Outline

- Multi-arm Bandits
- RL Successes & Settings
- Tabular example
- Q-Learning and SARSA
- Models
- Function Approximation
- DQN

- Selection of research

- Better/worse RL problems

Generalization and Function Approximation

How does experience in parts of the state space help us act over the entire state space?

- Generalization

How can does function approximation (supervised learning) merge with RL?

- Beg, borrow, and steal

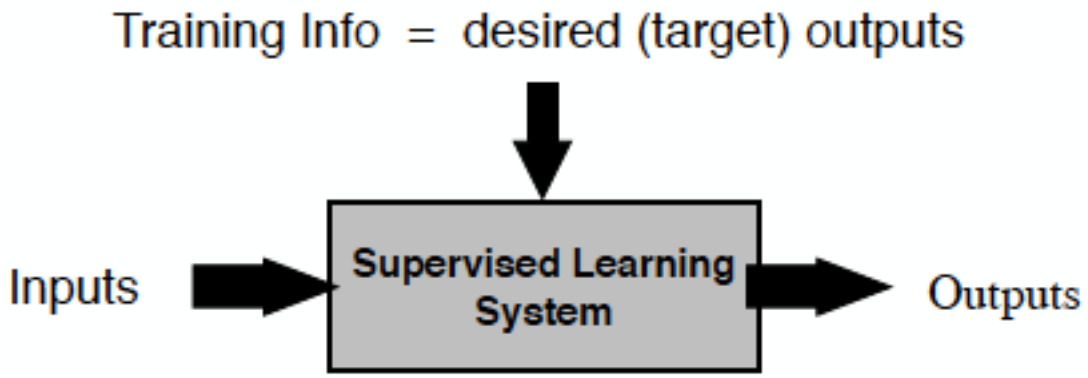
Function approximator convergence?

- Eh

- Instead of lookup table for values of V at time t (V_t), consider some kind of weight vector $\mathbf{w}_t$
- E.g., $\mathbf{w}_t$ could be the weights in a neural network
- Instead of one value (weight) per state, now we update this vector

Insight: Use existing supervised learning methods!

- Training = {X,Y}
- Error = target output – actual output



TD Backups as Training Examples

- Recall the TD(0) backup:

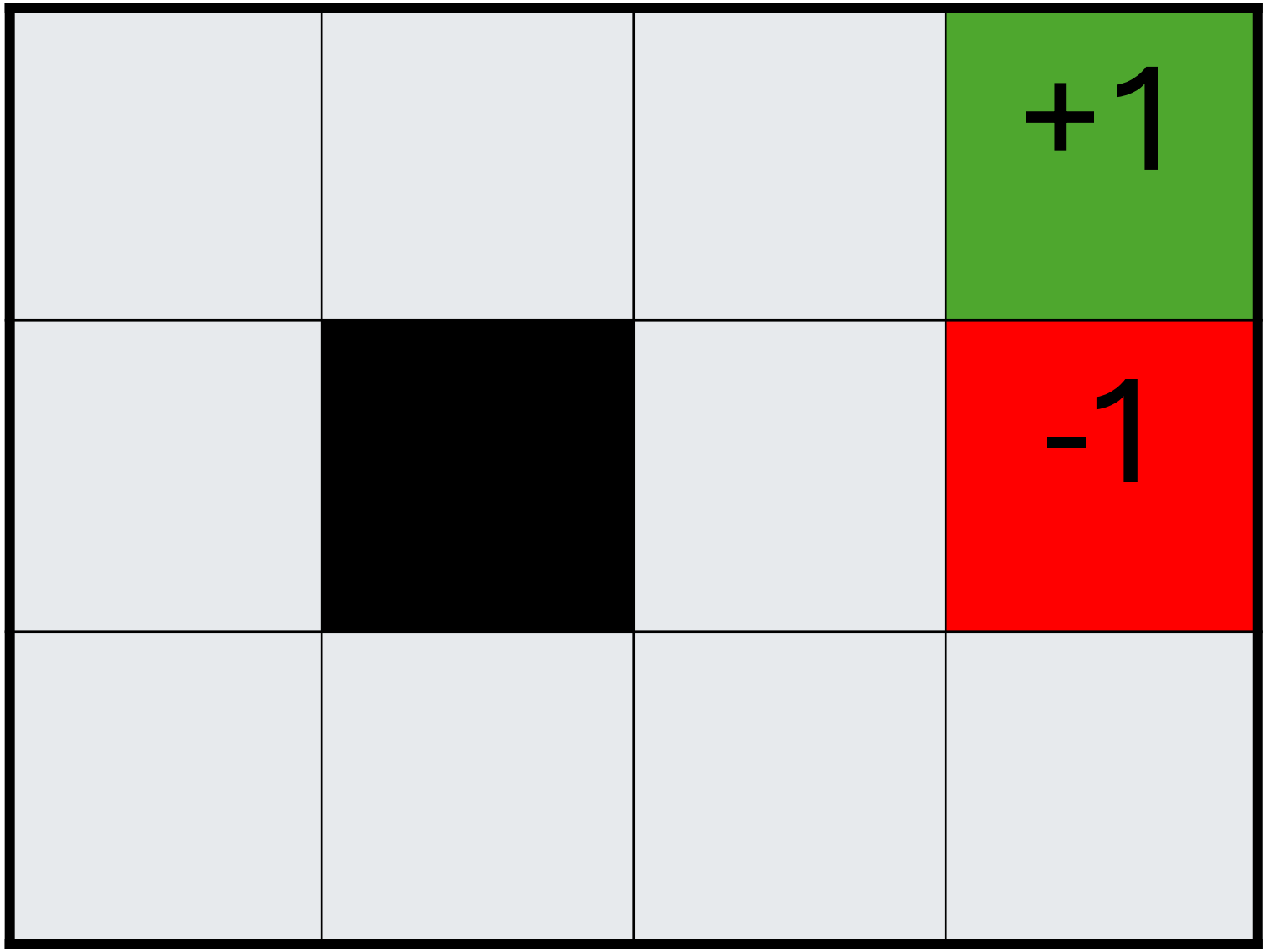
$$V(s_t) \leftarrow V(s_t) + \alpha\Big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big].$$

- As a training example:
  - Input = Features of s_t
  - Target output = $r_{t+1} + \gamma V(s_{t+1})$

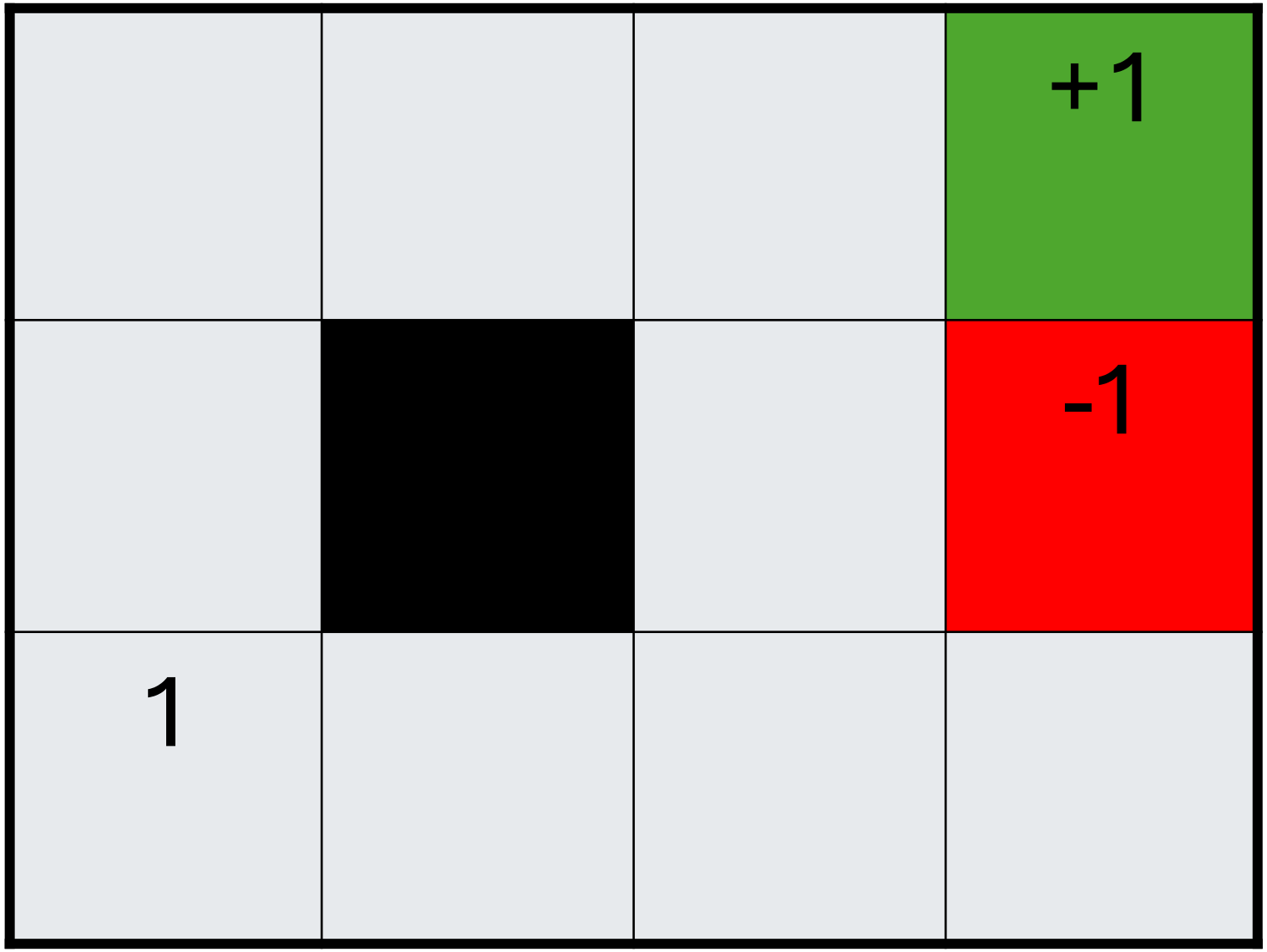
What FA methods can we use?

- In principle, anything!
  - Tile coding
  - Neural Networks
  - Decision trees
  - Multivariate regression
  - Support Vector Machines
  - Gaussian Processes
  - Deep Neural Networks
  - Etc.
- But we normally want to
  - Learn while interacting
  - Handle nonstationarity
  - Not take “too long” or use “too much” memory
  - Etc.

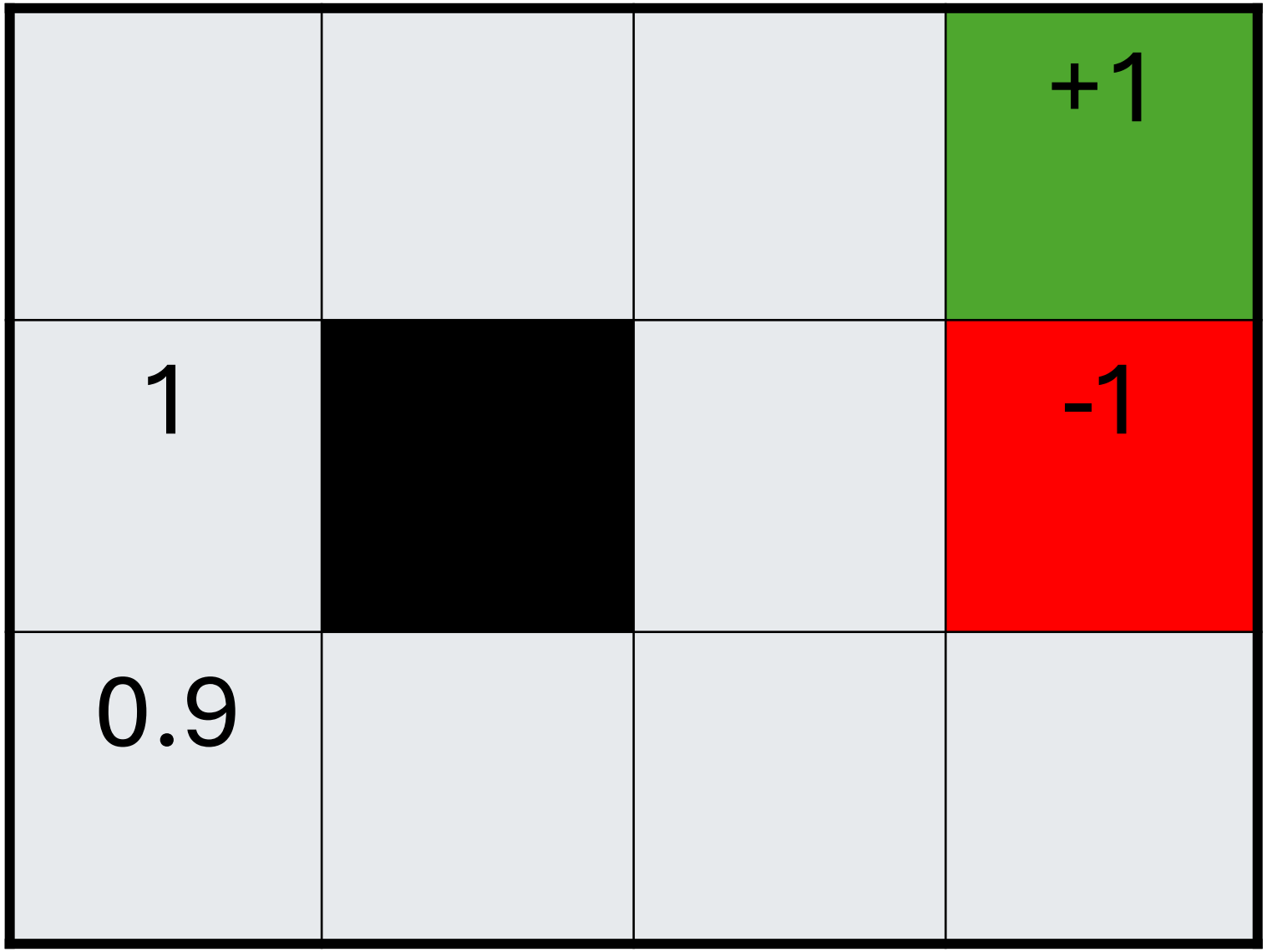
Eligibility trace



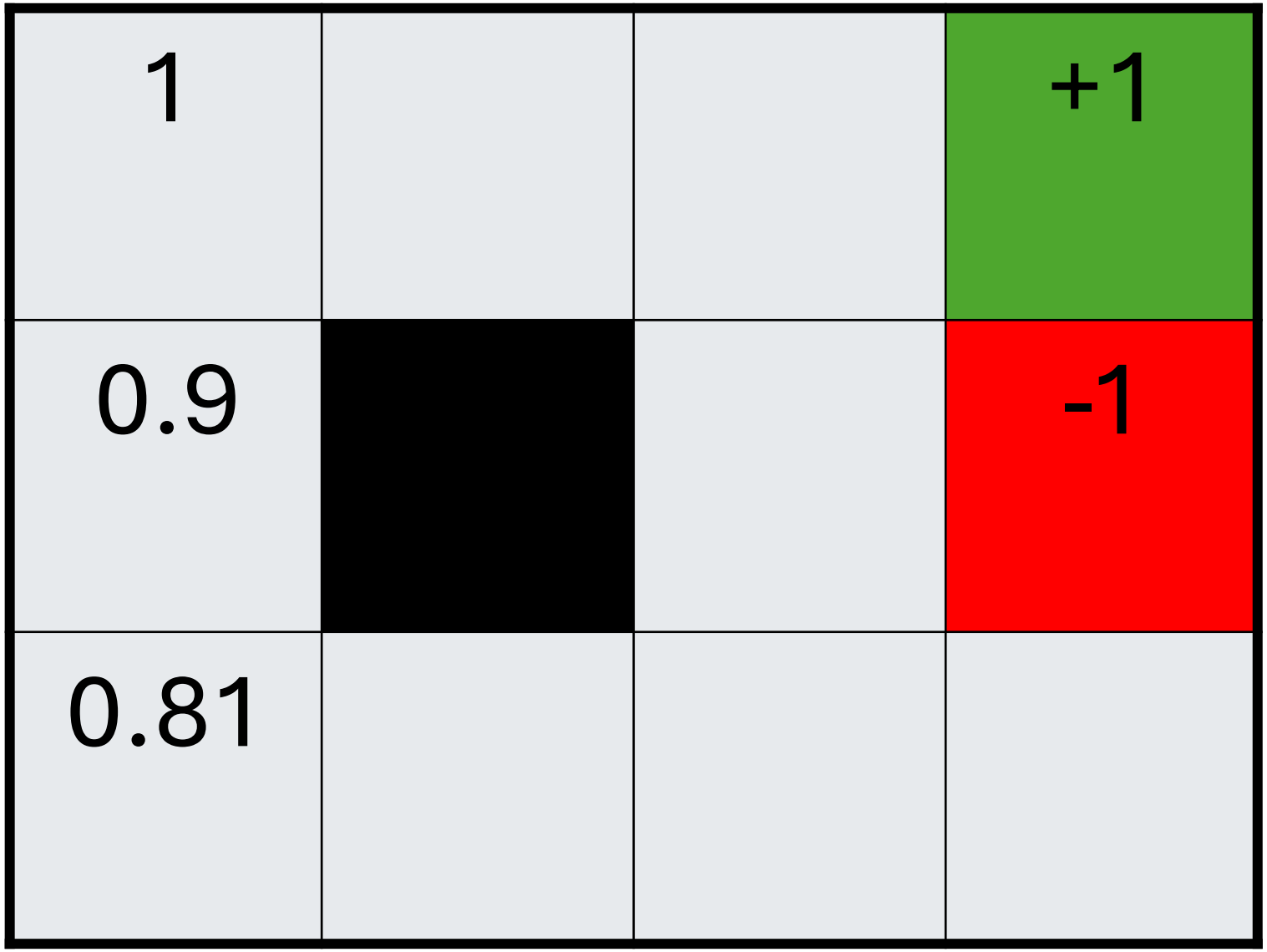
Eligibility trace



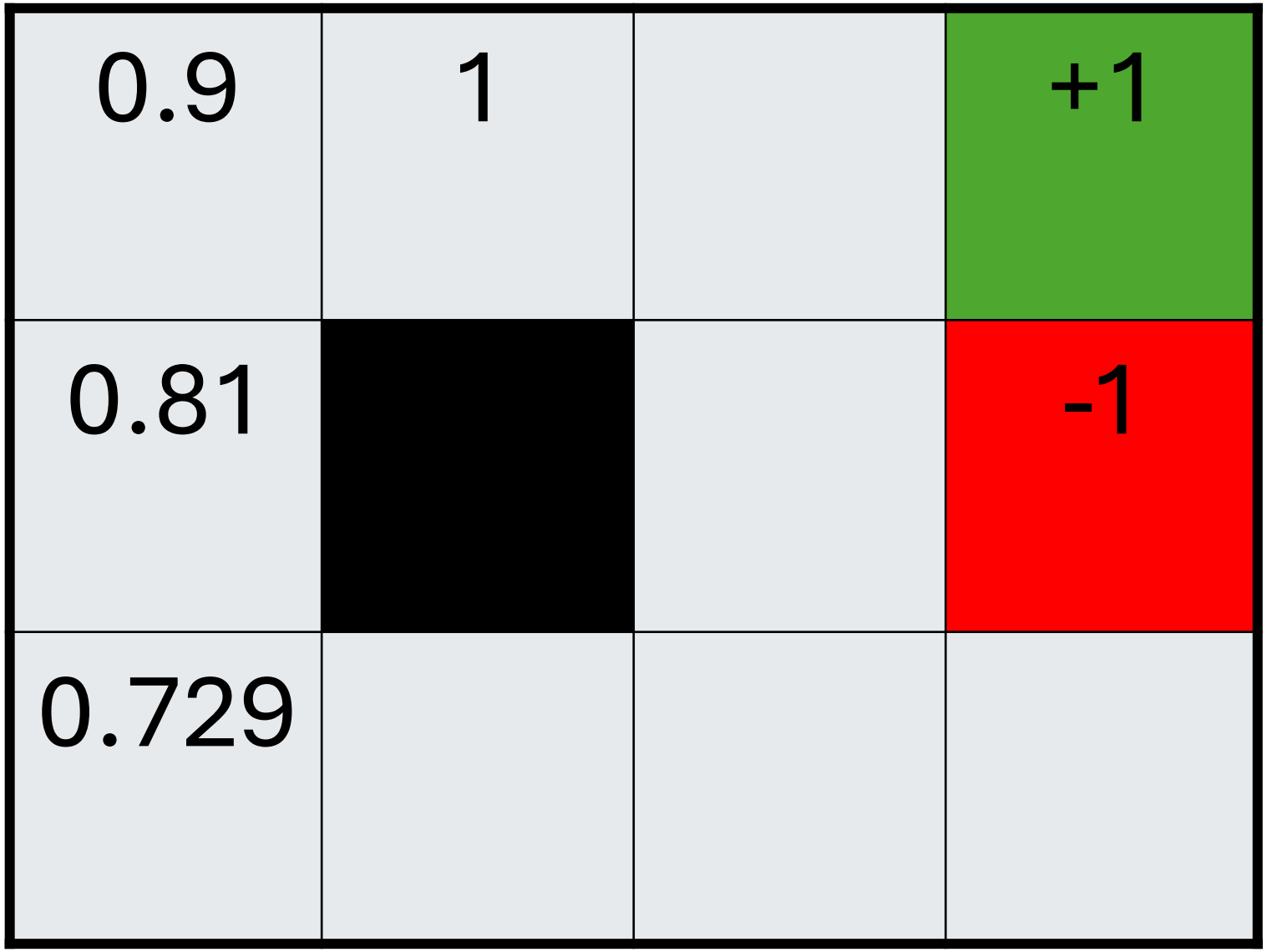
Eligibility trace



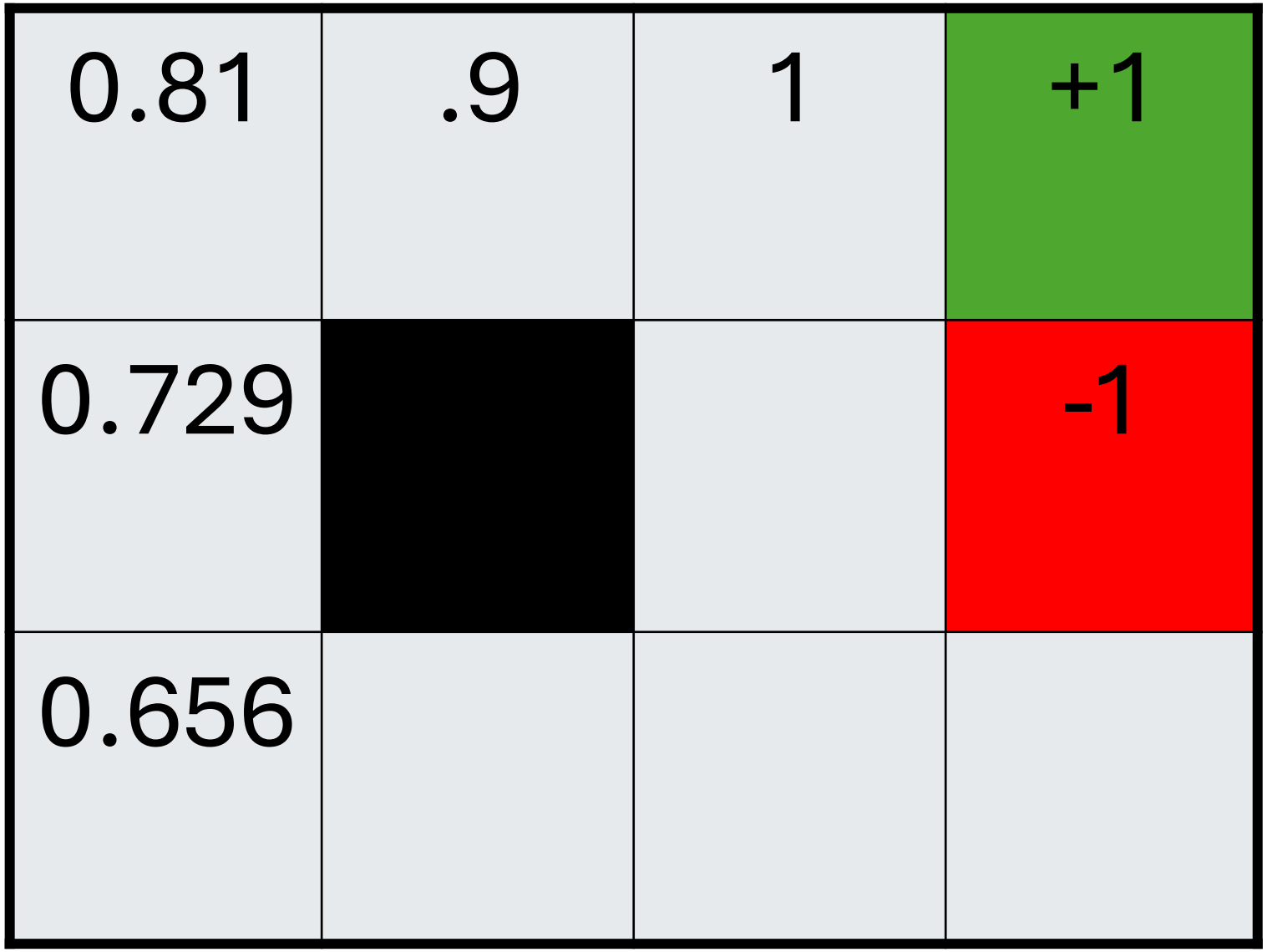
Eligibility trace



Eligibility trace



Eligibility trace



Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
    $\vec{e} = 0$
    $s \leftarrow$ initial state of episode
    Repeat (for each step of episode):
        $a \leftarrow$ action given by π for s
        Take action a, observe reward, r, and next state, s'
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $\vec{e} \leftarrow \gamma\lambda\vec{e} + \vec{\phi}_s$
        $\vec{\theta} \leftarrow \vec{\theta} + \alpha\,\delta\,\vec{e}$
        $s \leftarrow s'$
    until s is terminal

- δ is or normal TD error
- **e** is vector of eligibility traces
- **θ** is a weight vector

Linear Methods

- Why are these a particularly important type of function approximation?

$$V_t(s) = \sum_{i=1}^{n} \theta_t(i)\phi_s(i)$$

- Parameter vector θ_t
- Column vector of features ϕ_s for every state
- (same number of components)

(Code excerpts from Peter Stone: Keepaway)

```
void CMAC::loadTiles()
{
  int tilingsPerGroup = TILINGS_PER_GROUP;  /* num tilings per tiling group */
  numTilings = 0;

  /* These are the 'tiling groups'  --  play here with representations */
  /* One tiling for each state variable */
  for ( int v = 0; v < getNumFeatures(); v++ ) {
    for ( int a = 0; a < getNumActions(); a++ ) {
      GetTiles1( &(tiles[ a ][ numTilings ]), tilingsPerGroup, colTab,
                 state[ v ] / getResolution( v ), a , v );
    }
    numTilings += tilingsPerGroup;
  }
  if ( numTilings > RL_MAX_NUM_TILINGS )
    cerr << "TOO MANY TILINGS! " << numTilings << endl;
}
```

```
double CMAC::computeQ( int action )
{
  double q = 0;
  for ( int j = 0; j < numTilings; j++ ) {
    q += weights[ tiles[ action ][ j ] ];
  }
  return q;
}


int SarsaAgent::selectAction()
{
  int action;  // Epsilon-greedy

  if ( drand48() < epsilon) {
    action = range_rand( getNumActions() ); //explore
  }
  else{
    action = argmaxQ(); //exploit
  }
  return action;
}
```

```
int SarsaAgent::startEpisode( double state[] )
{
  FA->decayTraces( 0 );
  FA->setState( state );

  for ( int a = 0; a < getNumActions(); a++ ) {
    Q[ a ] = FA->computeQ( a );
  }

  lastAction = selectAction();

  FA->updateTraces( lastAction );
  return lastAction;
}
```

```
void SarsaAgent::endEpisode( double reward ){
  double delta = reward - Q[ lastAction ];
  FA->updateWeights( delta, alpha );
  lastAction = -1;
}
```

```
int SarsaAgent::step( double reward, double state[] )
{
  double delta = reward - Q[ lastAction ];
  FA->setState( state );

  for ( int a = 0; a < getNumActions(); a++ ) {
    Q[ a ] = FA->computeQ( a );
  }

  lastAction = selectAction();
  delta += Q[ lastAction ];
  FA->updateWeights( delta, alpha );

  // need to redo because weights changed
  Q[ lastAction ] = FA->computeQ( lastAction ));
  FA->decayTraces( gamma * lambda );

  for ( int a = 0; a < getNumActions(); a++ ) {  //clear other than F[a]
    if ( a != lastAction ) {
      FA->clearTraces( a );
    }
  }
  FA->updateTraces( lastAction );  //replace/set traces F[a]

  return lastAction;
}
```

Alternative: Policy Gradient Methods

- Great for continuous actions
- Directly learn to update policy
- Current methods include actor-critic methods (e.g., PPO)

Edward Snowden: Leaks that exposed US spy programme

Outline

- Multi-arm Bandits
- RL Successes & Settings
- Tabular example
- Q-Learning and SARSA
- Models
- Function Approximation
- DQN

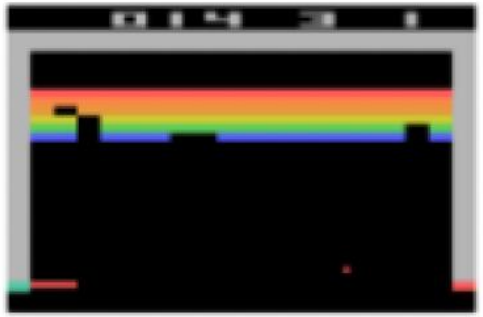
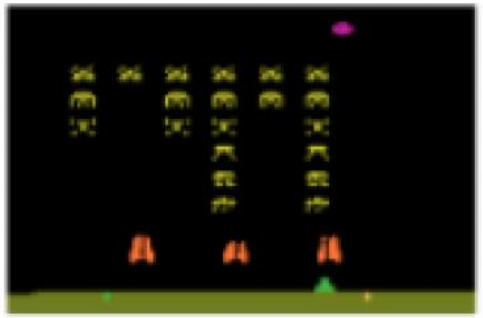
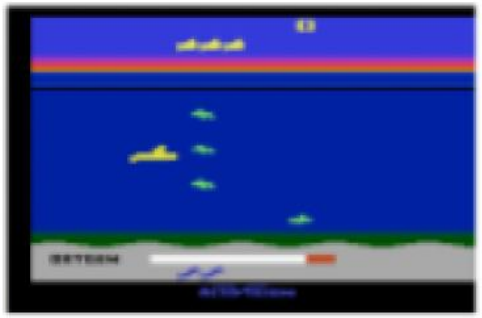
- Selection of research

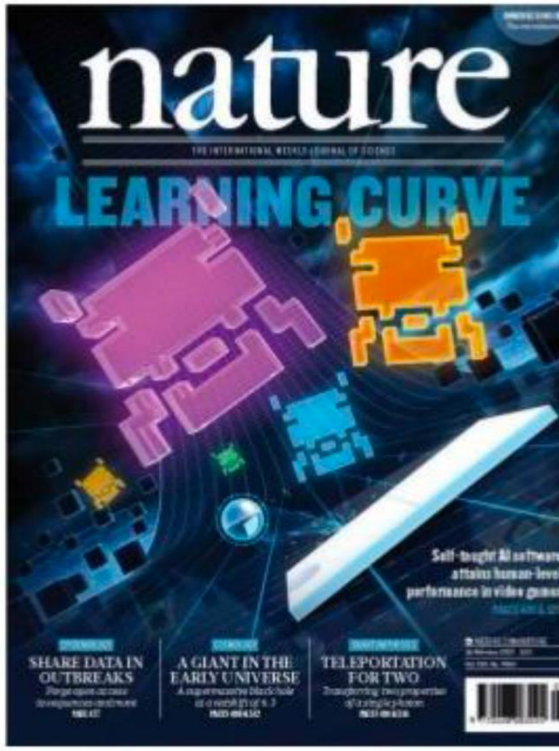
- Better/worse RL problems

DQN

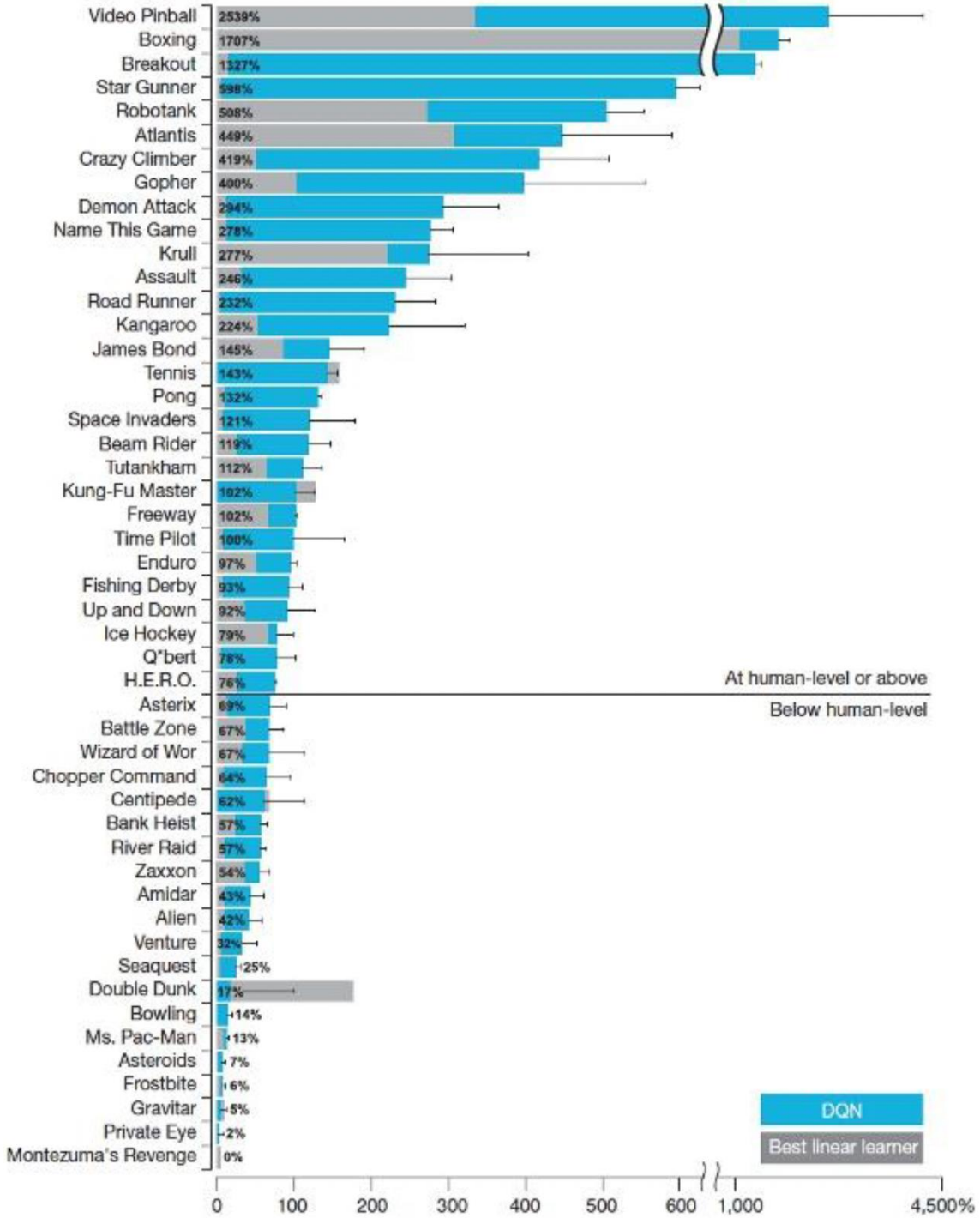
nature
LEARNING CURVE
Self-taught AI software attains human-level performance in video games
SHARE DATA IN OUTBREAKS
A GIANT IN THE EARLY UNIVERSE
TELEPORTATION FOR TWO

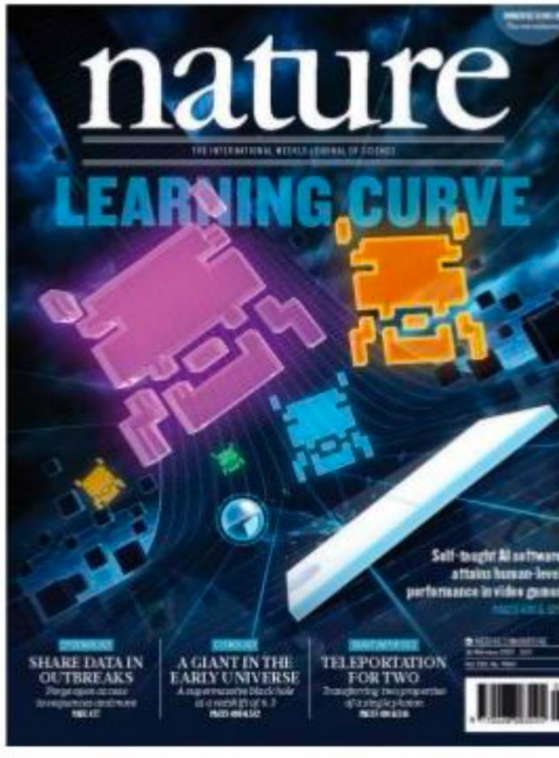
DQN


nature
LEARNING CURVE
SHARE DATA IN OUTBREAKS
A GIANT IN THE EARLY UNIVERSE
TELEPORTATION FOR TWO

DQN


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%


nature
LEARNING CURVE
SHARE DATA IN OUTBREAKS
A GIANT IN THE EARLY UNIVERSE
TELEPORTATION FOR TWO

DQN

(Deep) neural network

DQN

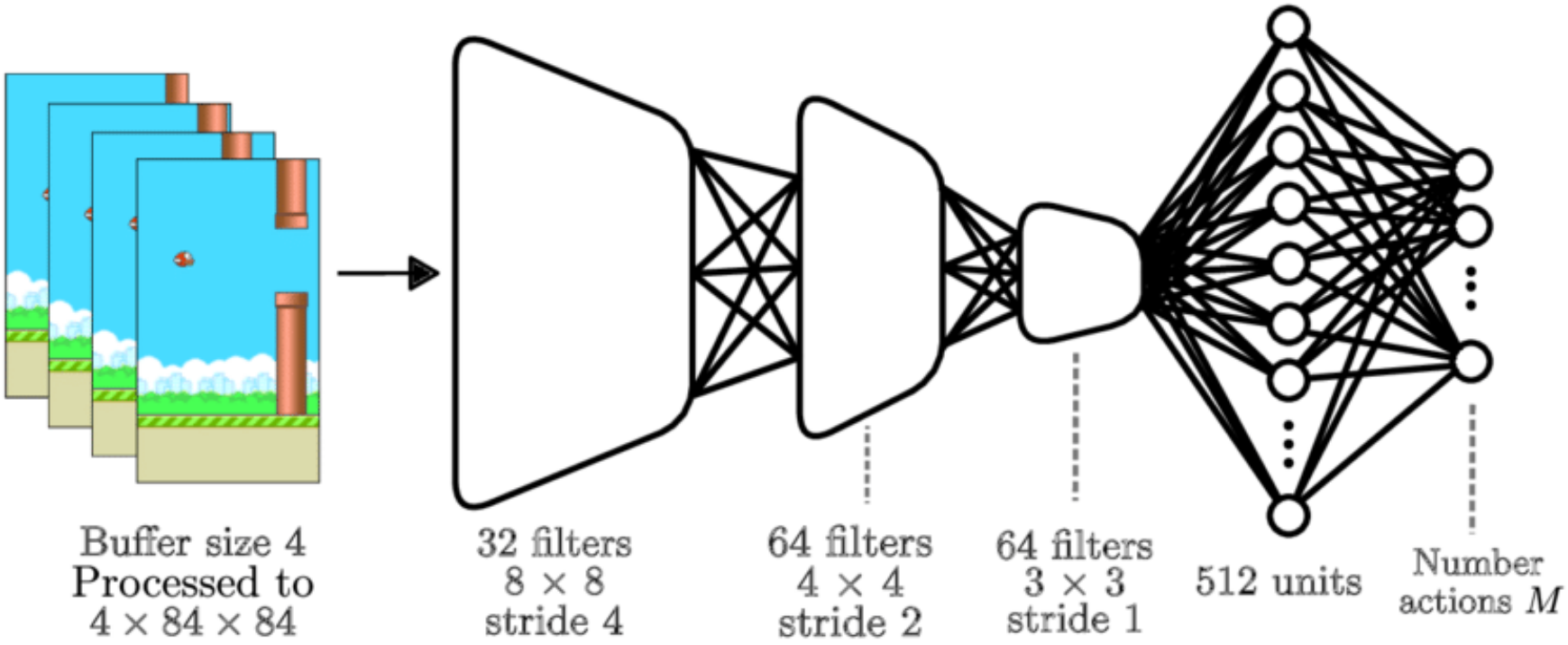
(Deep) neural network

1. CNN with Frame Stacking
2. Target network
3. Experience replay buffer
4. Reward clipping

DQN

1. CNN with Frame Stacking
2. Target network
3. Experience replay buffer
4. Reward clipping

$Q(s,a) \rightarrow Q(s,a;\theta)$



DQN

1. CNN with Frame Stacking
2. Target network
3. Experience replay buffer
4. Reward clipping

- Take a snapshot of the Q-network
- Use this network to calculate Q-updates

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right].$$

- Update a temporary network
- Every once in a while, overwrite the snapshot

DQN

1. CNN with Frame Stacking
2. Target network
3. Experience replay buffer
4. Reward clipping

- Store experience sequences
  - (s_t, a_t, r_t, s_{t+1})
- Randomly draw samples of experience & apply update in minibatch
- Have some method for clearing / cycling experience replay buffer
- Lots of hyperparameters around this….

- I.I.D. assumption!

DQN

1. CNN with Frame Stacking
2. Target network
3. Experience replay buffer
4. Reward clipping

   {-1,1}

DQN

1. CNN with Frame Stacking
2. Target network
3. Experience replay buffer
4. Reward clipping

Outline

- Multi-arm Bandits
- RL Successes & Settings
- Tabular example
- Q-Learning and SARSA
- Models
- Function Approximation
- DQN

- Selection of research

- Better/worse RL problems

RL Research Settings

Given an MDP

- Learn as quickly as possible
- Get the highest performance

Atari

Robot Soccer, Go

Given a problem

- Design and MDP and solve it
- Learn under some new assumptions

TD Gammon

Gran Turismo, Table Tennis

Agar.io, Learn from a human

Design & solve a new problem

Stock trading, Chicken health

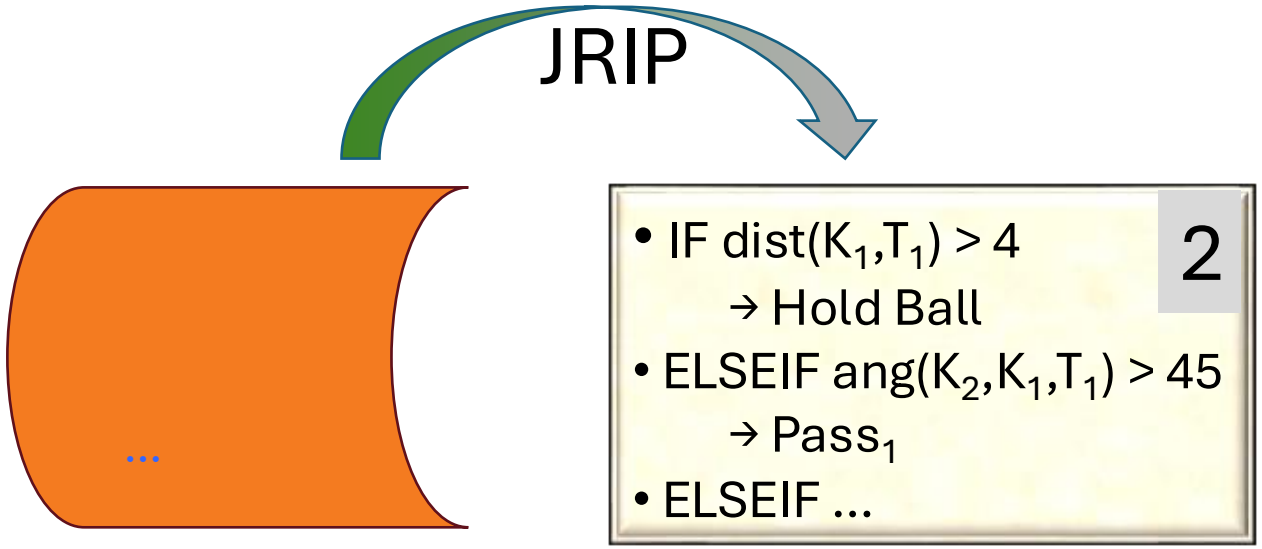
Improving RL via Human Demonstration

- RL can be slow: random bias
- Can be easy for humans to provide examples / teleoperate robot
- Want to
  1. Outperform human
  2. Learn faster than without using human
  3. Reduce total time (?)

Human-Agent Transfer (HAT)

1. Human Demonstration
2. Summarize Policy
3. Autonomous Learning

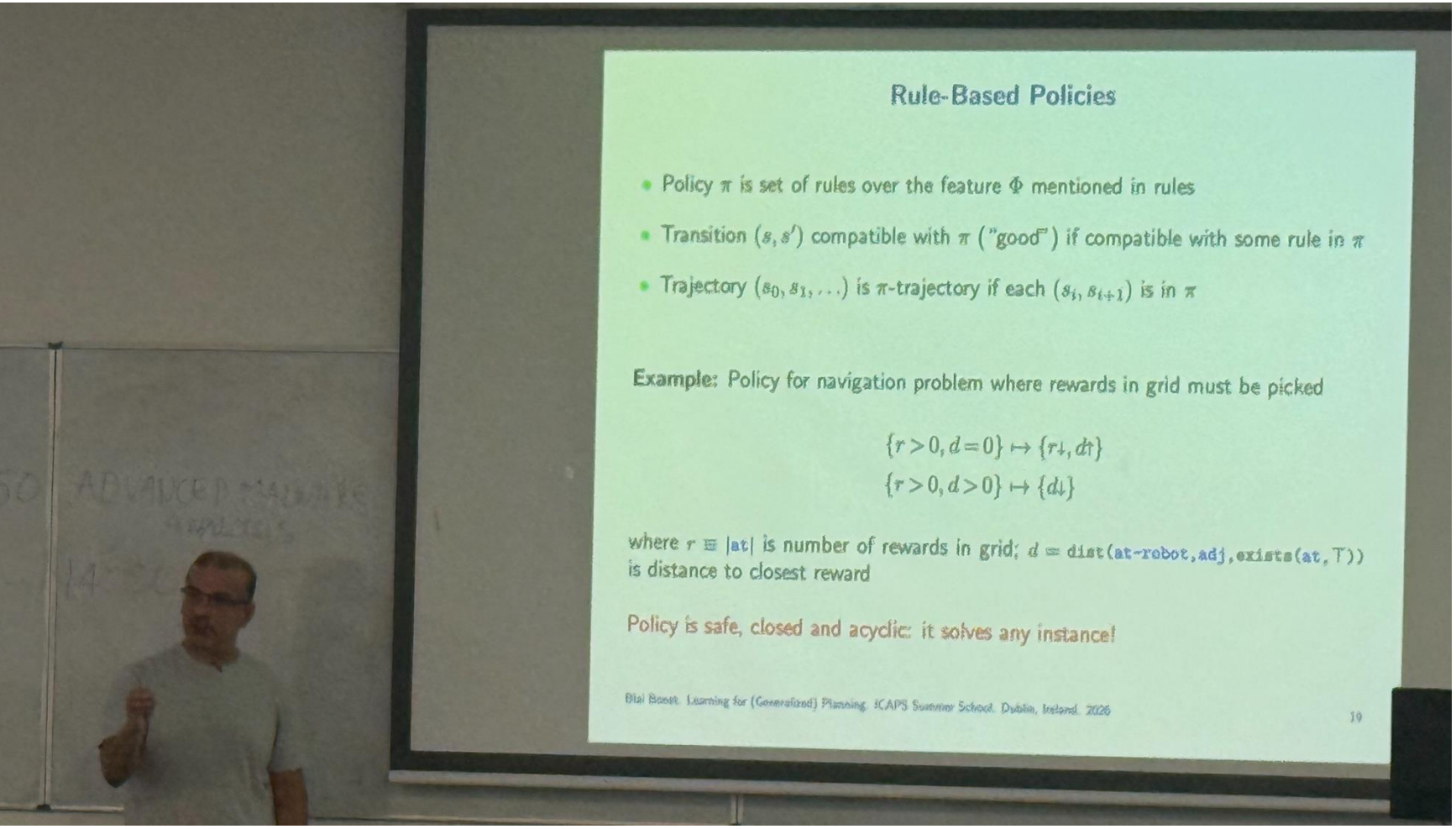
`Action 1, Action 2, …`



- P(ϕ) = Execute: Take suggested action
  - Decay ϕ over time
- P(ε) = Explore: Take random action
- P(1-ϕ-ε) = Exploit: Take action w/ max Q

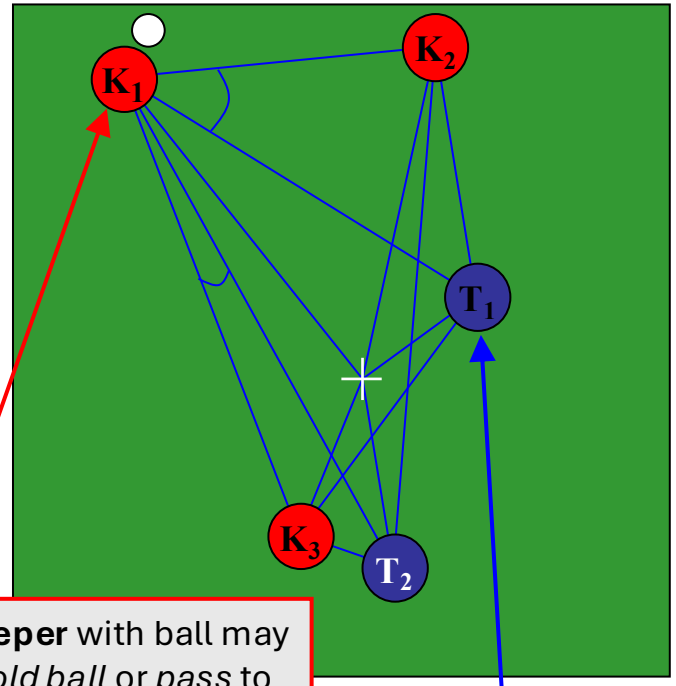
3

[MET, Suay, and Chernova, AAMAS-11]



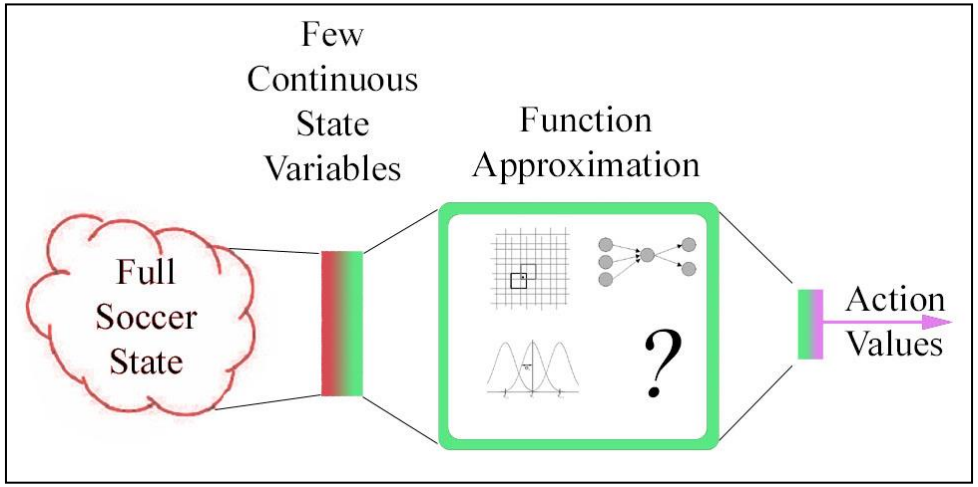
- IF dist(K_1,T_1) > 4
  → Hold Ball
- ELSEIF ang(K_2,K_1,T_1) > 45
  → $Pass_1$
- ELSEIF …

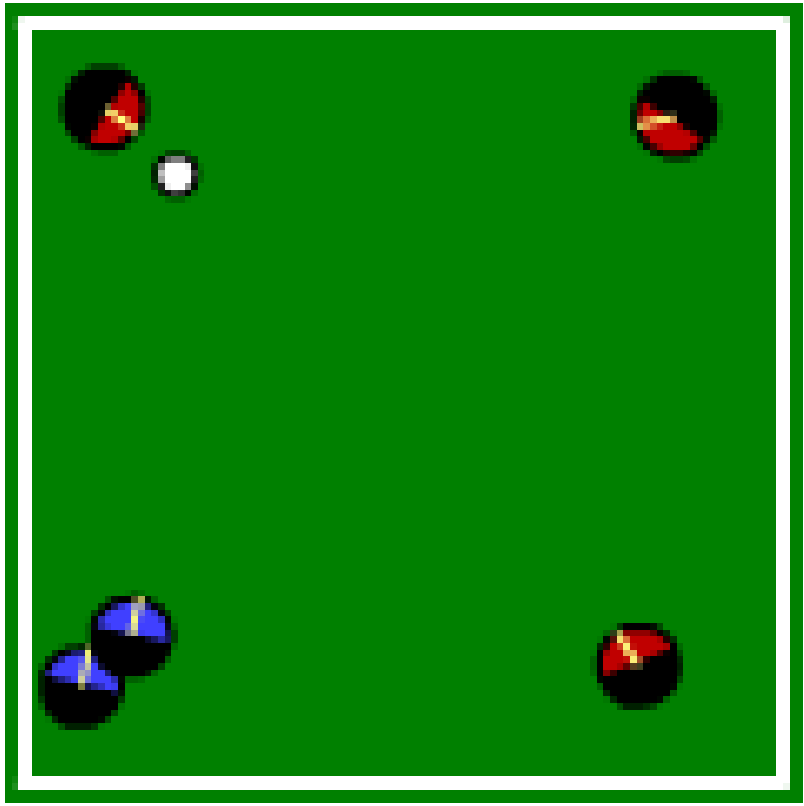
Human-Agent Transfer: Keepaway



Keeper with ball may *hold ball* or *pass* to either teammate

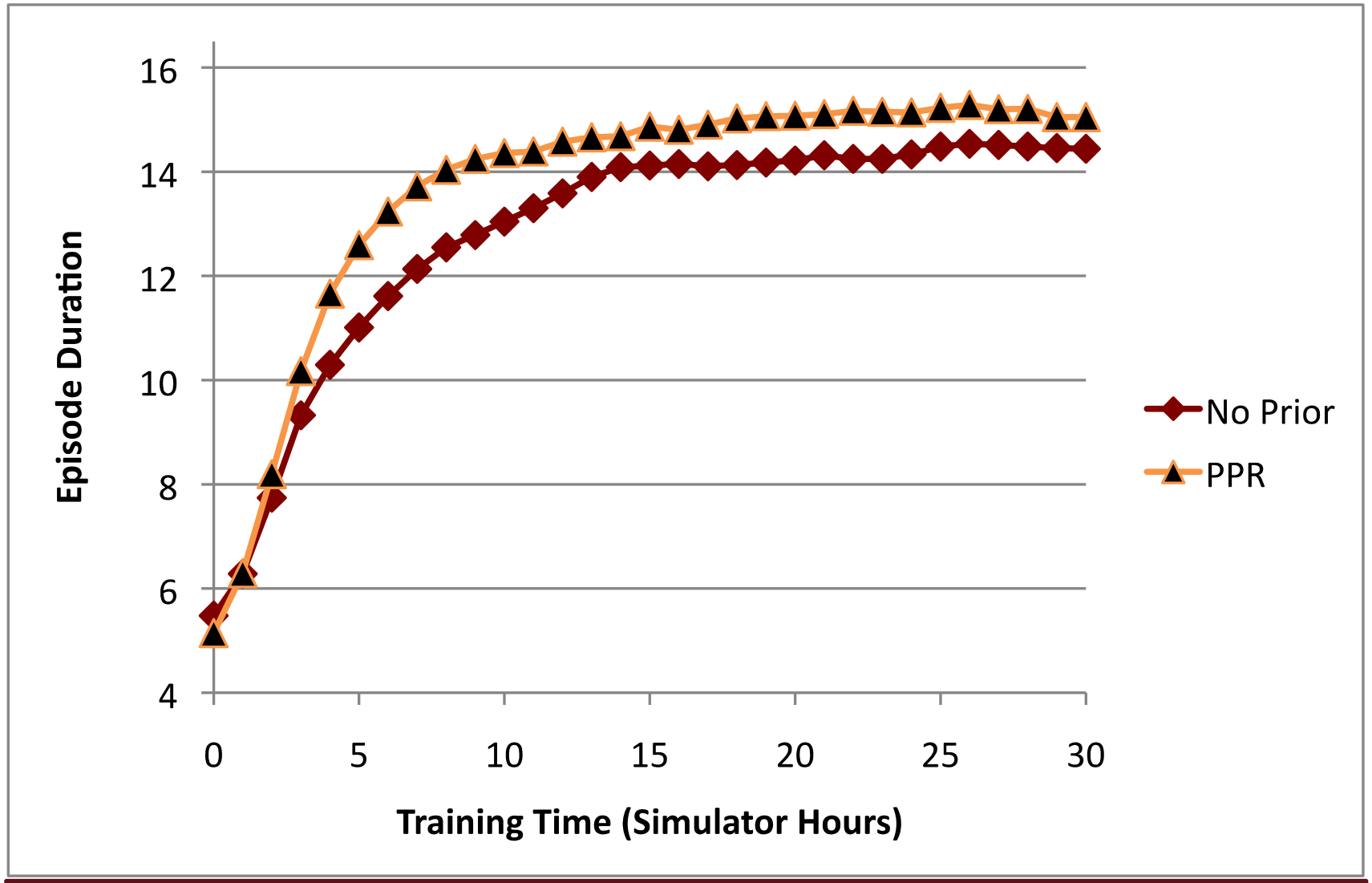
Both **takers** move towards player with ball





1. Human Demonstration: s, a, s', a', s'', …
2. Summarize Policy: $\pi_{human}(S) \rightarrow A$
3. Autonomous Learning: $\pi_{agent}(S) \rightarrow A$

HAT Results



- Jumpstart: Similar
- Time to Threshold: Improved
- Final Performance: Improved
- Total Reward: Improved

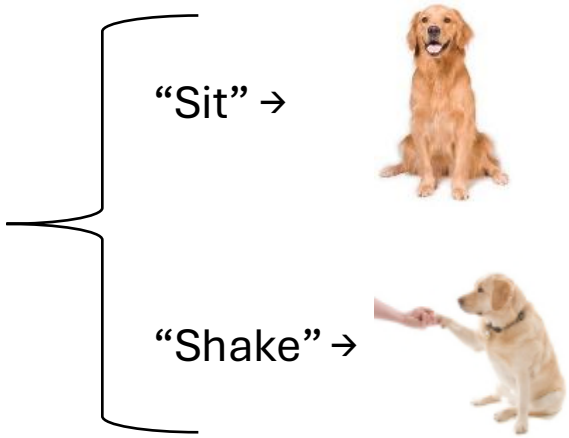
Improvements with only ~3 minutes of human time

Learning from Human Reinforcement

- What if no reward is defined?
- Want non-programmers to be able to teach robots & define tasks

- Teach dog to sit & shake

Policy



- Mapping from observations to actions
- Feedback: {Bad Dog, Good Boy}

Interaction History

Feedback history *h*

- Observation: “sit”, Action: , Feedback: “Bad Dog”
- Observation: “sit”, Action: , Feedback: “Good Boy”
- …

Really make sense to assign numeric rewards to these?

Bayesian Framework

- Trainer desires policy λ^*
- h_t is the training history at time t
- Find MAP hypothesis of λ^*:

$$\underset{\lambda}{\operatorname{argmax}}\, p(\lambda^* = \lambda | h_t) = \underset{\lambda}{\operatorname{argmax}}\, p(h_t | \lambda^* = \lambda) p(\lambda^* = \lambda)$$

Model of training process

Prior distribution over policies

Assumed Trainer Behavior

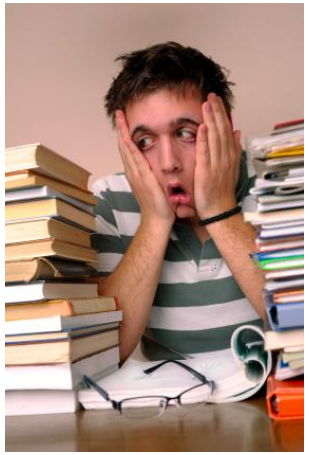
- Decide if action is correct
  - Does $\lambda^*(o)=a$? Trainer makes an error with $p(\varepsilon)$
- Decide if should give feedback
  - μ^+, μ^- are probabilities of neutral feedback
  - If thinks correct, give positive feedback with $p(1-\mu^+)$
  - If thinks incorrect, give negative feedback with $p(1-\mu^-)$
- Could depend on trainer

Inferring Neutral

- Try to learn what no-reward (μ^+ & μ^-) means
- Don't assume they're balanced

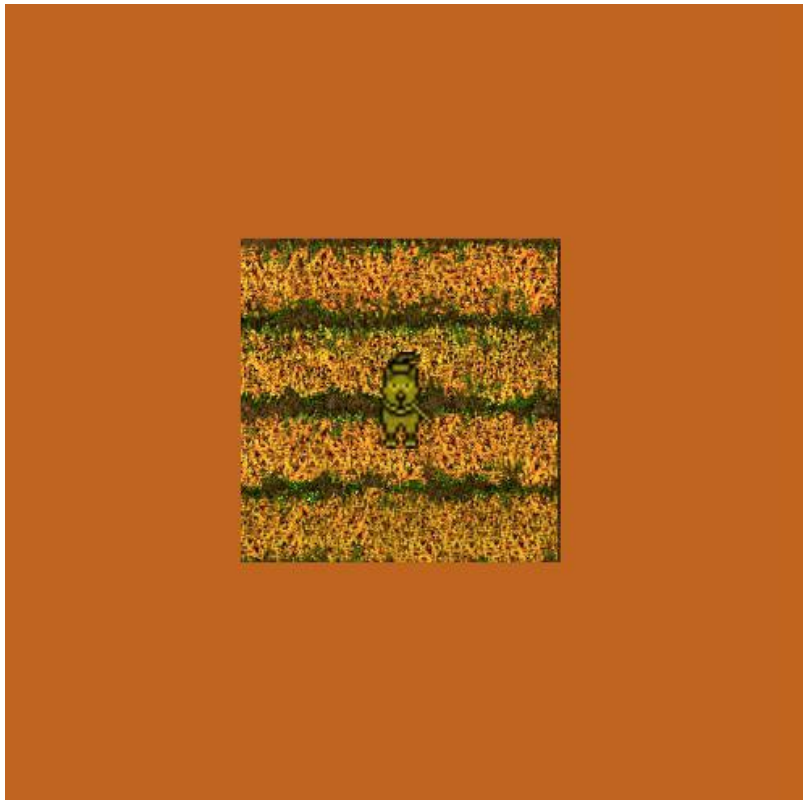
- Many trainers don't use punishment
  - Neutral feedback = punishment

- Some don't use reward
  - Neutral feedback = reward



Inferring Neutral: Human Studies

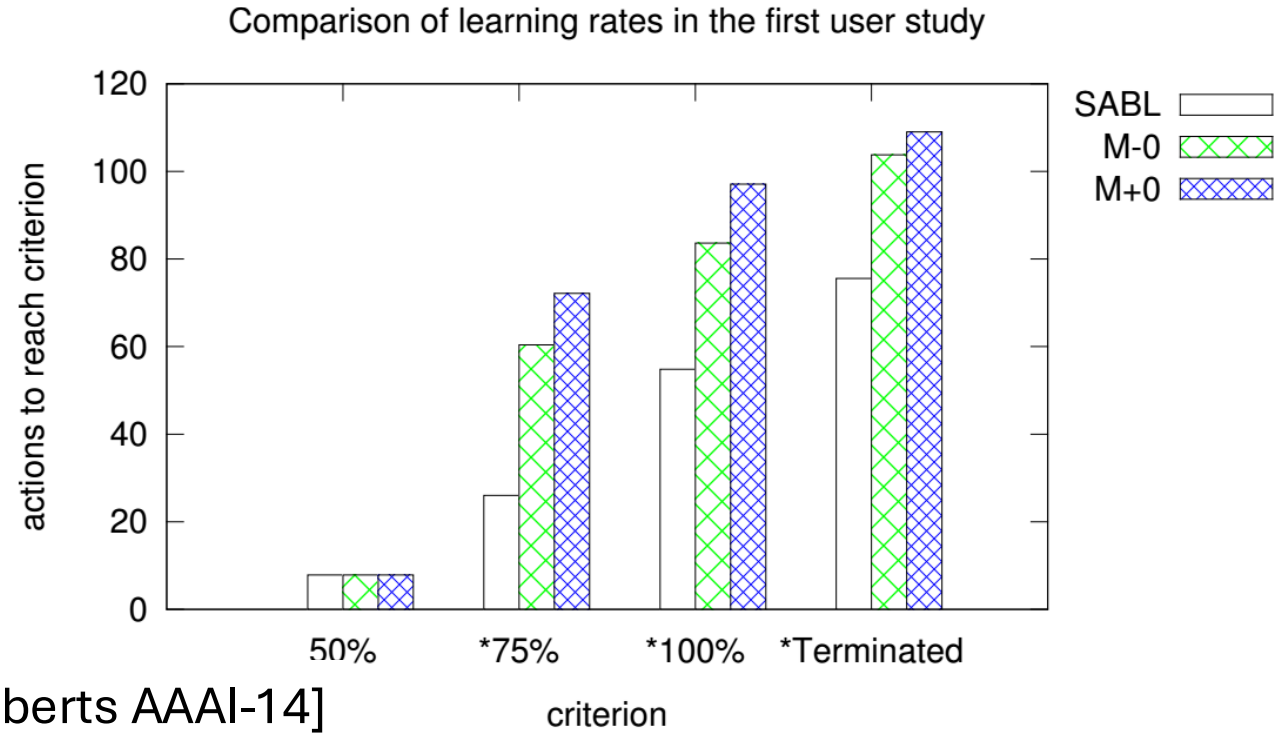
- Train simulated dog to protect the field
- Recruited Turkers and dog-training enthusiasts



	P+	P−
R+	93	125
R−	6	3

Explicitly reward good behavior? R+

Explicitly punish bad behavior? P+

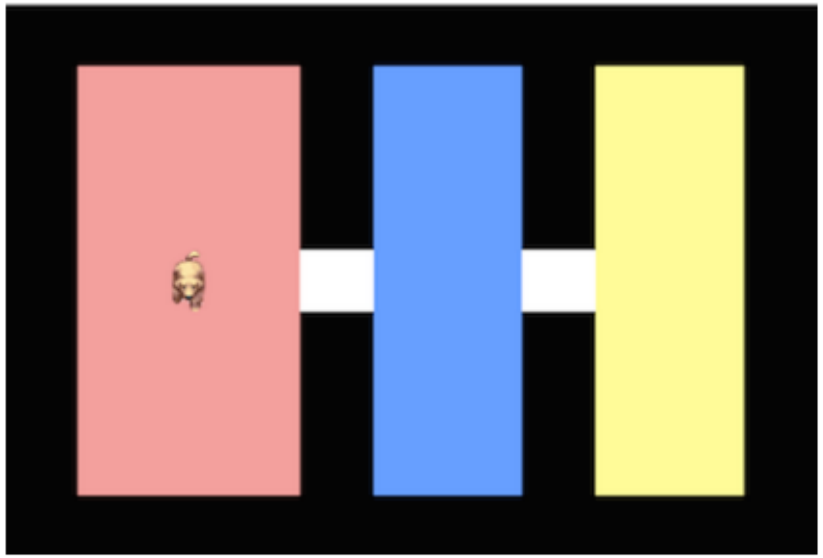


[Loftin, MacGlashan, Peng, MET, Littman, Huang & Roberts AAAI-14]

From Contextual Bandit to Sequential Task

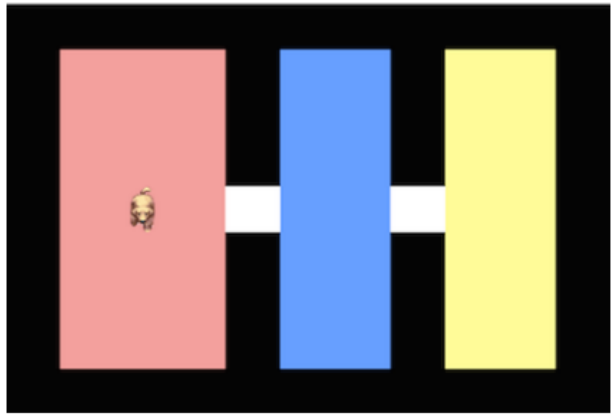
- Learn entire *policy*
- Given behavior consistent with some task
  - reward is evidence for the task
  - punishment is evidence against the task
- Given prior over possible tasks (reward functions)
  - Feedback used to update the posterior
- Knowing reward function = knowing behavior

Note: Assume known transition function & access to planner

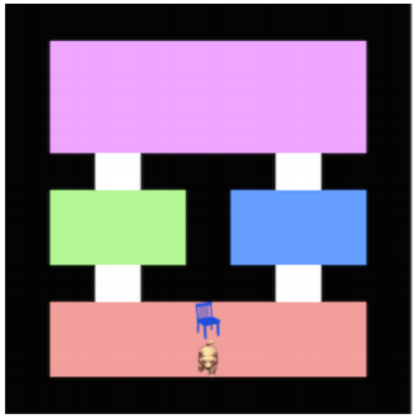


(a) "Move to the yellow room"

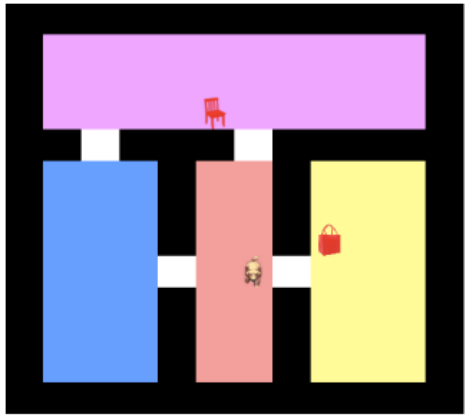
Users can Train Sequential Tasks



(a) "Move to the yellow room"



(b) "Move the blue chair to the purple room"



(c) "Move the red bag to the blue room"

#	Command Given	Accuracy
1	Move to the yellow room	98.3%
2	Move the blue chair to the purple room	81.7%
3	Move the red bag to the blue room	91.7%

Implicit Communication using Confidence

- The agent's entropy-based confidence level:

$$H = -\sum_{a \in A} Pr\left(a = a^* | s, F\right) ln\left(Pr\left(a = a^* | s, F\right)\right)$$

- The probability distribution over optimal action:

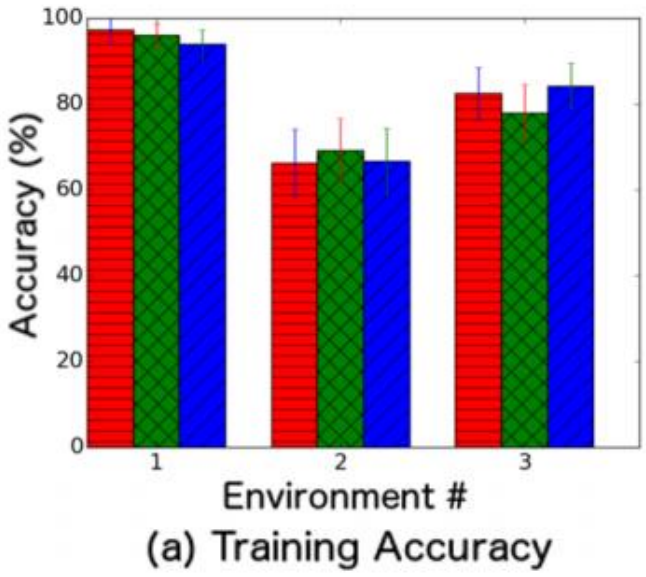
$$Pr\left(a = a^* | s, F\right) = \sum_{r \in R} \pi\left(s, a | r\right) Pr\left(r | F\right)$$

s : current state a : action taken F : feedback history

r : intended reward function

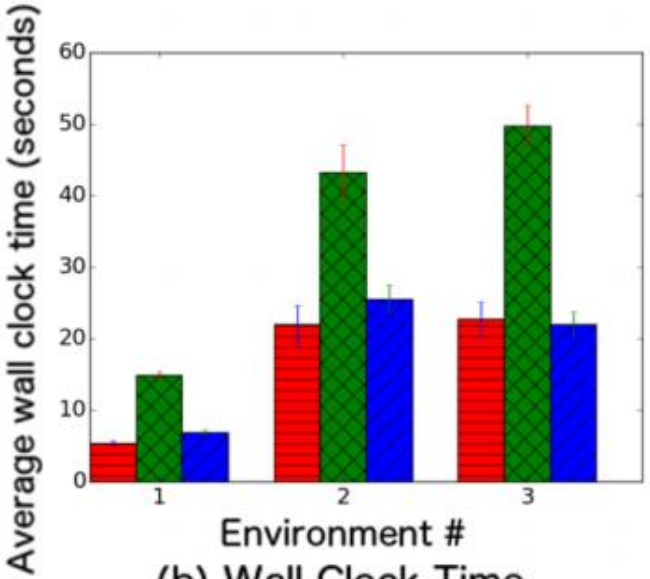
[Peng, MacGlashan, Loftin, Littman, Roberts & Taylor. AAMAS-16]

Adaptivity Helps Training



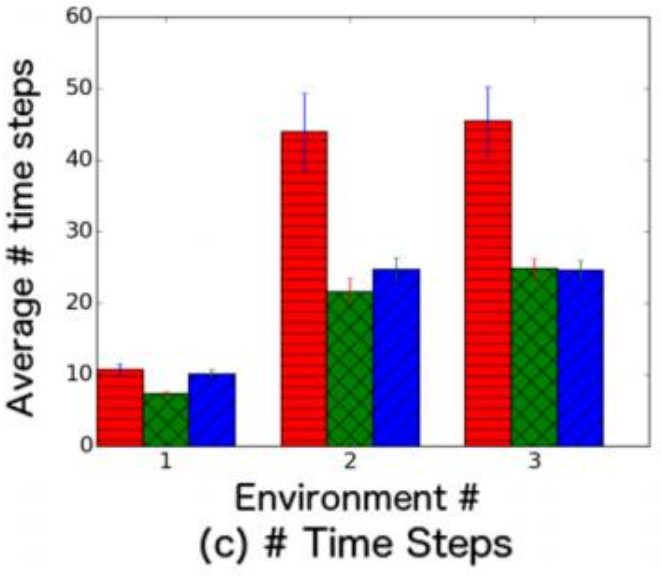
(a) Training Accuracy

Adaptive as accurate as “fast” and “slow”



(b) Wall Clock Time

Adaptive as quick as “fast”



(c) # Time Steps

Adaptive has as few training steps as “slow”

RL Research Settings

Given an MDP

- Learn as quickly as possible
- Get the highest performance

Atari

Robot Soccer, Go

Given a problem

- Design and MDP and solve it
- Learn under some new assumptions

TD Gammon
Gran Turismo, Table Tennis

Agar.io, Learn from a human

Design & solve a new problem

Stock trading, Chicken health

Towards Improving Reward Design in RL: A Reward Alignment Metric for RL Practitioners

Callie Muslimani, Kerrick Johnstonbaugh, Suyog Chandramouli, Serena Booth, W. Bradley Knox, MET

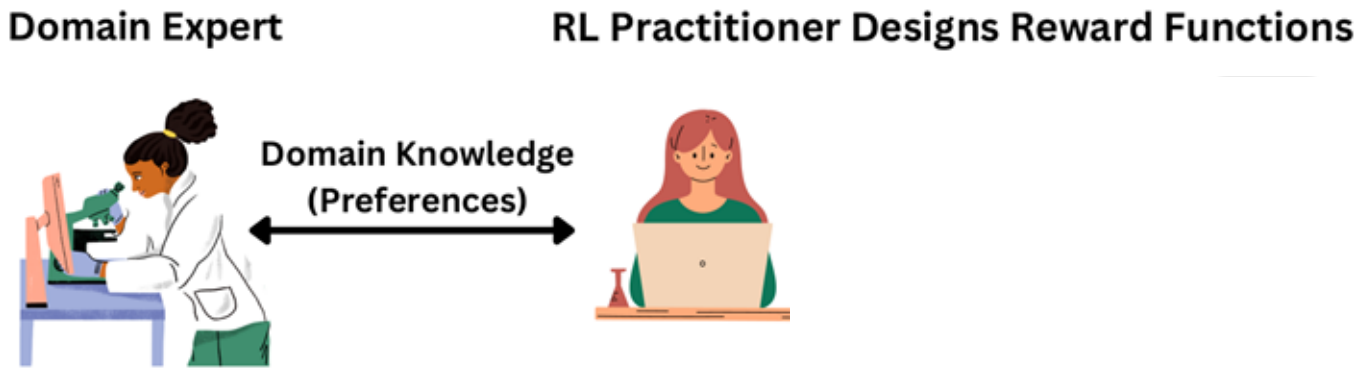
Reward functions are hard to design, Rich and Andy think so!

"The parts of our brains that generate these [reward] signals for us evolved over millions of years to be well suited to the challenges our ancestors had to face… We should therefore not think that designing a good reward signal is always an easy thing to do!"

– page 469 of RL textbook (2nd edition)

Why is reward design difficult?

1. Reward design is often a collaborative process

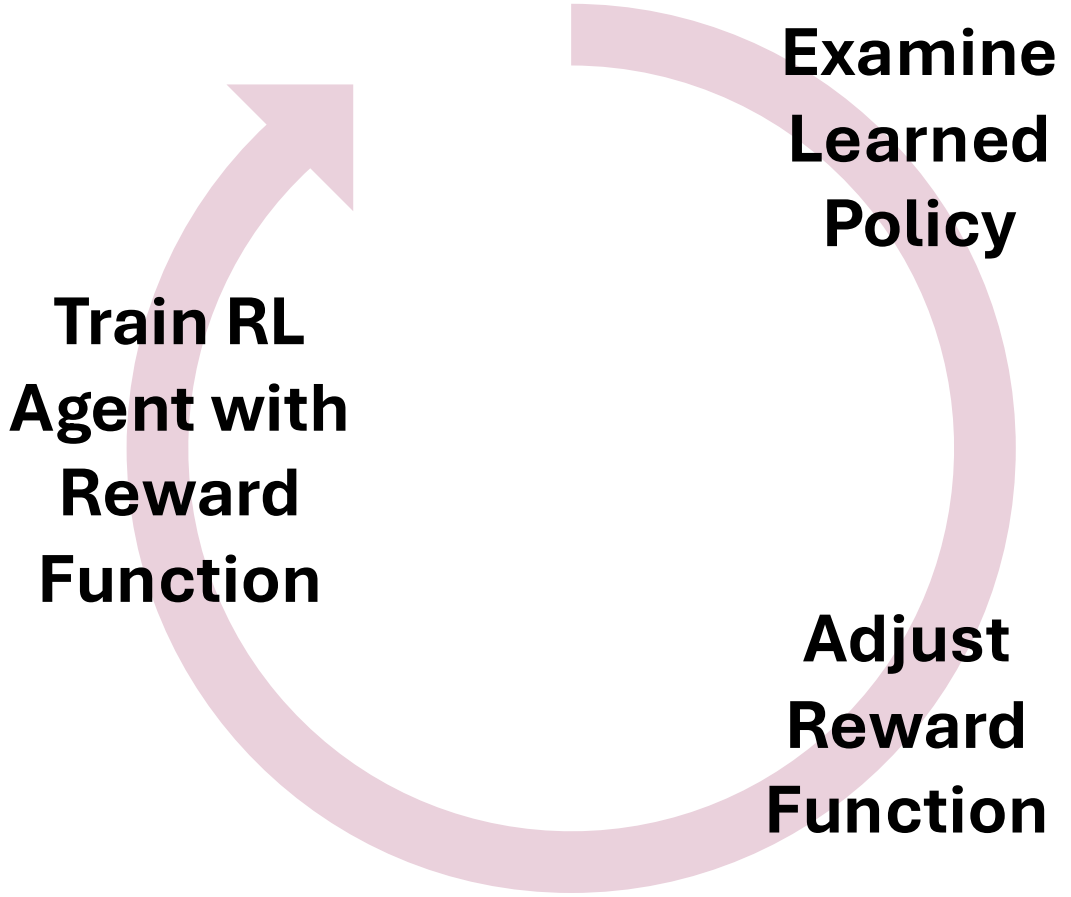


2. We have to consider return: *How do the individual rewards affect the total discounted reward?*

How do we evaluate if our reward functions are “properly specified”?

Common Evaluation Protocol:
Trial and Error Reward Design

1. Expensive
2. Can result in reward overfitting



Reward Alignment as an Evaluation Scheme: Does the reward encode the preferences of a domain expert?

→ Propose *Trajectory Alignment Coefficient*

evaluates the similarity between a domain expert's preferences over trajectory distributions to those induced by a given reward, discount factor pair

Trajectory Alignment Coefficient in Action

1. Get trajectory rankings

Domain Expert

Expected return:

(r, discount)

Best

$\tau_{success}$

10

τ_{idle}

0

$\tau_{success-crash}$

Prob. distribution:
90% success, 10% crash

4

Worst

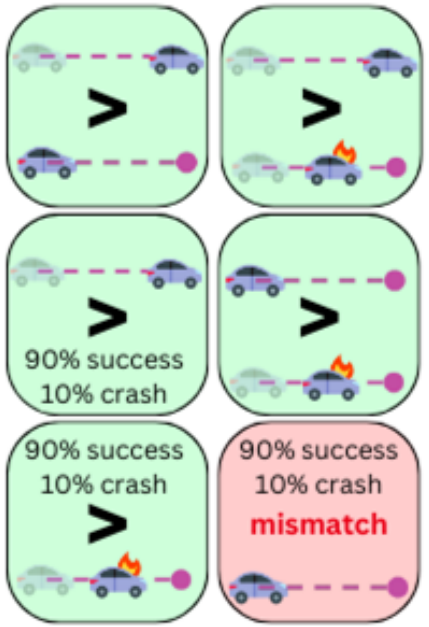
τ_{crash}

-50

Trajectory Alignment Coefficient in Action

2. Calculate score based on Kendall's Tau-b

Preference Matches (P) = 5
Preference Mis-matches (Q) = 1



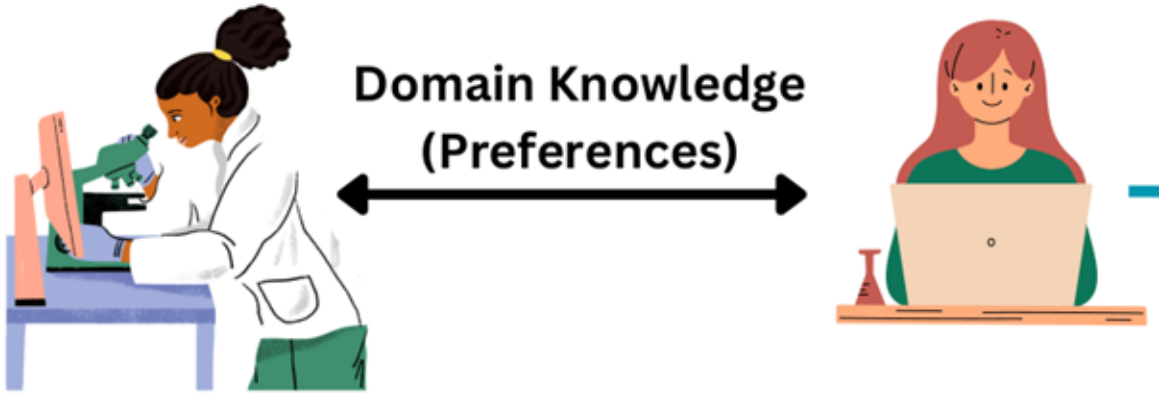
$$\frac{P - Q}{\sqrt{(P + Q + X_0)(P + Q + Y_0)}}$$

Where:
- P = Count of Concordant Pairs
- Q = Count of Discordant Pairs
- X_0 = Number of ties in human ranking
- Y_0 = Number of ties in (r, γ)-ranking

Trajectory Alignment Coefficient = 0.67

Can the Trajectory Alignment Coefficient help RL practitioners during reward design?

Study of RL 11 Practitioners in Reward Selection



Reward Function 1:

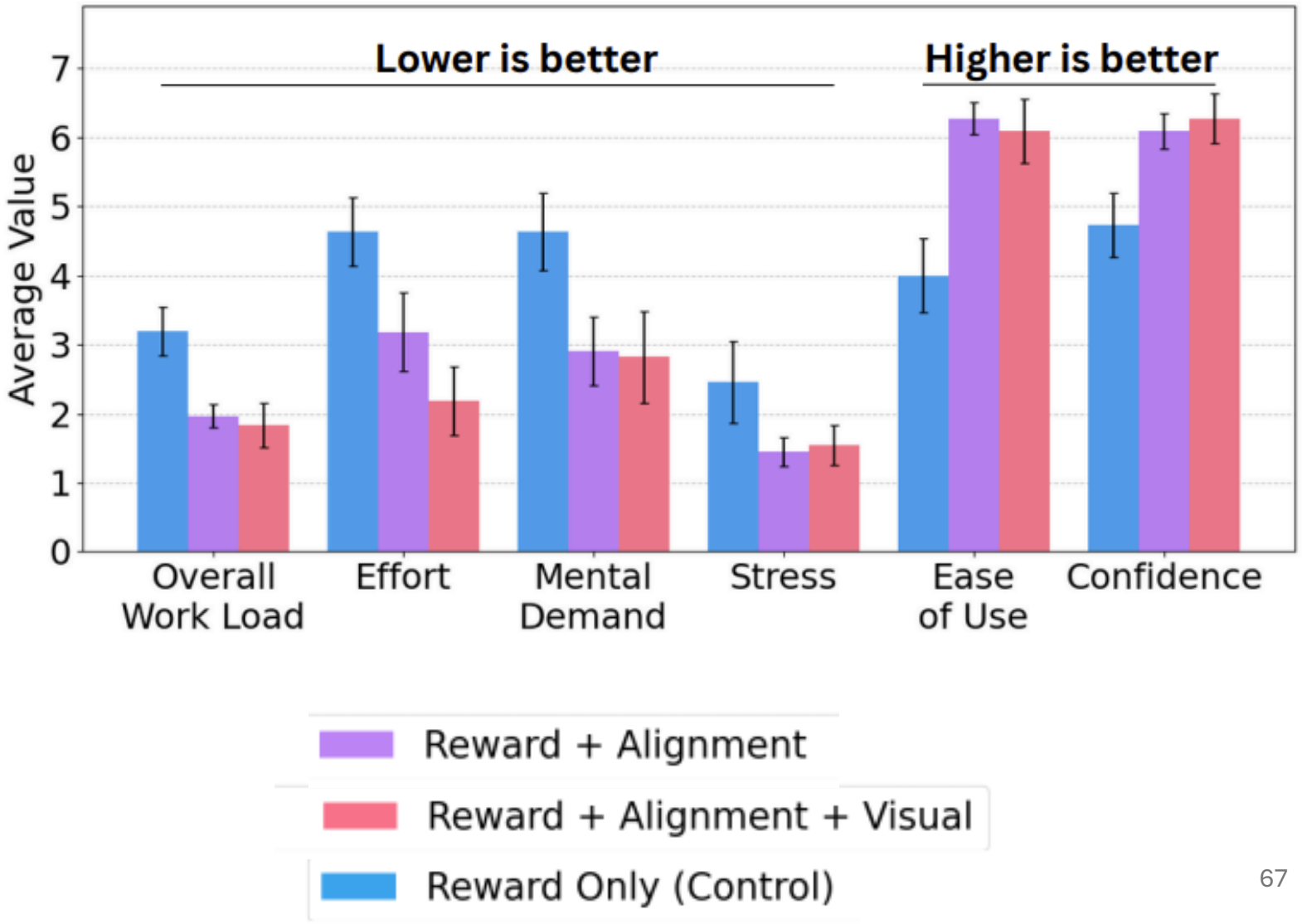
Hungry, Thirsty = 0
Hungry, Not Thirsty = 0
Not Hungry, Thirsty = 5
Not Hungry, Not Thirsty = 5

Reward Function 2:

Hungry, Thirsty = 0
Hungry, Not Thirsty = 0
Not Hungry, Thirsty = 0
Not Hungry, Not Thirsty = 1

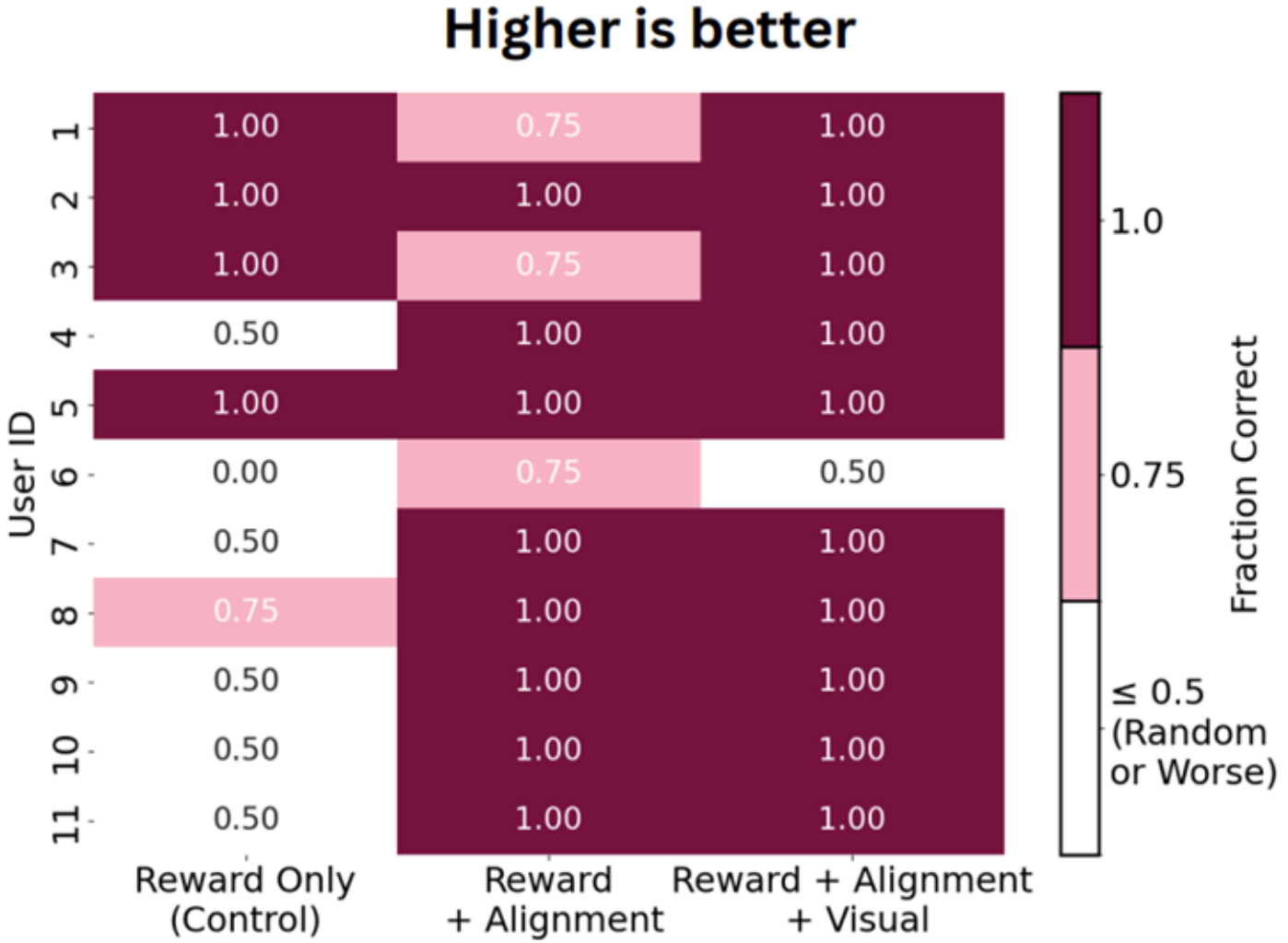
Does access to the trajectory alignment coefficient enhance perceived benefit?

Measured via
modified NASA TLX survey



Can access *to our metric helps users choose more performant reward functions?*

Measured by the proportion of times user selected reward that improved learning



Question # 1: Can access to the trajectory alignment coefficient increase perceived benefit (e.g, less cognitive workload, more confidence)?

Question # 2: Does it help users in choosing reward functions that improve performance of learned policies?

Key takeaways:

- Reward design is hard!
- Trajectory Alignment Coefficient is one means to evaluate reward functions.
- It can be used to improve reward design!

Questions?

Full Paper:

Personal Webpage:
https://cmuslima.github.io/

Ranked Return Regression for Reward Learning in Reinforcement Learning

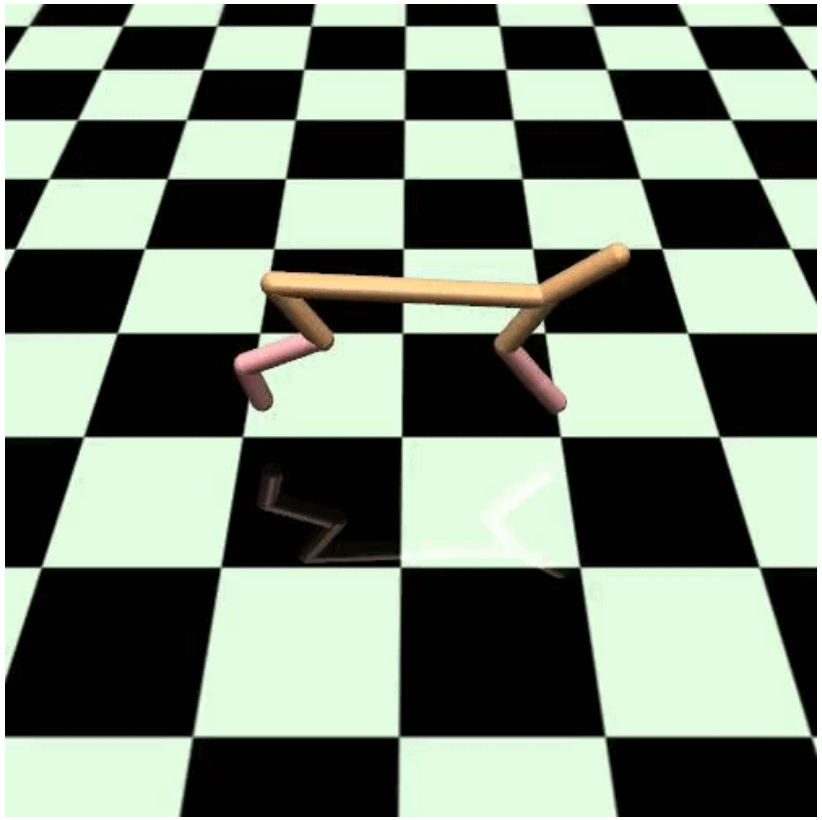
Chaitanya Kharyal, Calarina Muslimani, MET

Contributions

- Desiderata for reward learning methods from ratings
- A rating-based reward learning method
- Theoretical guarantees that the existing rating-based method cannot offer
- Studying the method empirically, both with simulated and real user data

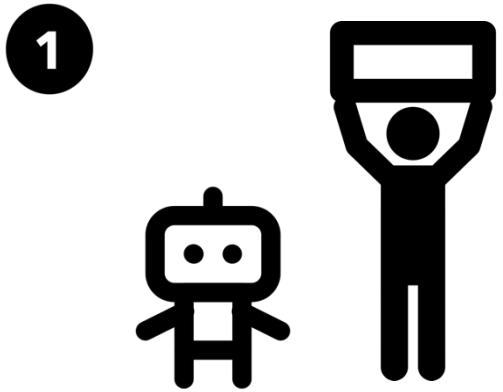
Reward Design is Hard!

- Designing sparse reward functions that achieve the end goal might be easy
  - A reward for winning the chess game
  - A reward for forward distance in half-cheetah
- But we often care just as much about the behavior
- Designing reward functions that follow a specified behavior is non-trivial



Alex Irpan, “Deep Reinforcement Learning Doesn't Work Yet,” 2018
Booth et al., “The Perils of Trial-and-Error Reward Design: Misdesign through Overfitting and Invalid Task Specifications,” AAAI’23

Learning Reward Functions from Feedback

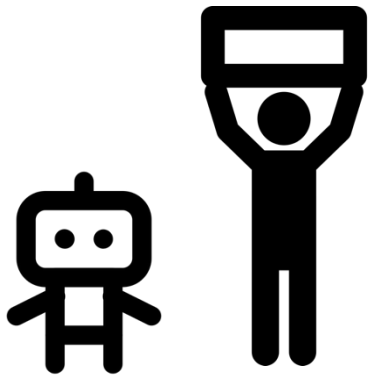


Demonstrations

- Inverse Reinforcement Learning
- Usually requires (near) optimal trajectories

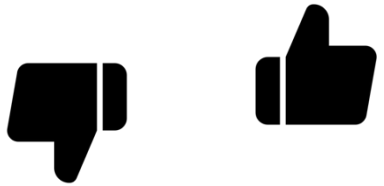
Learning Reward Functions from Feedback

1



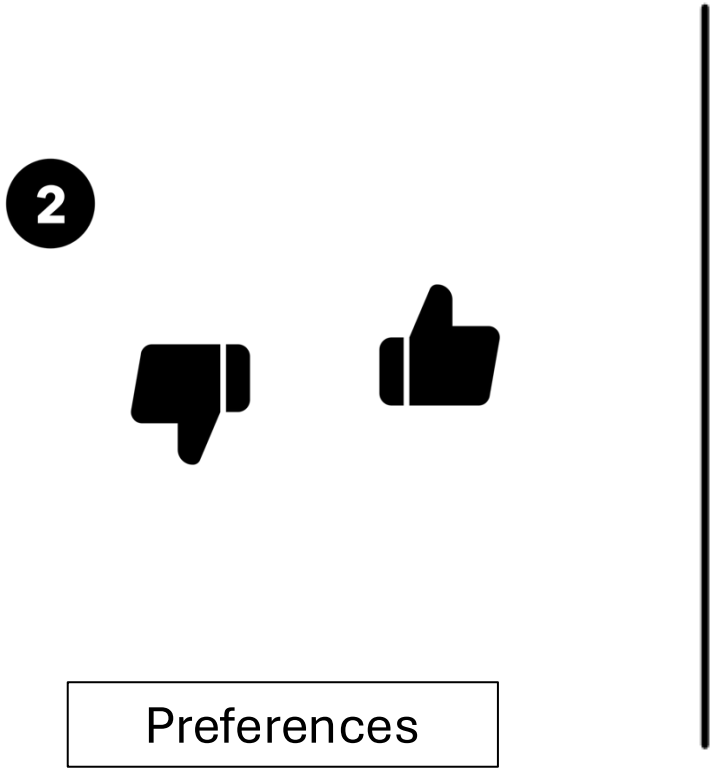
Demonstrations

2



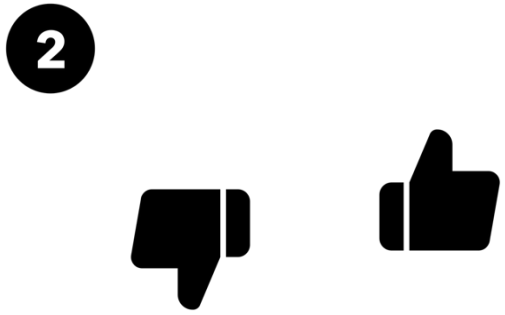
Preferences

Learning Reward Functions from Feedback



Bradley and Terry, "Rank Analysis of Incomplete Block Designs," Biometrika, 1952.
Lee et al., "PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training," ICML'21
Park et al., "SURF:Semi-supervised Reward Learning with Data Augmentation for Feedback-efficient Preference-based Reinforcement Learning," ICLR'22
Hu et al., "Query-policy mis-alignment in preference-based reinforcement learning," ICLR'24

Learning Reward Functions from Feedback



Preferences

- Commonly modeled using Bradley-Terry model:

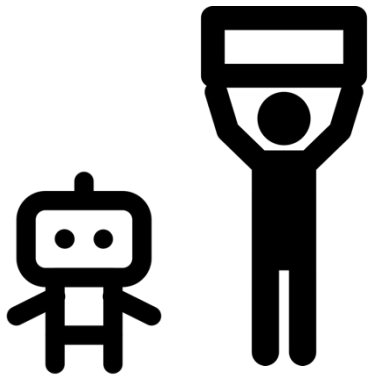
$$P(\tau_i \succ \tau_j) = \frac{e^{G_\theta(\tau_i)}}{e^{G_\theta(\tau_i)} + e^{G_\theta(\tau_j)}}$$

- Improvements include entropy training for better feedback (PEBBLE), data augmentation (SURF), and improving query selection (QPA)

Bradley and Terry, "Rank Analysis of Incomplete Block Designs," Biometrika, 1952.
Lee et al., "PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training," ICML'21
Park et al., "SURF:Semi-supervised Reward Learning with Data Augmentation for Feedback-efficient Preference-based Reinforcement Learning," ICLR'22
Hu et al., "Query-policy mis-alignment in preference-based reinforcement learning," ICLR'24

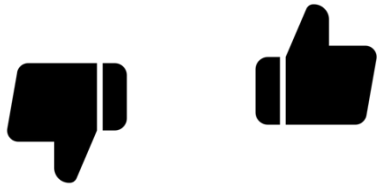
Learning Reward Functions from Feedback

1



Demonstrations

2



Preferences

3

Ratings

Learning Reward Functions from Feedback

Ratings

- Rating-based reinforcement learning (RbRL) baseline treats ratings as categorical
- Makes unjustified assumptions about the trajectories with the same rating having identical quality

White et al., “Rating-based Reinforcement Learning,” AAAI’24

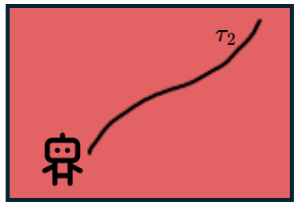
Problem Setting

Dataset $\mathcal{D} = \{\tau_i, c(\tau_i)\}_{i=0}^{n-1}$

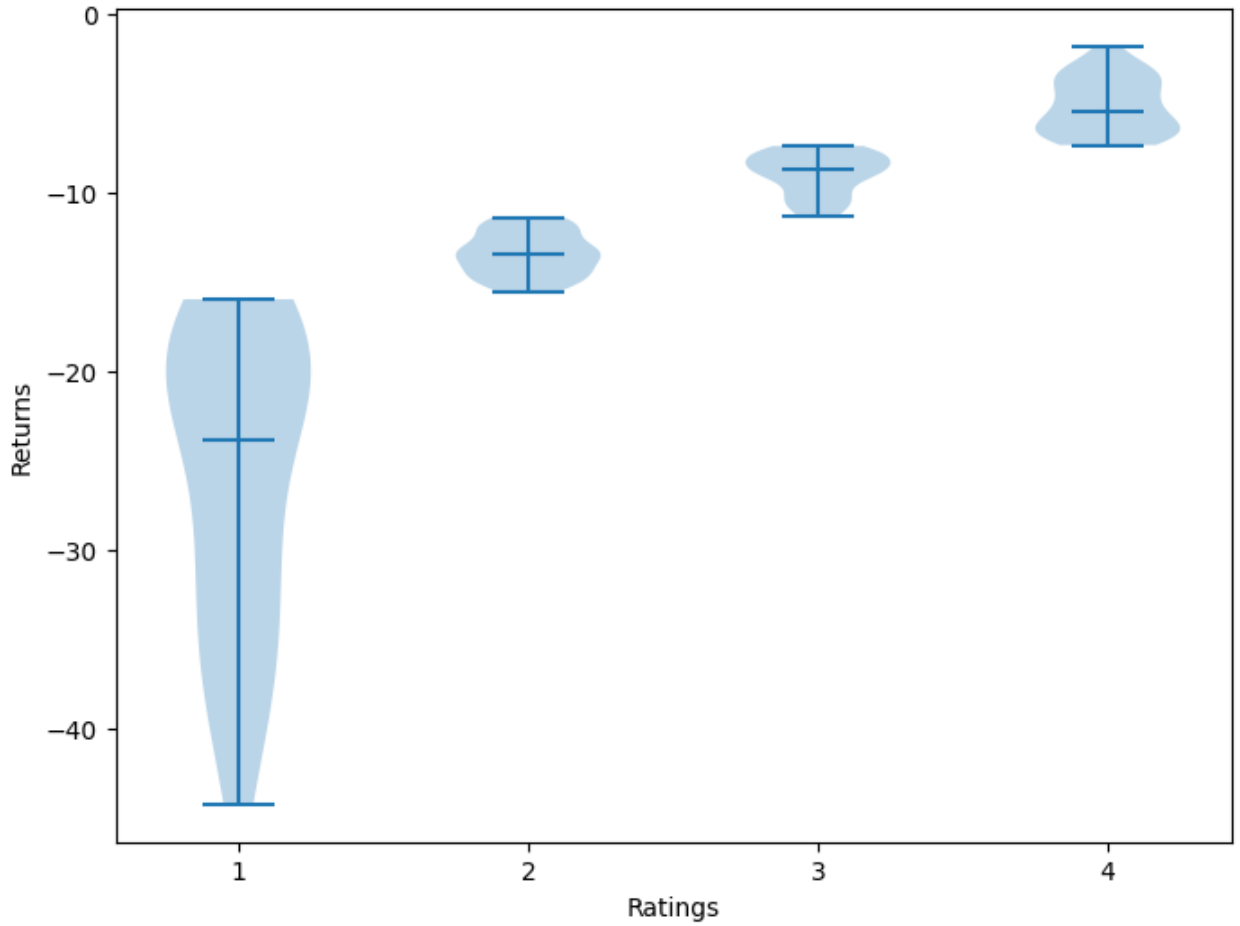
Learn a reward function aligned with the user provided ratings

Learn a policy using that reward function. Policy ideally should perform well on the task we care about

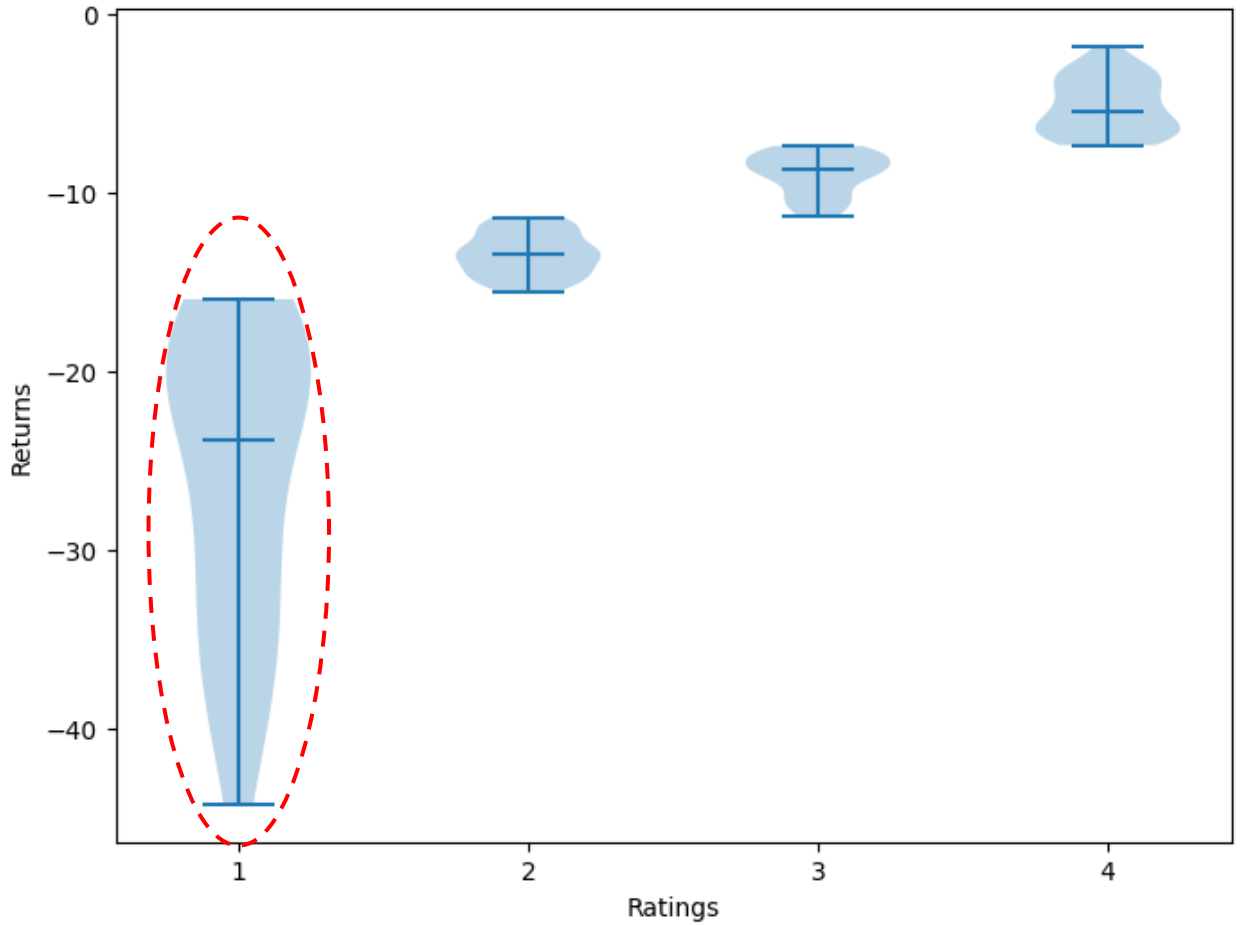
Alignment is ill defined in literature



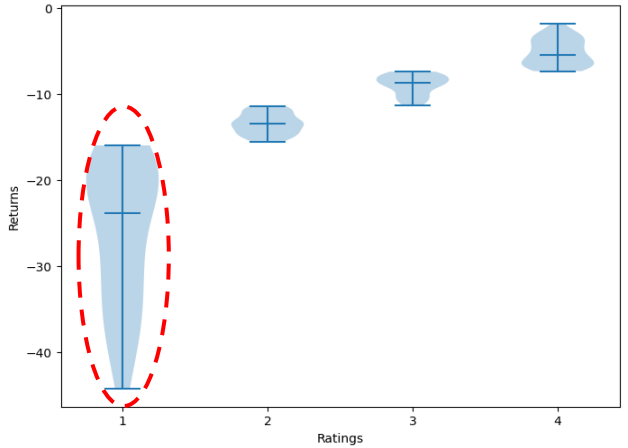
An Ideal Dataset

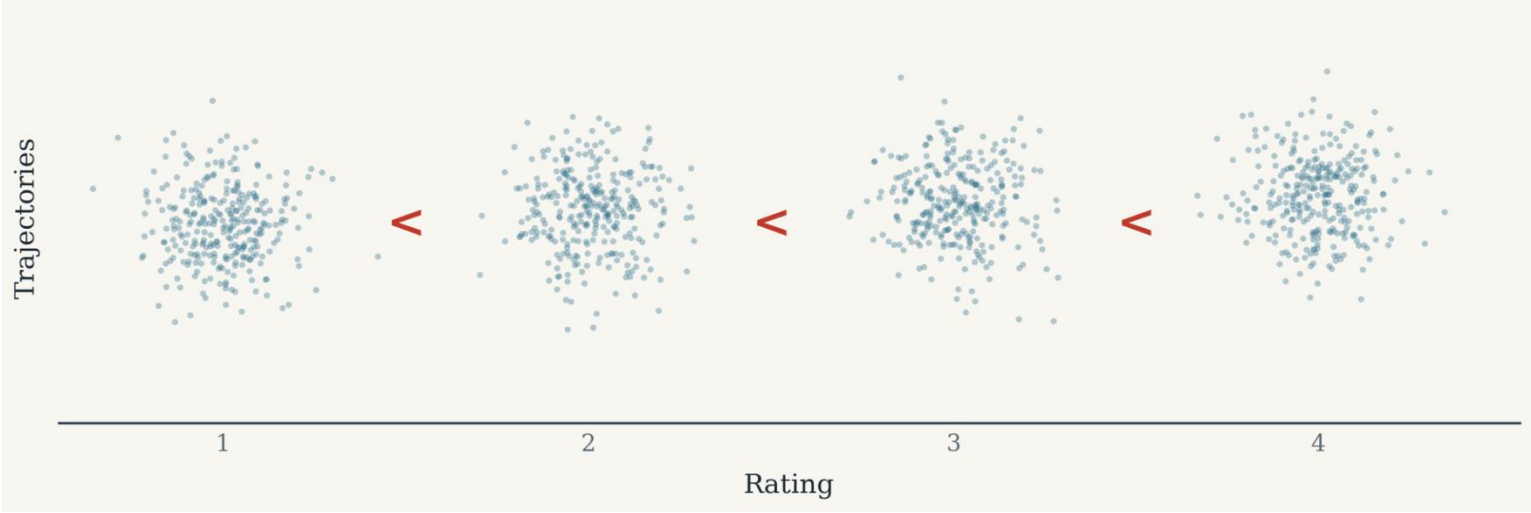


Algorithm's Viewpoint



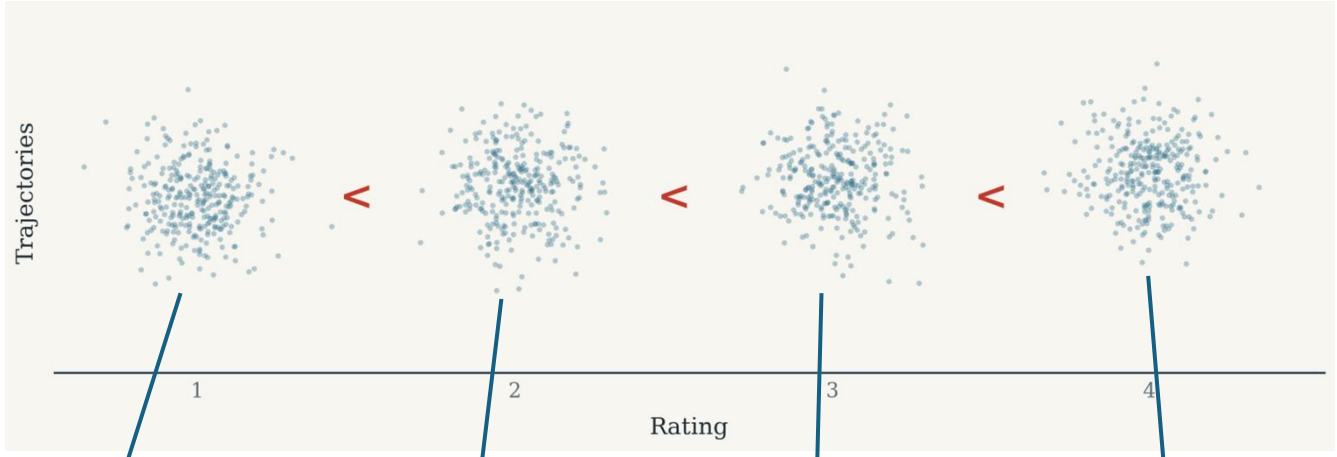
Algorithm's Viewpoint





$$\mathcal{D} = \{\tau_i, c(\tau_i)\}_{i=0}^{n-1}$$

Ratings To Ranks



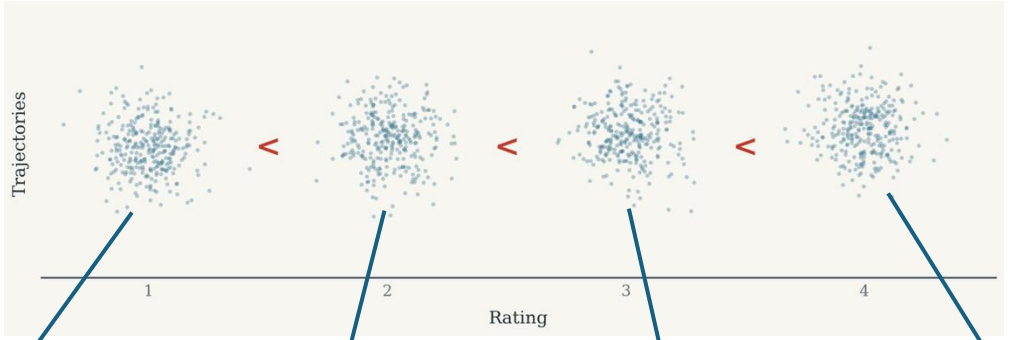
$$\text{Unknown } r^* \quad \{ \; \tau_a < \tau_b < \tau_c < \tau_d \; \}$$

$$r_\theta \quad \{ \; \tau_b < \tau_d < \tau_c < \tau_a \; \}$$

-0.34

Ranked Return Regression for RL (R4)

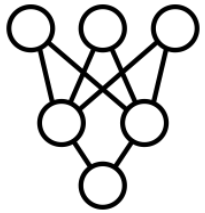
Ratings



Sample

τ_a τ_b τ_c τ_d

Forward Pass r_θ



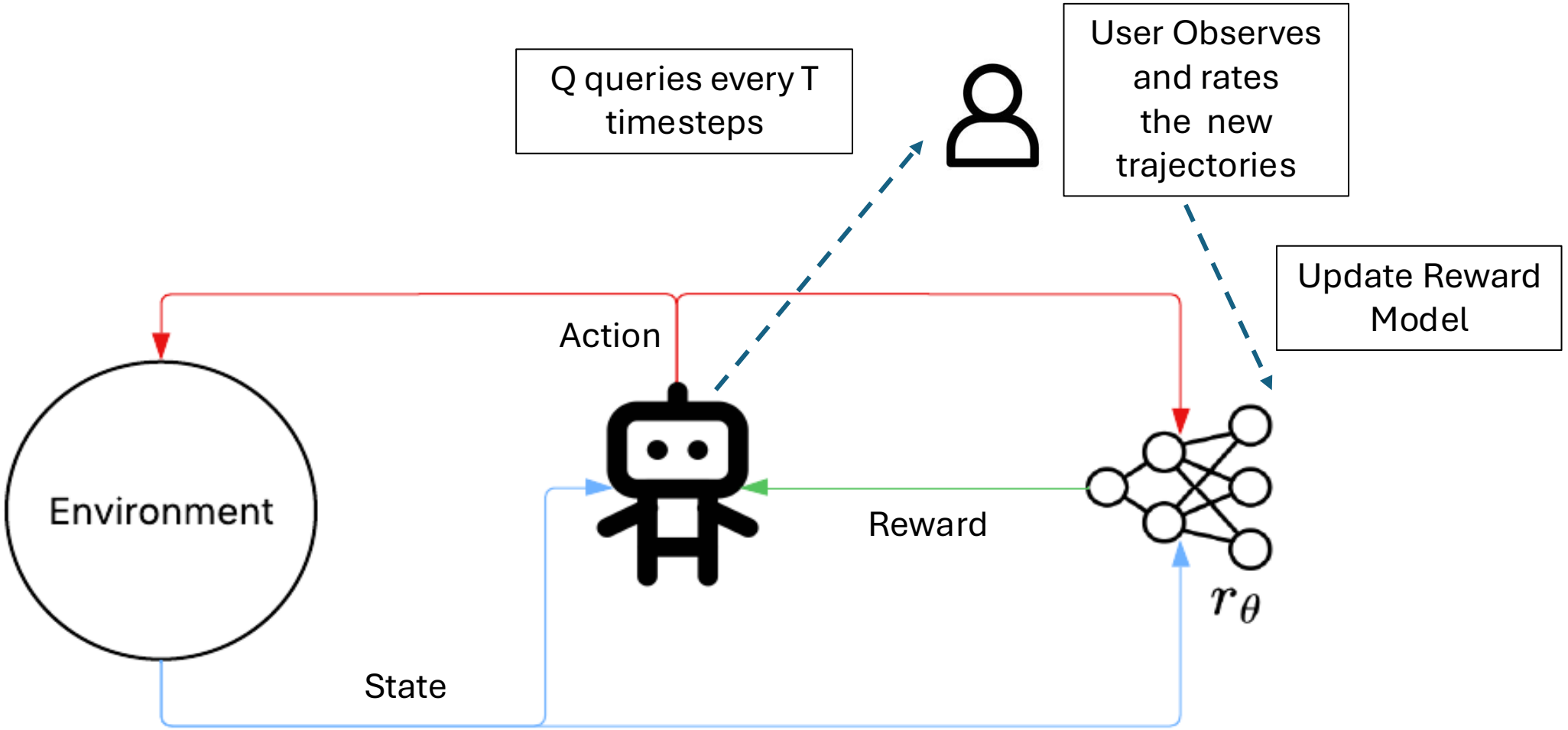
Predicted Returns

Ga	Gb	Gc	Gd

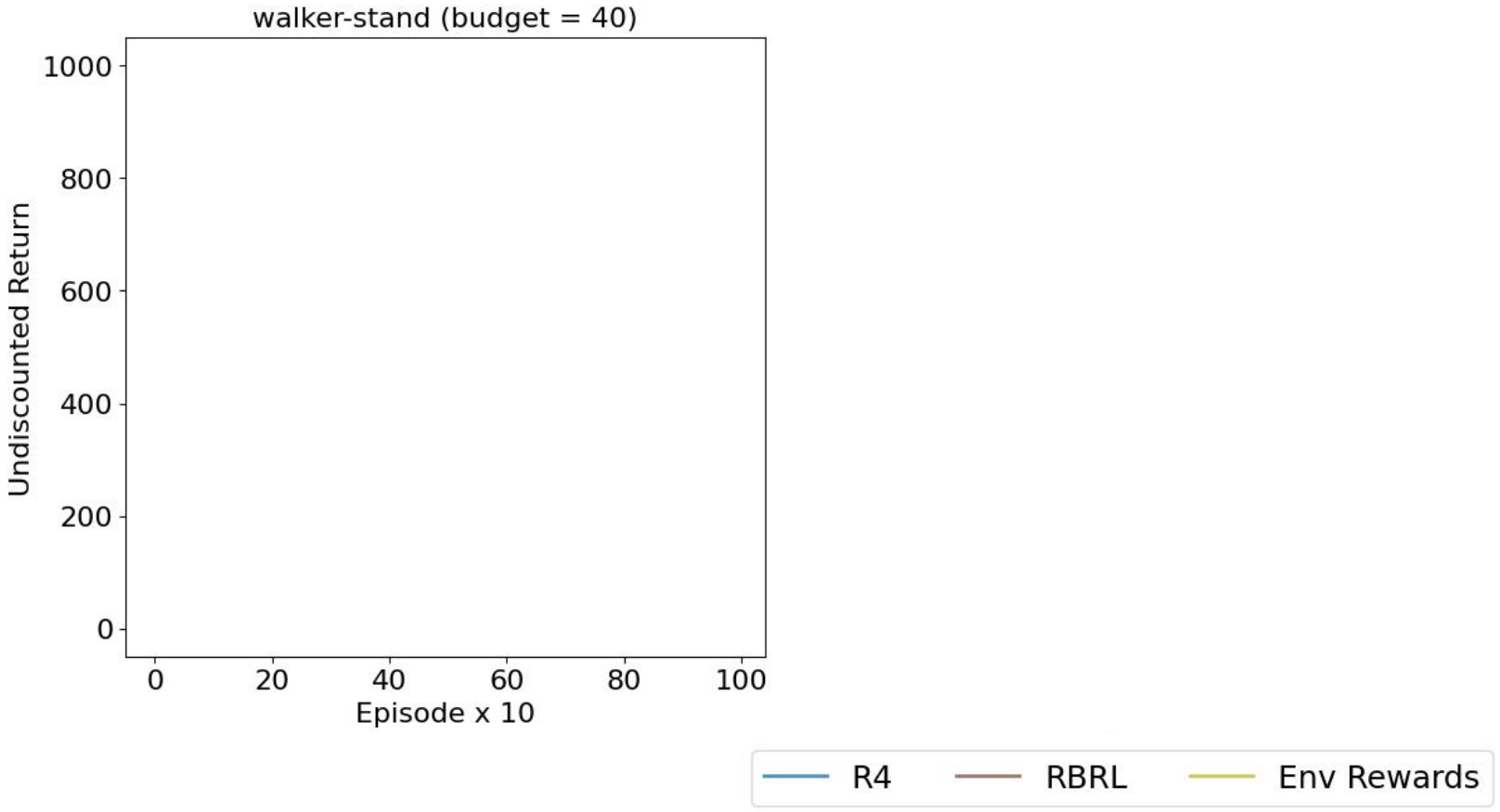
Theory Results

- R4's solution set always has the data generating reward function
  - This is not true for RbRL baseline, even in perfect, noise-free setting
- R4's loss function is the tightest possible loss function for rating-based dataset
  - To reduce R4's solution set further, the method will either have to:
    - Make and justify additional assumptions
    - Or possibly miss the data generating reward function

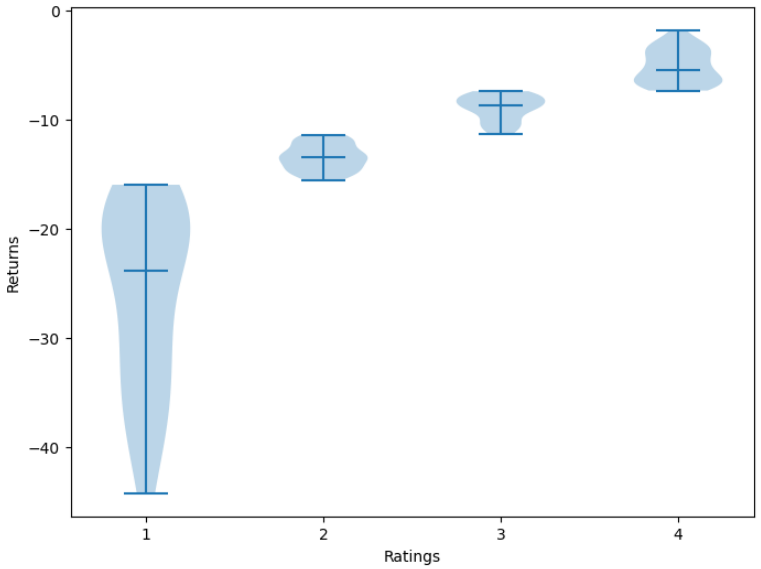
Online Reward Learning

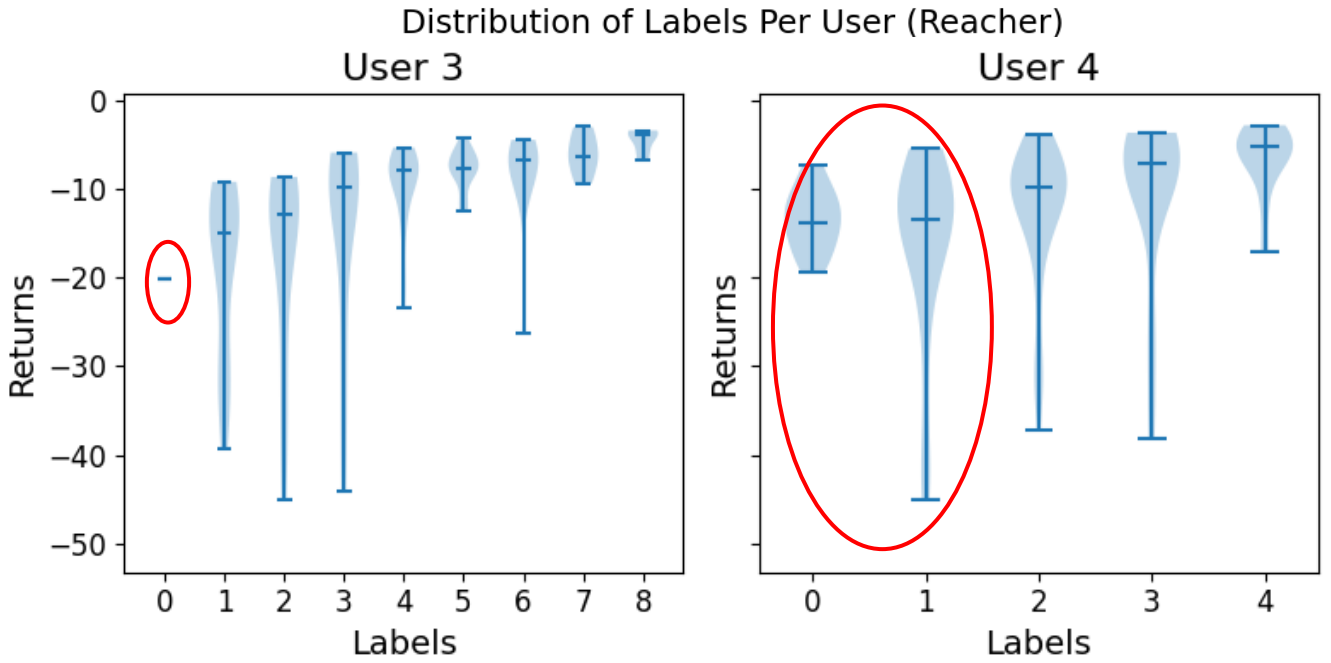


Empirical Results - Online



Empirical Results- Human Ratings



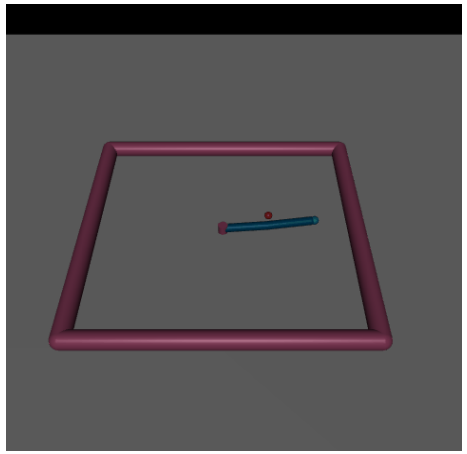


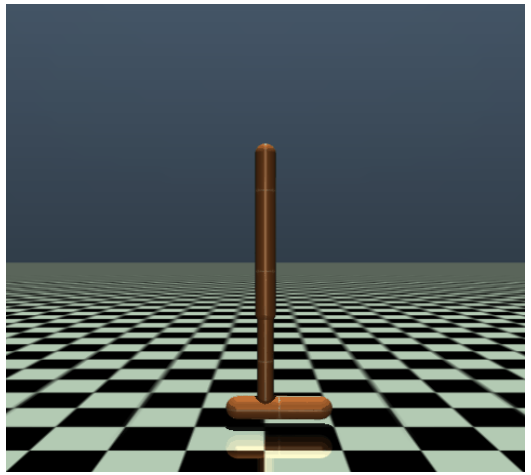
Study Design

- Ethics approved (REB:Pro00145121), eight participants.
- Two domains: Reacher and Hopper
- The participants were allowed to choose their own rating scale.

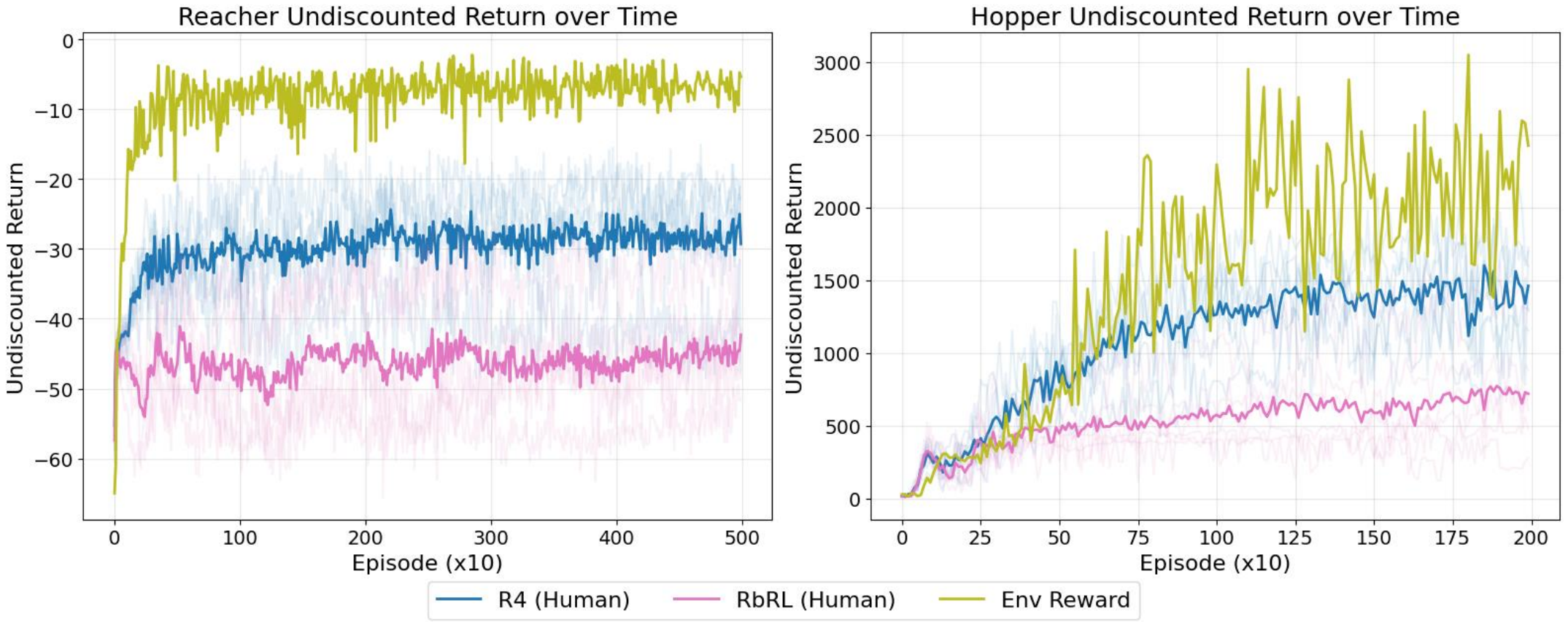
Study Design

- Ethics approved (REB:Pro00145121), eight participants.
- Two domains: Reacher and Hopper
- The participants were allowed to choose their own rating scale.

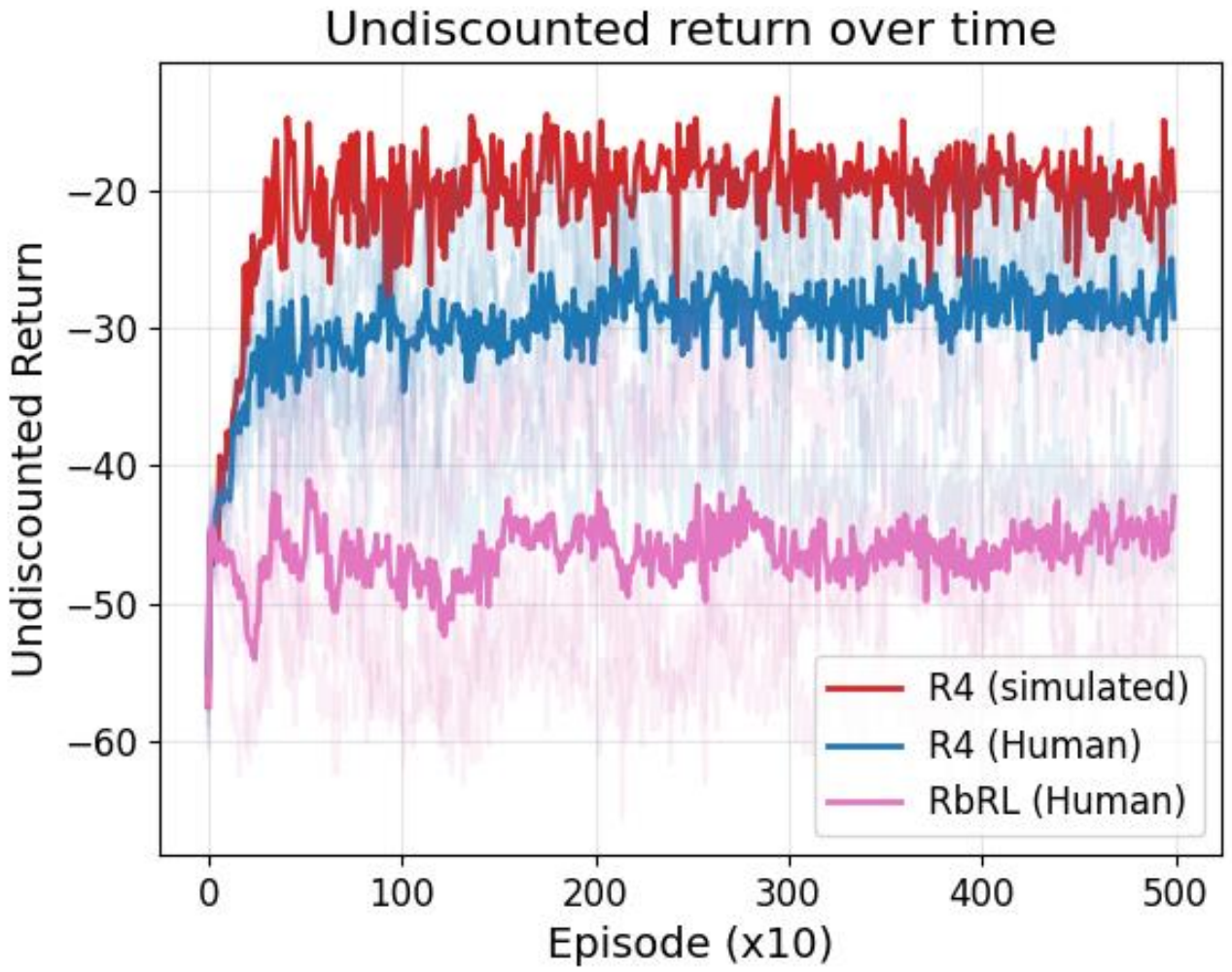




Empirical Results- Human Ratings



Empirical Results- Human Ratings



R4 Conclusions

- Ratings are a viable feedback method to learn reward functions from
- Key Insight: Ratings are ordinal, not just categorical
- R4 provides theoretical guarantees that the baselines fail to provide
- R4 outperforms existing rating, and preference-based methods, including with real human feedback

RL Research Settings

Given an MDP

- Learn as quickly as possible
- Get the highest performance

Atari

Robot Soccer, Go

Given a problem

- Design and MDP and solve it
- Learn under some new assumptions

TD Gammon

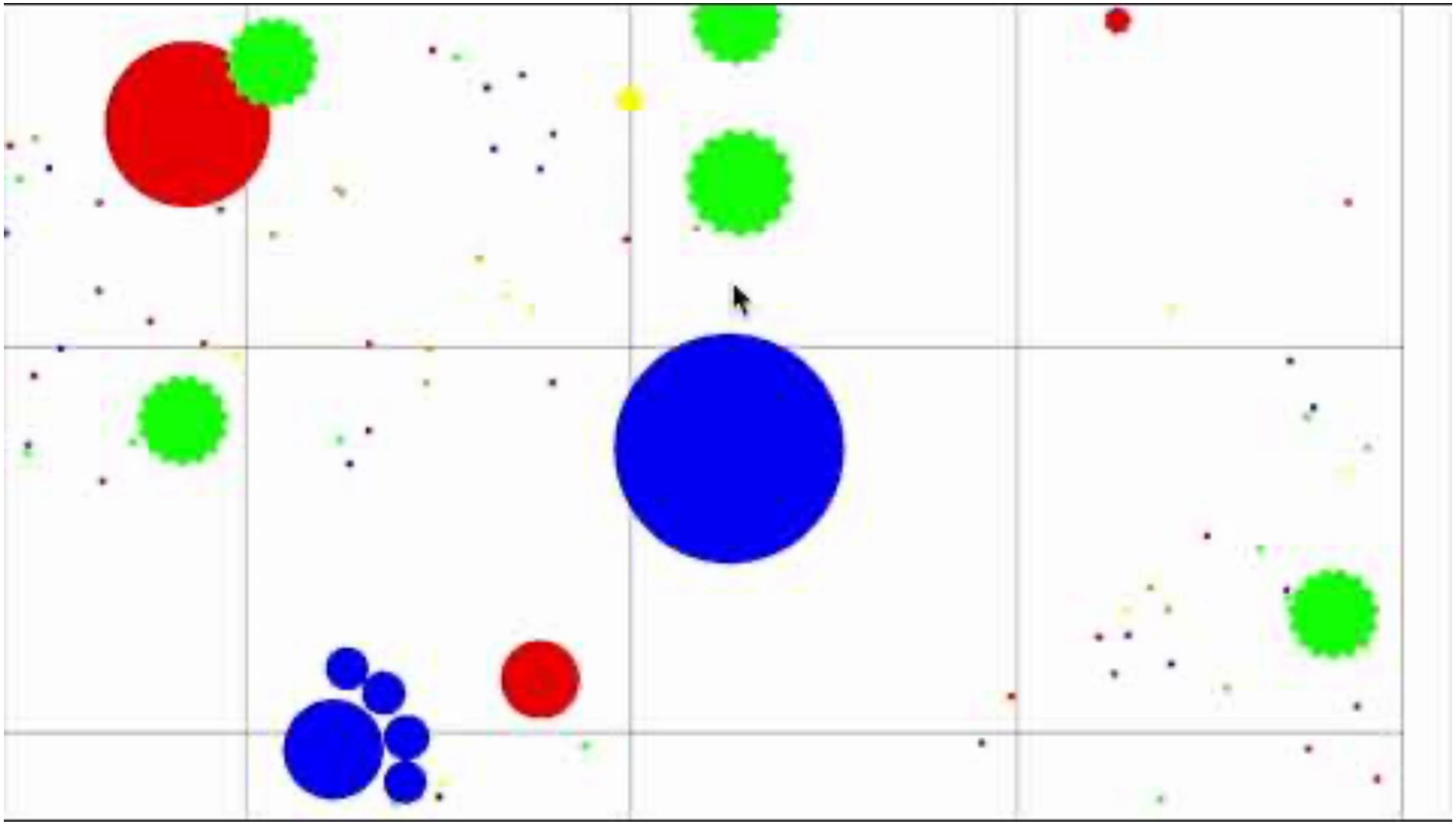
Gran Turismo, Table Tennis

Agar.io, Learn from a human

Design & solve a new problem

Stock trading, Chicken health

AgarCL



The Cell Must Go On: Agar.io for Continual Reinforcement Learning

Mohamed A. Mohamed, Kateryna Nekhomiazh, Vedant Vyas, Marcos M. Jose, Andrew Patterson, Marlos C. Machado

Outline

- Multi-arm Bandits
- RL Successes & Settings
- Tabular example
- Q-Learning and SARSA
- Models
- Function Approximation
- DQN

- Selection of research

- Better/worse RL problems

RL Strengths

Agent can autonomously learn to maximize rewards

Programmer just specifies goals

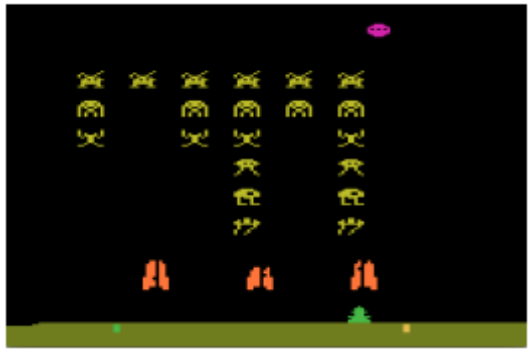
Often much less work than directly programming

Can achieve superhuman performance

Can handle unanticipated changes in the environment

This work is licensed under a Creative Commons Attribution 4.0 International License.

Deep RL Successes (Many in Games)



DQN - Atari

AlphaGO

AlphaStar - StarCraft

OpenAI Five - Dota

One Challenge: Sample Efficiency

OpenAI Five uses **180 years** of gameplay data **each day** across tens of thousands of simultaneous games, consuming **128,000 CPU** cores and **256 GPUs**

RL Weaknesses

Agent maximizes reward whether it's what you actually wanted or not!

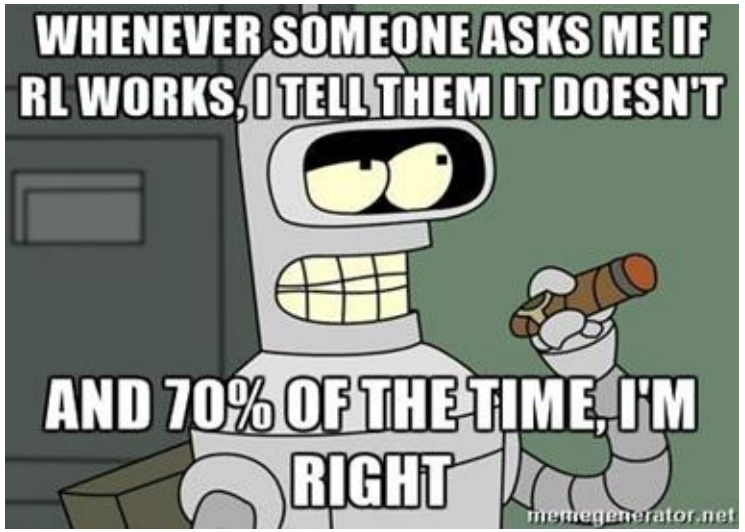
Can require lots of computation and/or interaction with the real world

- Interacting with the real world can have cost in terms of time, money, wear, etc.

Solutions are often black box

- Explainability is not well understood (yet)

Initial performance could be very poor



Source:https://www.alexirpan.com/2018/02/14/rl-hard.html

Challenges of Real-World Reinforcement Learning
Gabriel Dulac-Arnold, Daniel Mankowitz, Todd Hester

An empirical investigation of the challenges of real-world reinforcement learning

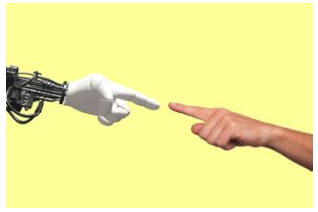
Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, Todd Hester

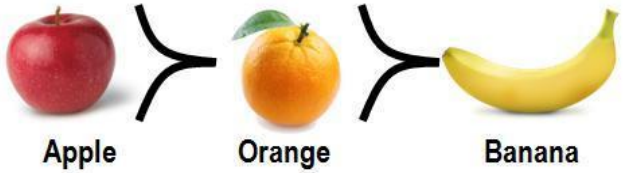
Initial Performance

How bad are random actions?

- What is the cost of exploring

Can you use an existing agent or a human?





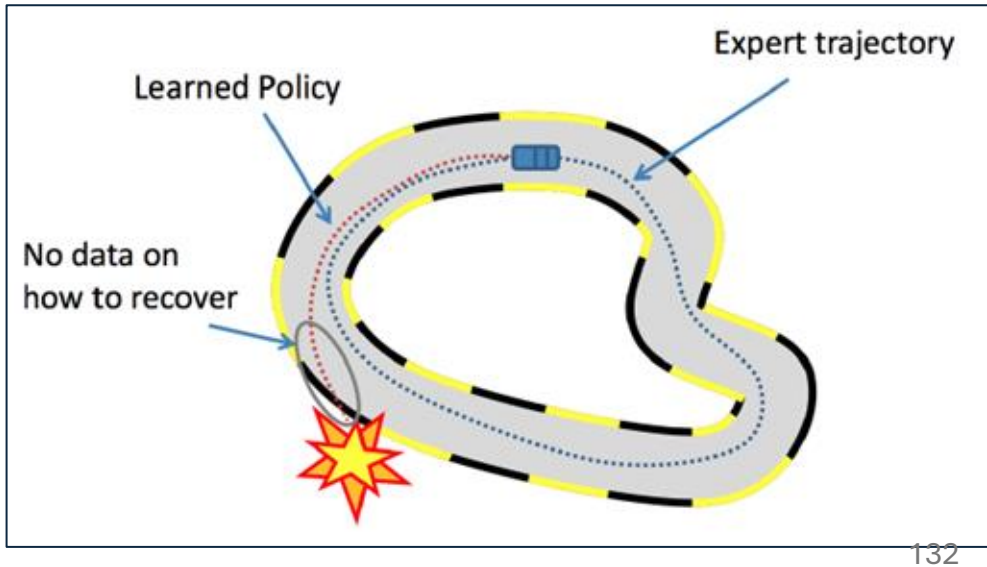
Batch and Offline Training

Just need a data set

- Could be generated by agent, program, or human
- Typically available if "improving" instead of "innovating"
- Jumpstart learning, ideally outperform the legacy system

Approach: Train policy from non-RL controller with supervised learning

- May visit different parts of state space



Limited Samples

Parallelization helps, but need simulator

For many real-world problems, there is no simulator available

- Balancing exploration vs. exploitation, with real world costs

Training could be too slow to be useful (non-stationarity)

Safety Constraints

Maximizes expected return

- E.g., number of points in an Atari game
- Does not enforce "what not to do" (critical states)
- Could lead to unintended behaviors
- May not correctly factor in risk

Could use action filter

Might have access to safe policy

Reward Function

Can be difficult to define

- Additional rewards can help for faster learning
- But… might discover high reward policy without doing the intended task!
- High-scoring, counter-intuitive behaviors

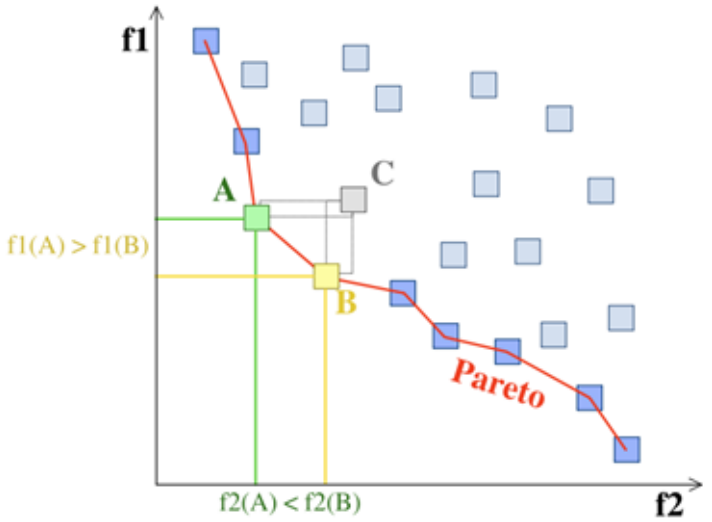
MDP reward

- Expert elicitation?
- User preferences?
- Shaping reward from expert?

Multi-Objective Reward Functions

Many problems have naturally (conflicting) multi-objective rewards

- Navigate from A to B quickly, with minimal fuel
- Pareto optimality
- Preference elicitation from user / expert?



Explainability / Interpretability

Explainability for failure cases

- Helpful for debugging

Helps to build trust with users / encourage adoption

Might learn new strategies from the agent

- AlphaGo employed unseen strategies in Go

Summary: What Makes an RL Task Harder?

Size of the state space: larger is often harder

Are there multiple agents you need to interact with or reason about?

Can you see the true state of the world or just an approximation?

Do you have to work in the real world, without a simulator?

Is there no existing data you can bootstrap off of?

Does the reward need to be carefully designed to elicit desired behavior?

Summary: Targeting a Good RL Task

Sequential decision task & can't use planning

How expensive are failures (e.g., from exploration or early performance)?

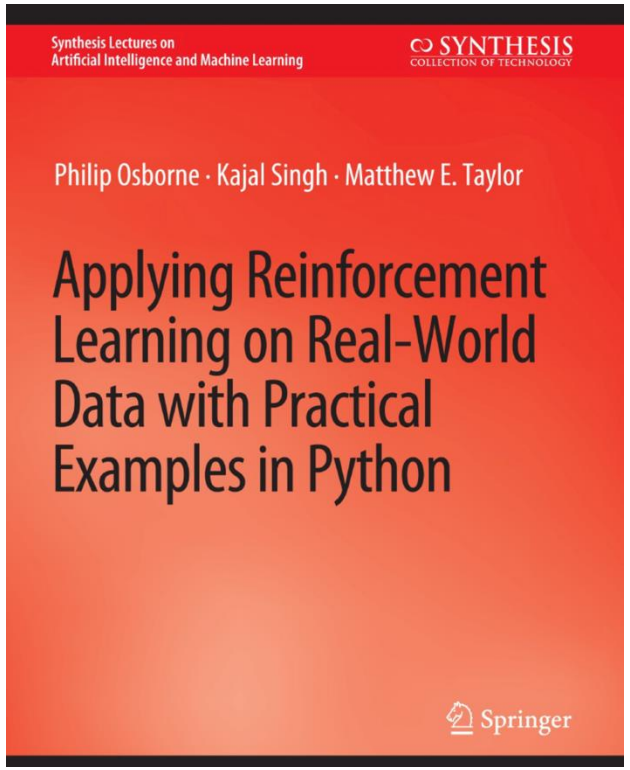
What improvement can we expect over state of the art?

Are there second-order effects (e.g., marketing)?

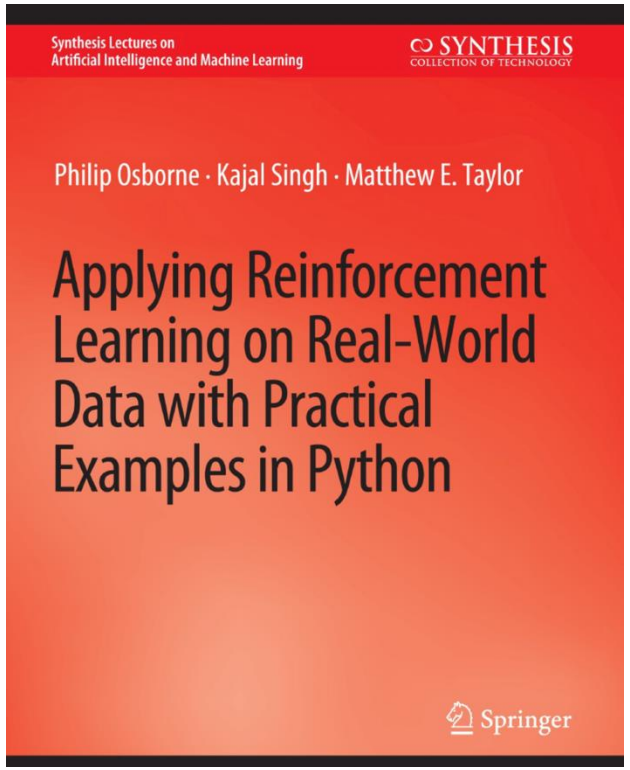
How difficult will it be to set up an environment where an agent can get the data it needs and execute actions?

Can I first tackle an abstract version of the task to understand how difficult it is?

Where to learn more?



Where to learn more?



RL

- Coursera RL specialization from U Alberta
  - https://www.coursera.org/specializations/reinforcement-learning
- Udacity class from Georgia Tech
  - https://www.udacity.com/course/reinforcement-learning--ud600
- THE book on RL (Sutton & Barto, 2018)
  - http://www.incompleteideas.net/book/the-book-2nd.html
- Csaba Szepesvári: Algorithms in Reinforcement Learning
  - https://sites.ualberta.ca/~szepesva/rlbook.html

Deep RL

- Class on YouTube from UCL/Deepmind
  - https://www.youtube.com/playlist?list=PLqYmG7hTraZDNJre23vqCGIVpfZ_K2RZs
- OpenAI Spinning Up in Deep RL
  - https://spinningup.openai.com/en/latest/