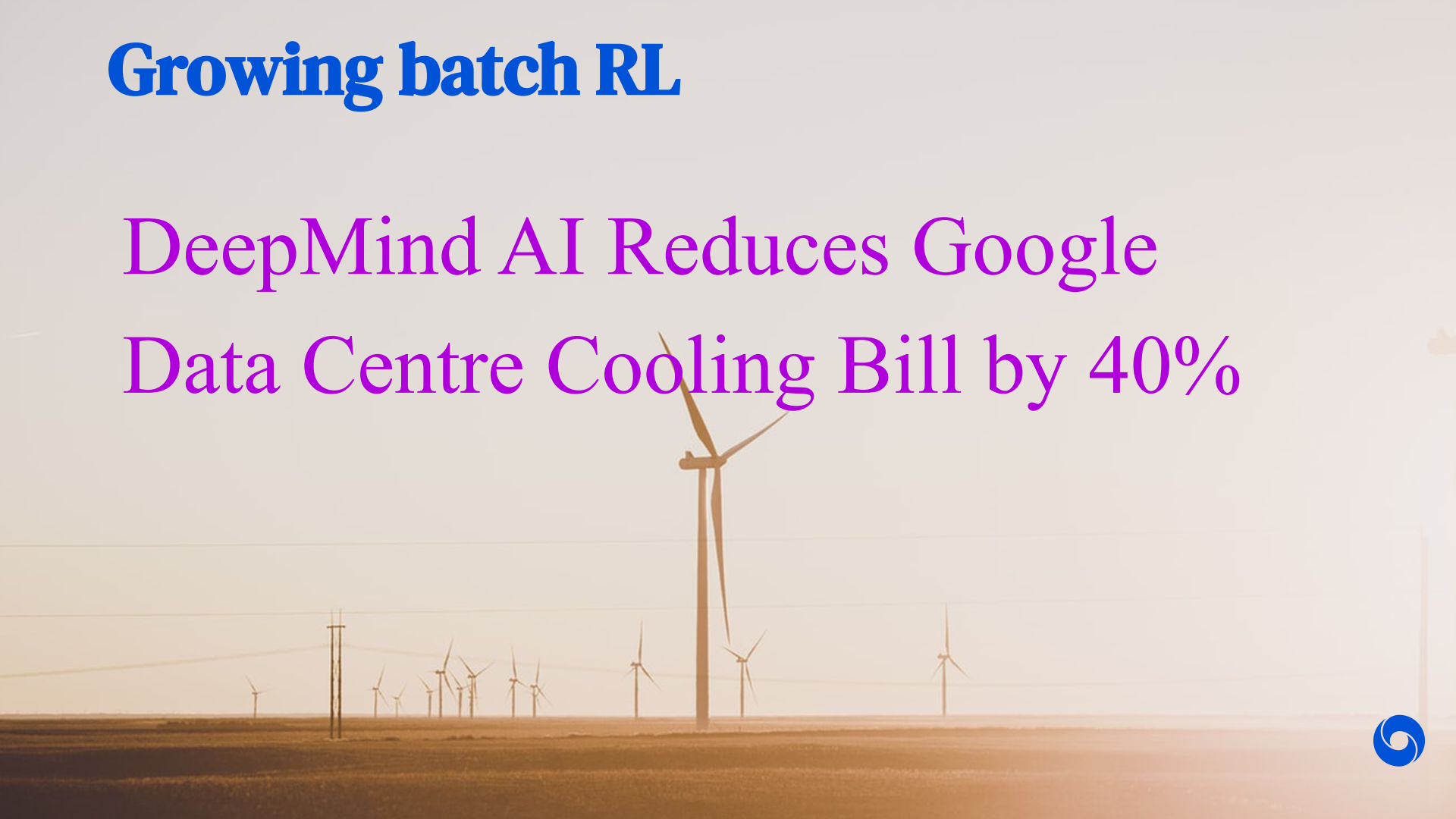
How **DIVERSE (non-repetitive)** were the chat bot's responses?

Not at All — Extremely

How **FLUENT** was the chat bot?

Growing batch RL

DeepMind AI Reduces Google Data Centre Cooling Bill by 40%



David Silver's principles

1. Evaluation drives progress
2. Scalability determines success
3. Generality: Future proof algorithms
4. Trust in the agent's experience
5. Optimise end-to-end
6. Empower the network
7. Control the sensorimotor stream
8. Value functions are knowledge
9. State is subjective
10. Plan (dynamically think ahead) with jumps

DeepMind

Challenges and open questions

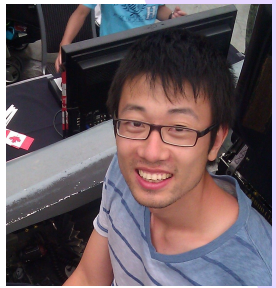
- Reproducibility
- Long horizon, time
- Abstraction, space-time and universality
- Policy evaluation and exploration in growing batch RL
- Distributional RL
- Prioritized replay and self-imitation
- Safety and risk












Thank you!

References

- Wang et al., 2016 Dueling Network Architectures for Deep Reinforcement Learning
- Modayil et al., 2011 Multi-timescale Nexting in a Reinforcement Learning Robot
- Sutton et al., 2000 Policy Gradient Methods for Reinforcement Learning with Function Approximation
- Mnih et al., 2013 Playing Atari with Deep Reinforcement Learning
- Espeholt et al., 2018 IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures
- Yu & Finn et al 2018 One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning
- Paine et al., 2018 One-Shot High-Fidelity Imitation: Training Large-Scale Deep Nets with RL
- Bellemare, Dabney, Munos, 2017 A Distributional Perspective on Reinforcement Learning
- Agrawal et al., 2019 Learning to Generalize from Sparse and Underspecified Rewards
- Bard et al., 2019 The Hanabi Challenge: A New Frontier for AI Research
- Dabney et al., 2018 Implicit Quantile Networks for Distributional Reinforcement Learning
- Lillicrap et al., 2015 Continuous control with deep reinforcement learning
- Barth-Maron et al., 2018 Distributed Distributional Deterministic Policy Gradients
- Ho & Ermon, 2019 Generative Adversarial Imitation Learning
- Goodfellow et al., 2014 Generative Adversarial Networks
- Zolna et al., 2019 Task-Relevant Adversarial Imitation Learning
- OpenAI et al., 2019 Solving Rubik's Cube with a Robot Hand
- Paine et al., 2019 Making Efficient Use of Demonstrations to Solve Hard Exploration Problems
- Pierrot et al., 2019 Learning Compositional Neural Programs with Recursive Tree Search and Planning
- Cabi et al., 2019 A Framework for Data-Driven Robotics