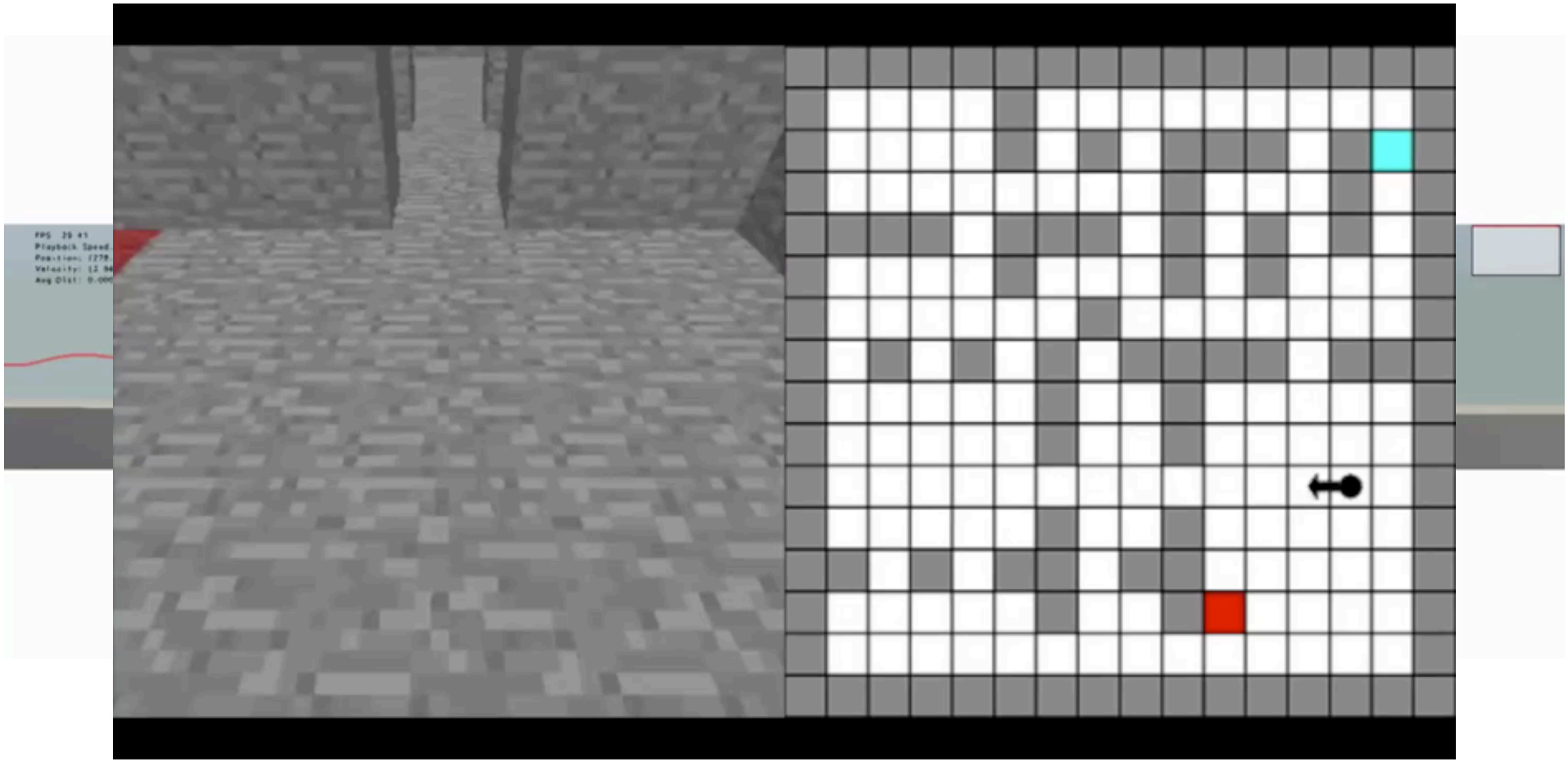
Cooperation and Communication in Multiagent Deep Reinforcement Learning

Matthew Hausknecht

Nov 28, 2016

Advisor: Peter Stone

Motivation



Thesis Question

How can the power of Deep Neural Networks be leveraged to extend Reinforcement Learning towards domains featuring partial observability, continuous parameterized action spaces, and sparse rewards?

How can Deep Reinforcement Learning agents learn to cooperate in a multiagent setting?

Contributions

- Half Field Offense Enivronment
- Deep RL in parameterized action space
- Multiagent Deep RL
- Deep Recurrent Q-Network (DRQN)
- Curriculum learning in HFO

Outline

1. Background

2. Deep Reinforcement Learning

3. Multiagent Architectures

4. Communication

Markov Decision Process

State s_t

Action a_t

Reward r_t

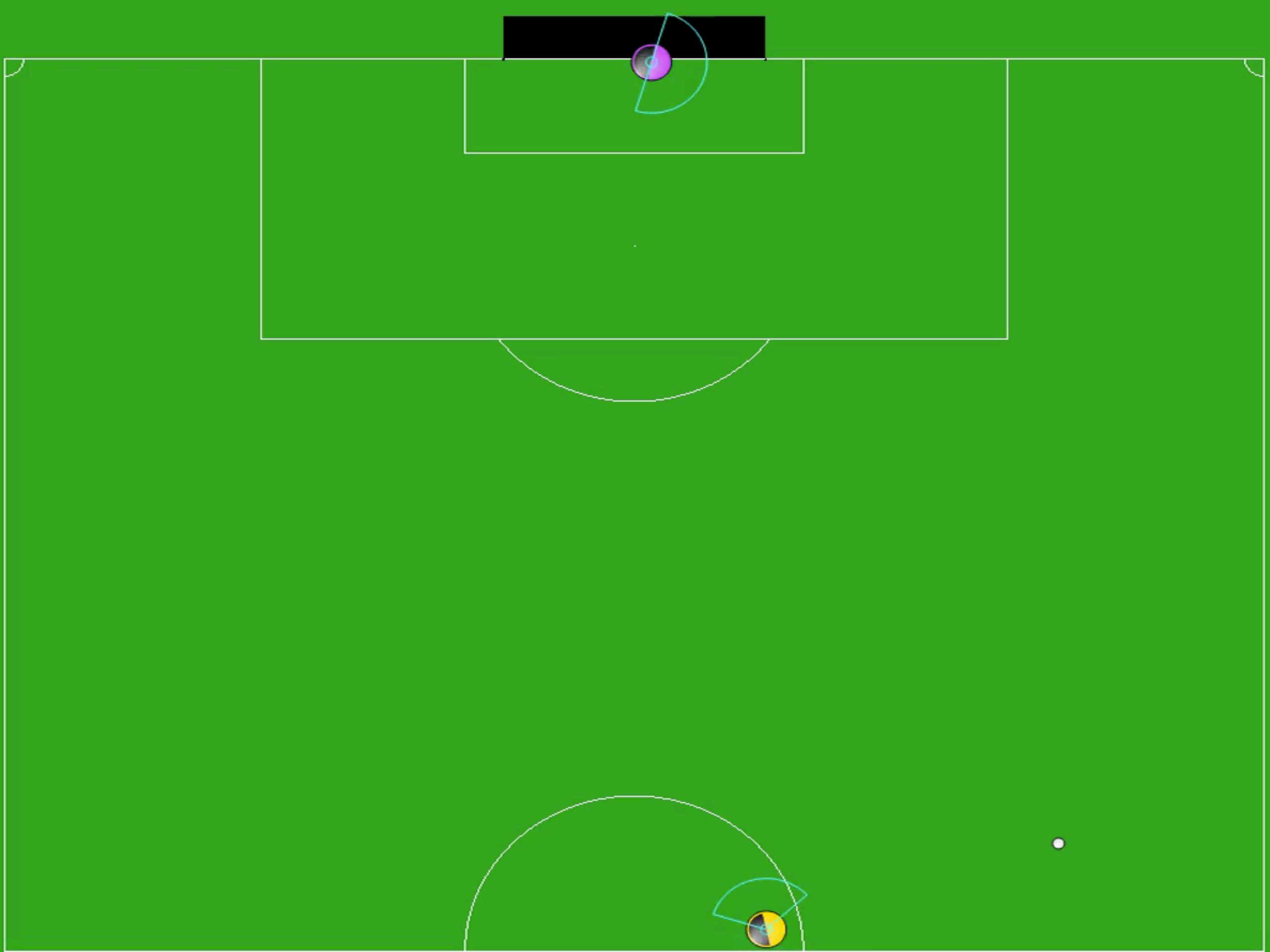
Formalizes the interaction between the agent and environment.

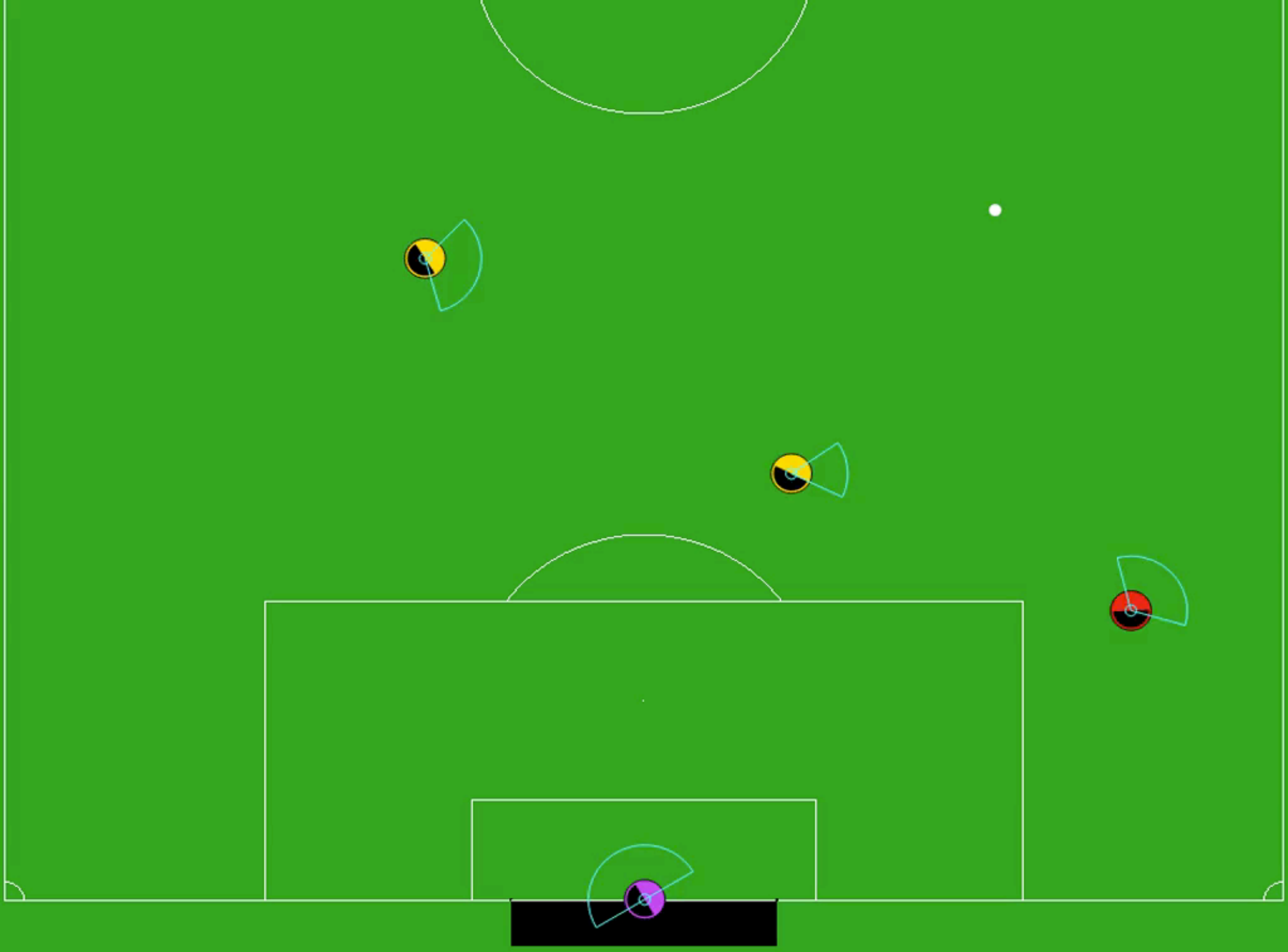
Half Field Offense

Cooperative multiagent soccer domain built on the libraries used by the RoboCup competition

Objective: Learn a goal scoring policy for the offense agents

Features continuous actions, partial observability, and opportunities for multi agent coordination



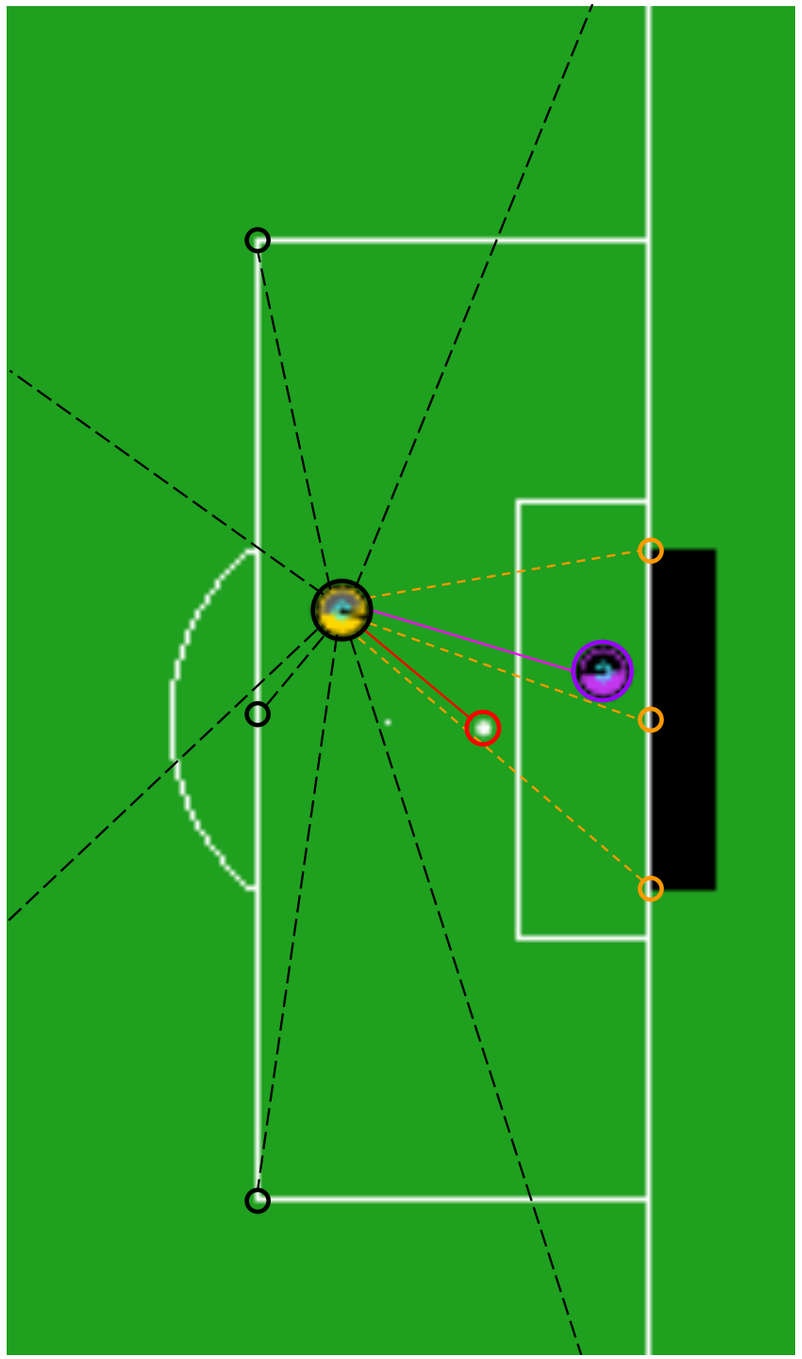


State Action Spaces

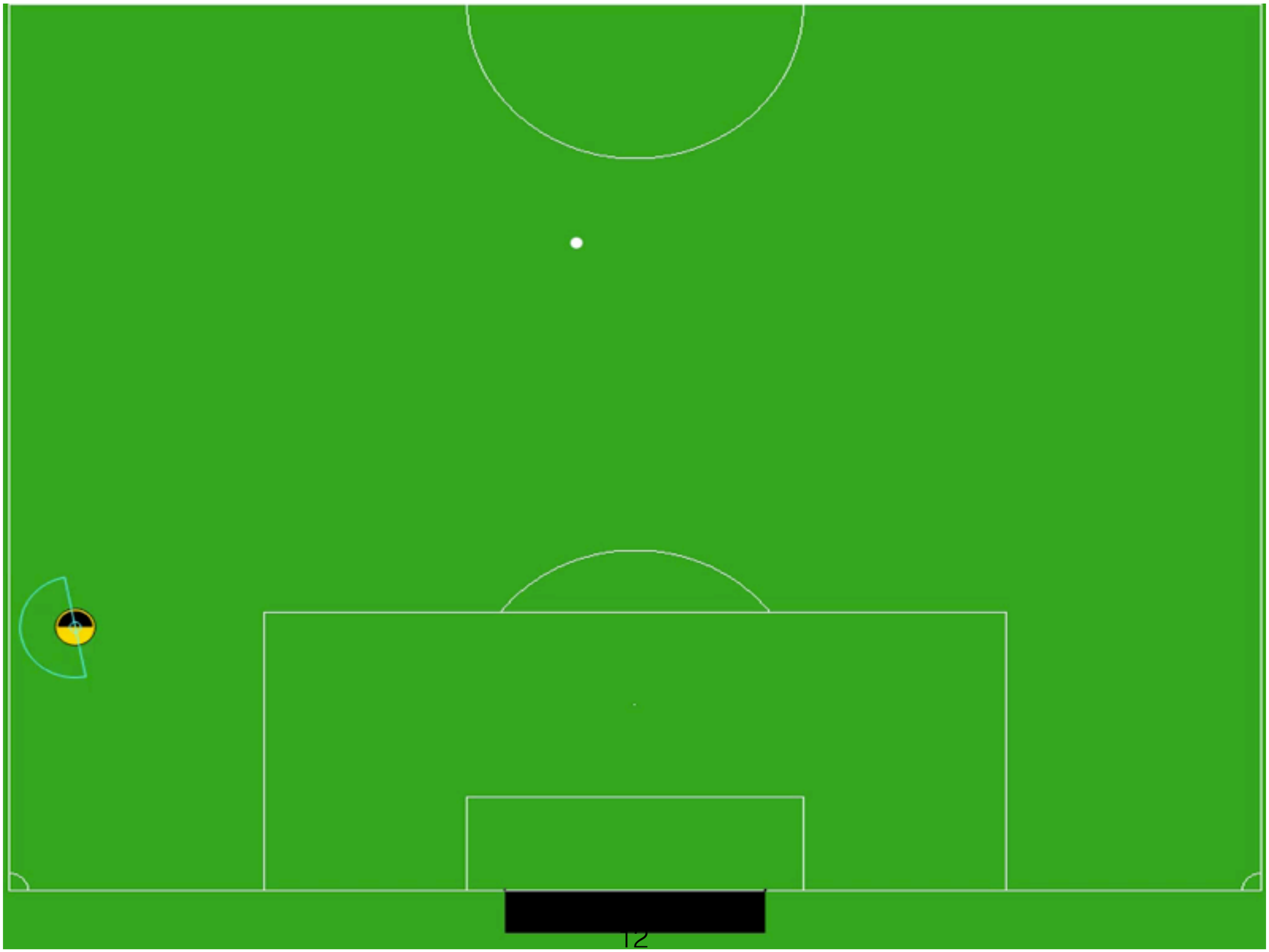
58 continuous state features encoding distances and angles to points of interest

Parameterized-Continuous Action Space:
Dash(direction, power)
Turn(direction)
Tackle(direction)
Kick(direction, power)

Choose one discrete action + parameters every timestep



Learning in HFO is difficult



Reinforcement Learning

Reinforcement Learning provides a general framework for sequential decision making.

Objective: Learn a policy that maximizes discounted sum of future rewards.

Deterministic policy π is a mapping from states to actions.

For each encountered state, what is the best action to perform.

Q-Value Function

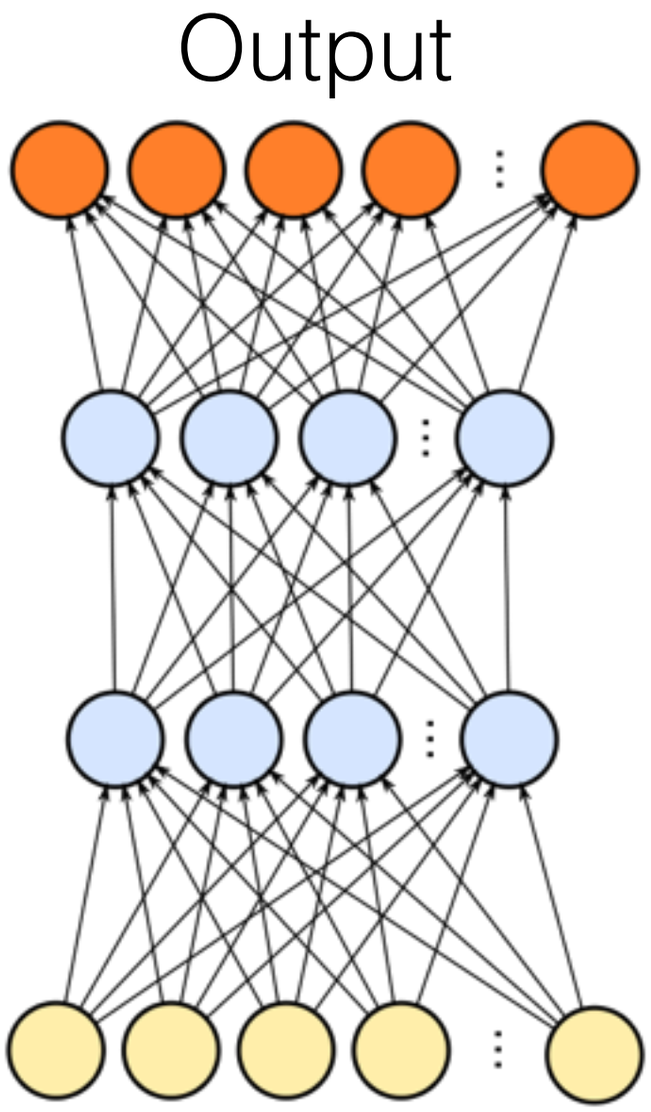
Estimates the expected return from a given state-action:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s, a\right]$$

Answers the question: “How good is action *a* from state *s*.”

Optimal Q-Value function yields an optimal policy.

Deep Neural Network

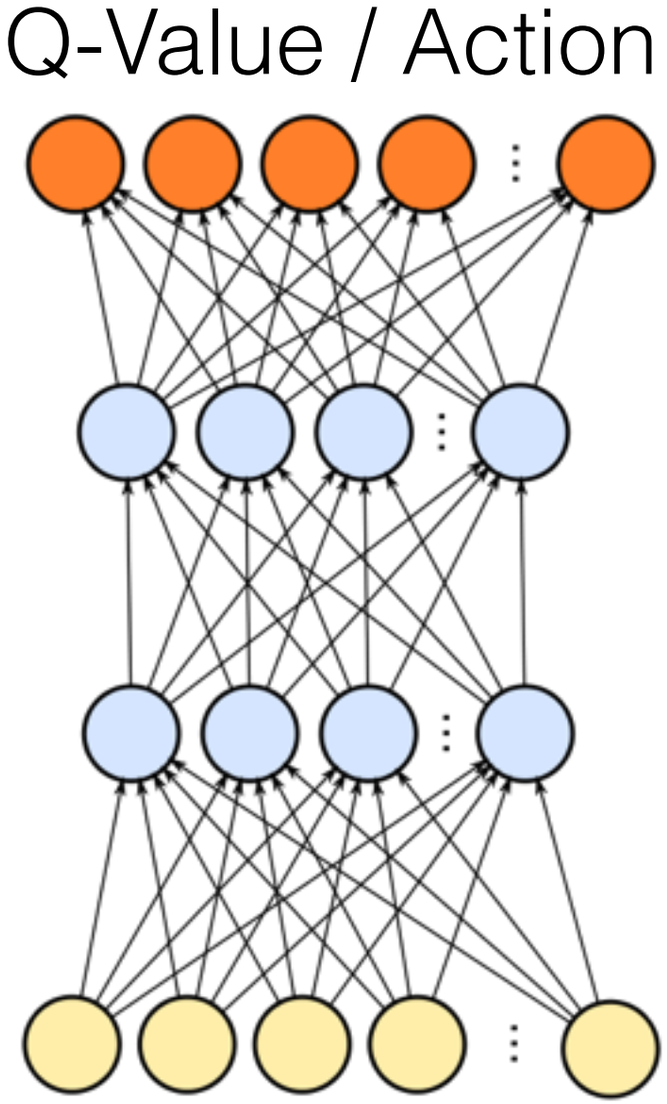


Parametric model with stacked layers of representation.

Powerful, general purpose function approximator.

Parameters θ optimized via backpropagation.

Deep Reinforcement Learning



Neural network used to approximate Q-Value function and policy π.

Replay Memory: a queue of recent experience tuples *(s,a,r,s')* seen by agent.

Updates to network are done on experience sampled randomly from replay memory.

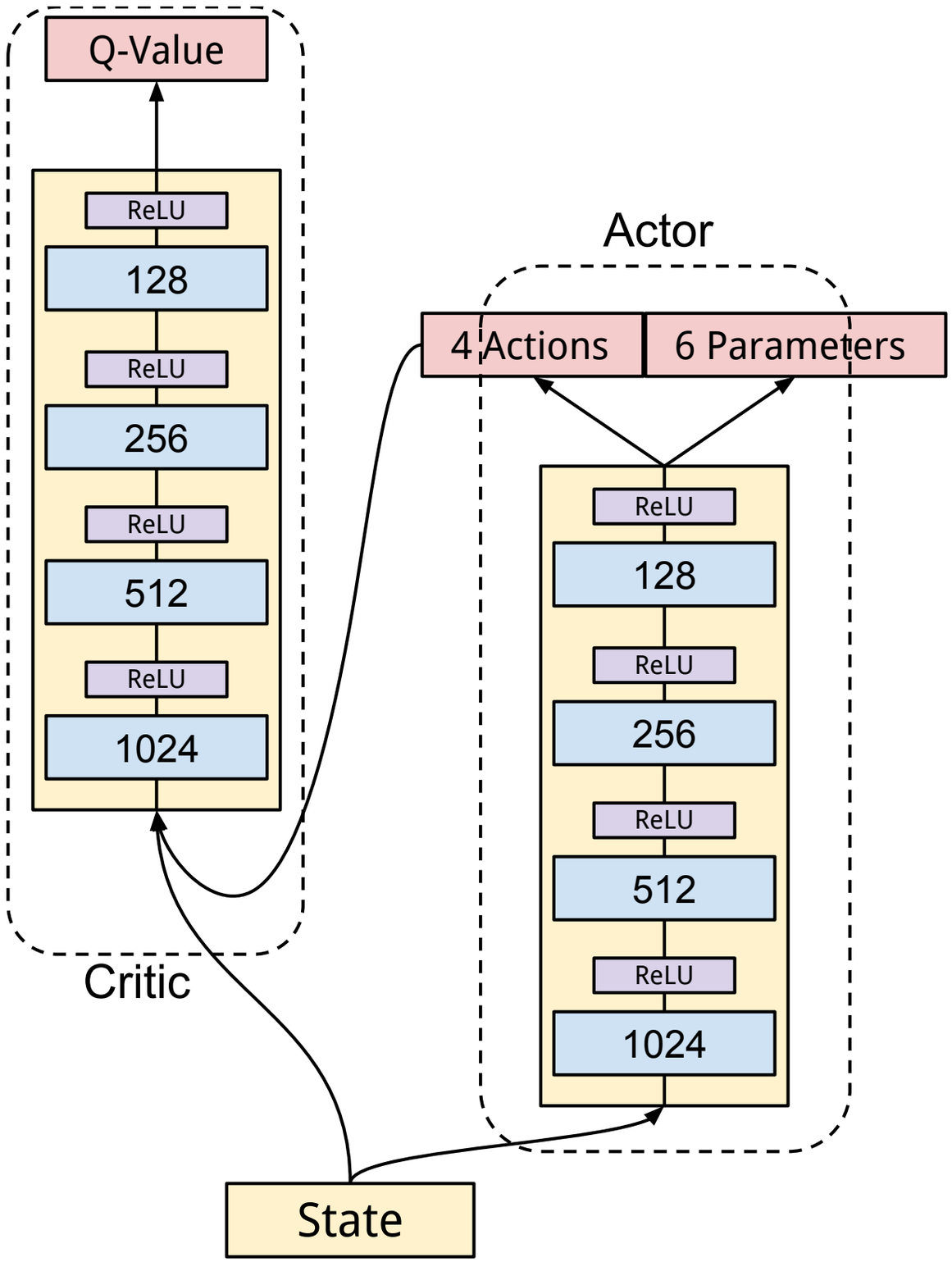
Deep Deterministic Policy Gradients

Model-free Deep Actor Critic architecture *[Lillicrap '15]*

Actor learns a policy π, Critic learns to estimate Q-values

Actor outputs 4 actions + 6 parameters.

a_t = max(4 actions) + associated parameter(s)



Training

Critic trained using temporal difference:

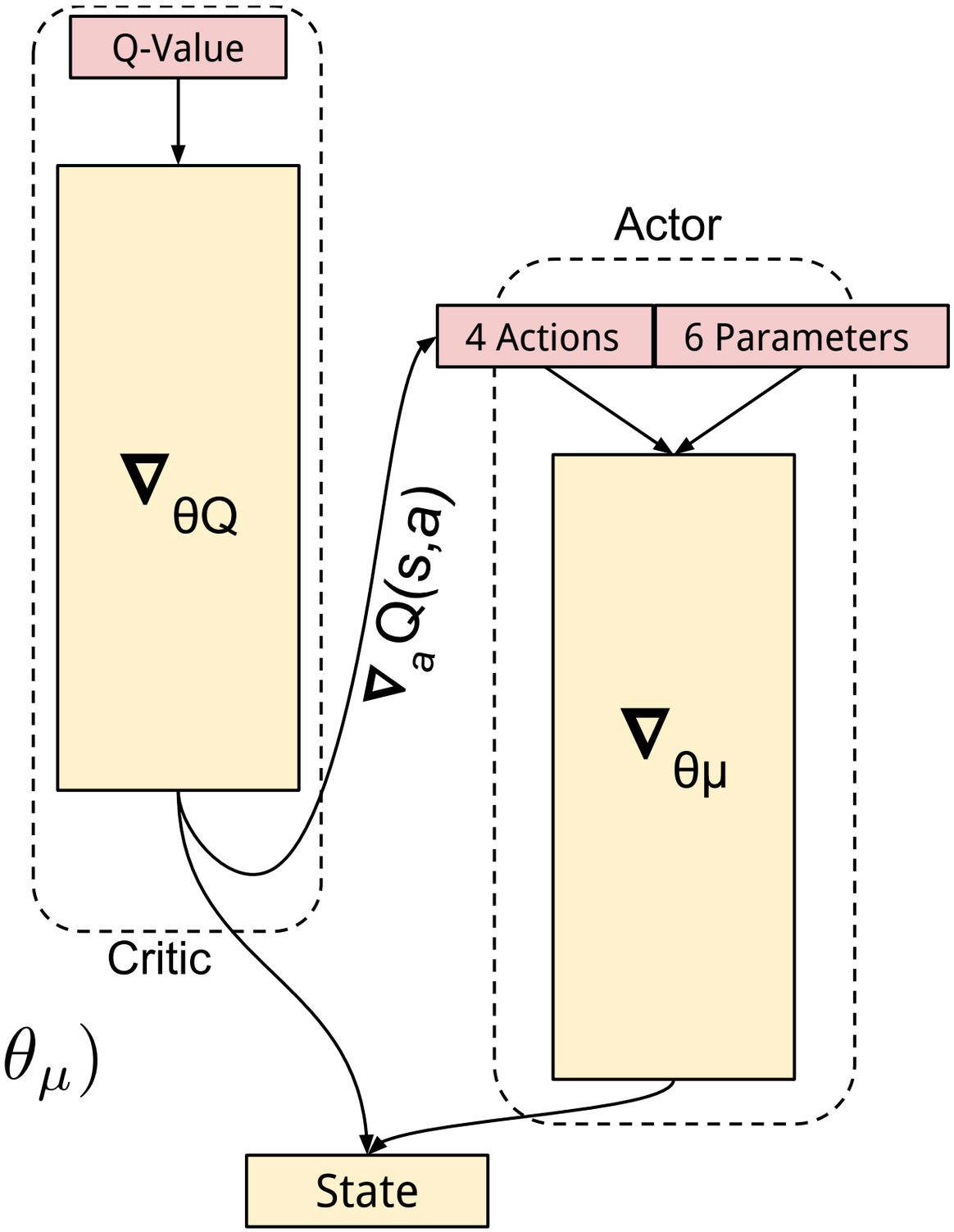
Given Experience (s_t, a_t, r_t, s_{t+1})

$$L = (Q(s_t, a_t|\theta_Q) - y)^2$$

$$y = r_t + \gamma(Q(s_{t+1}, \mu(s_{t+1})|\theta_Q))$$

Actor trained via Critic gradients:

$$\nabla_{\theta_\mu}\mu(s) = \nabla_a Q(s, a|\theta_Q)\nabla_{\theta_\mu}\mu(s|\theta_\mu)$$



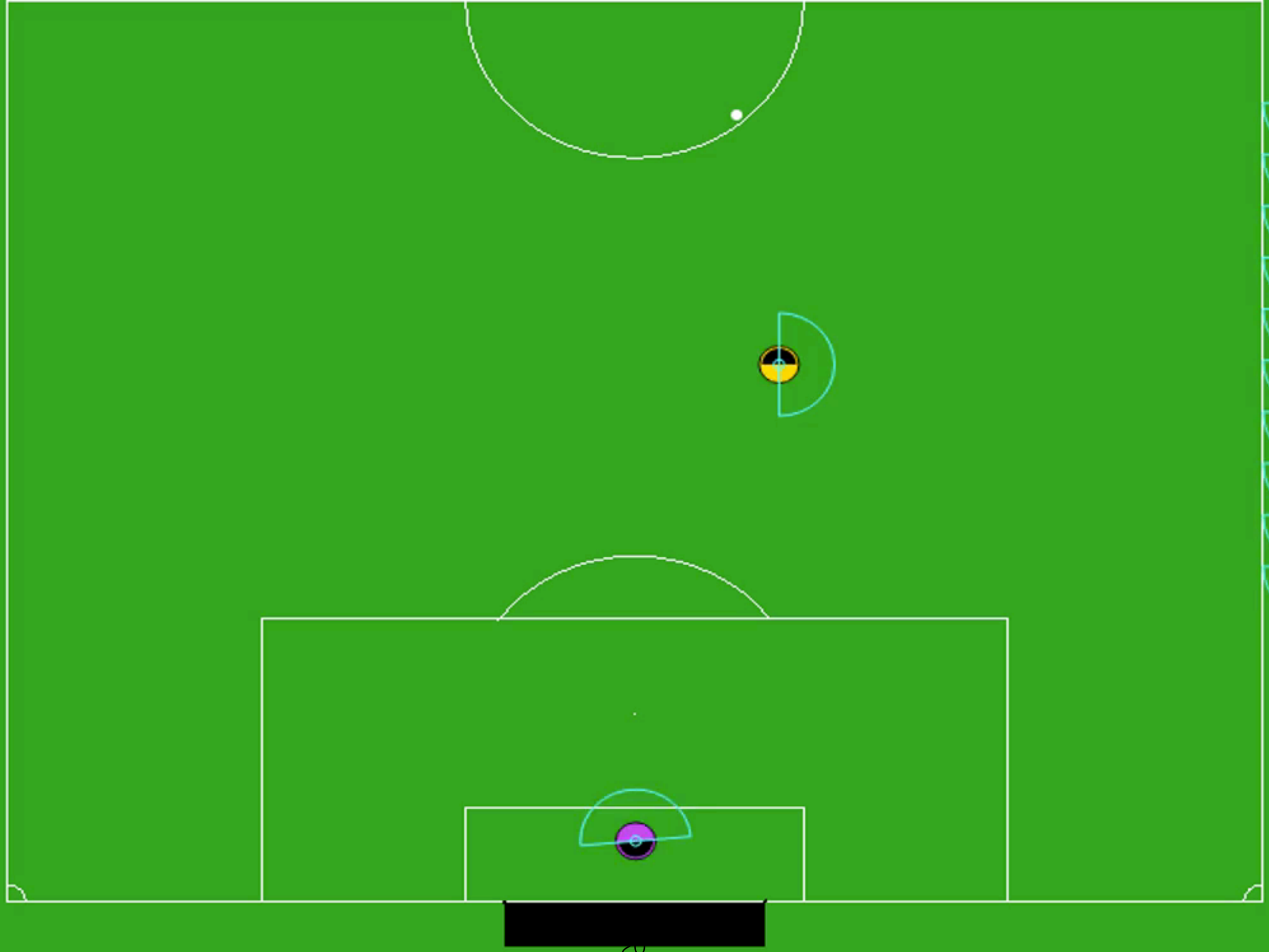
Reward Signal

With only goal-scoring reward, agent never learns to approach the ball or dribble.

$$r_t = -\Delta d(\text{Agent, Ball}) + I^{\text{kick}} + -3\Delta d(\text{Ball, Goal}) + 5I^{\text{Goal}}$$

Go to Ball

Kick to Goal



Results

	Scoring Percent	Avg. Steps to Goal
DDPG_1	1.0	108.0
DDPG_2	.99	107.1
DDPG_3	.98	104.8
DDPG_4	.96	112.3
Helios' Champion	.96	72.0
DDPG_5	.94	119.1
DDPG_6	.84	113.2
SARSA	.81	70.7
DDPG_7	.80	118.2

[*Deep Reinforcement Learning in Parameterized Action Space*, Hausknecht and Stone, in ICLR '16]

Offense versus keeper

Automated Helios goal keeper is quite effective at stopping shots.

Independently created by Helios RoboCup team.

DDPG fails to reliably score against keeper.

Better value estimates

Q-Learning is known to overestimate Q-Values *[Hasselt ’16]*.

Several approaches have been found, but don’t always extend to an actor/critic framework.

We will show that mixing off-policy updates with on-policy Monte-Carlo updates yields quicker, more stable learning.

Q-Learning Spectrum

Q-Learning is a bootstrap off-policy method:

$$Q(s_t, a_t|\theta) = r_{t+1} + \gamma \max_a Q(s_{t+1}, a|\theta)$$

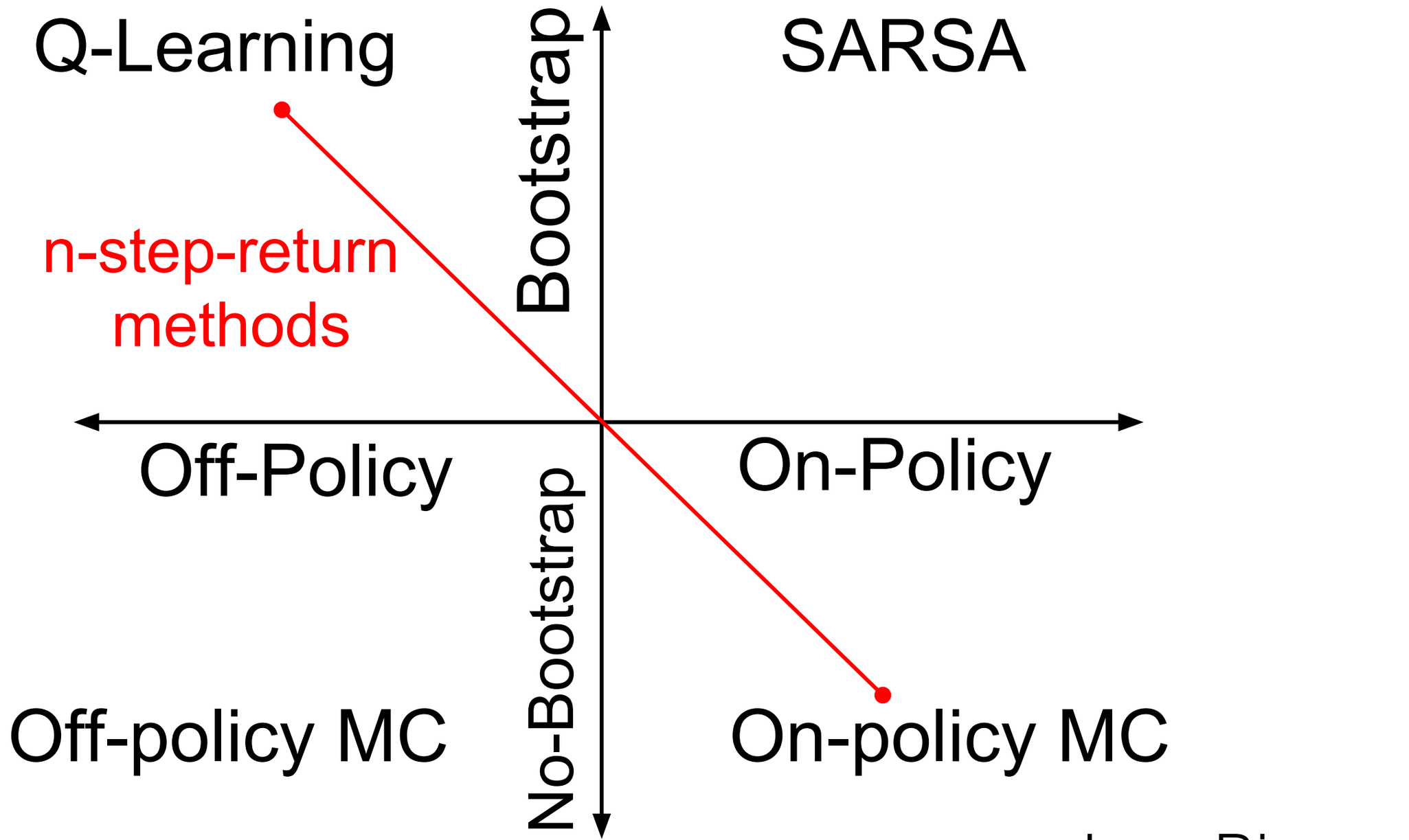
N-step Q-learning *[Watkins '89]*:

$$Q(s_t, a_t|\theta) = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max_a Q(s_{t+n}, a|\theta)$$

On-Policy Monte-Carlo updates are on-policy, non-bootstrap:

$$Q(s_t, a_t|\theta) = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^T r_T$$

High Bias
Low Variance



Low Bias
High Variance

On-Policy Monte Carlo

On-Policy Monte-Carlo updates make sense near the beginning of learning, since $\max_a Q(s_{t+1}, a|\theta)$ is nearly always wrong.

After Q-Values are refined, off-policy, bootstrap updates more efficiently utilize experience samples.

A middle path is to mix both update types:

$$y = \beta \; y_{\text{on-policy-MC}} + (1 - \beta) \; y_{\text{1-step-q-learning}}$$

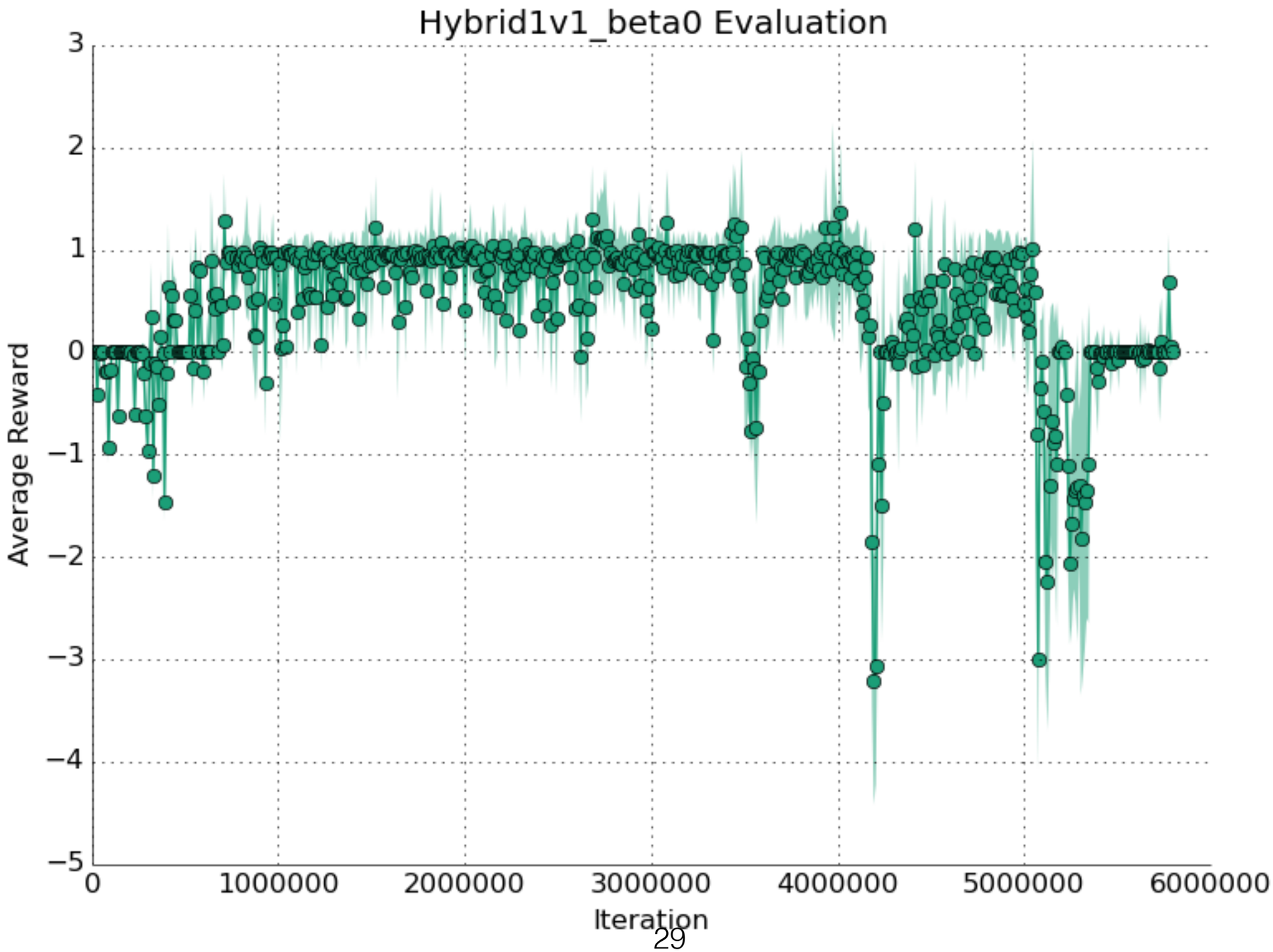
Experiments

Trained on 1v1 task

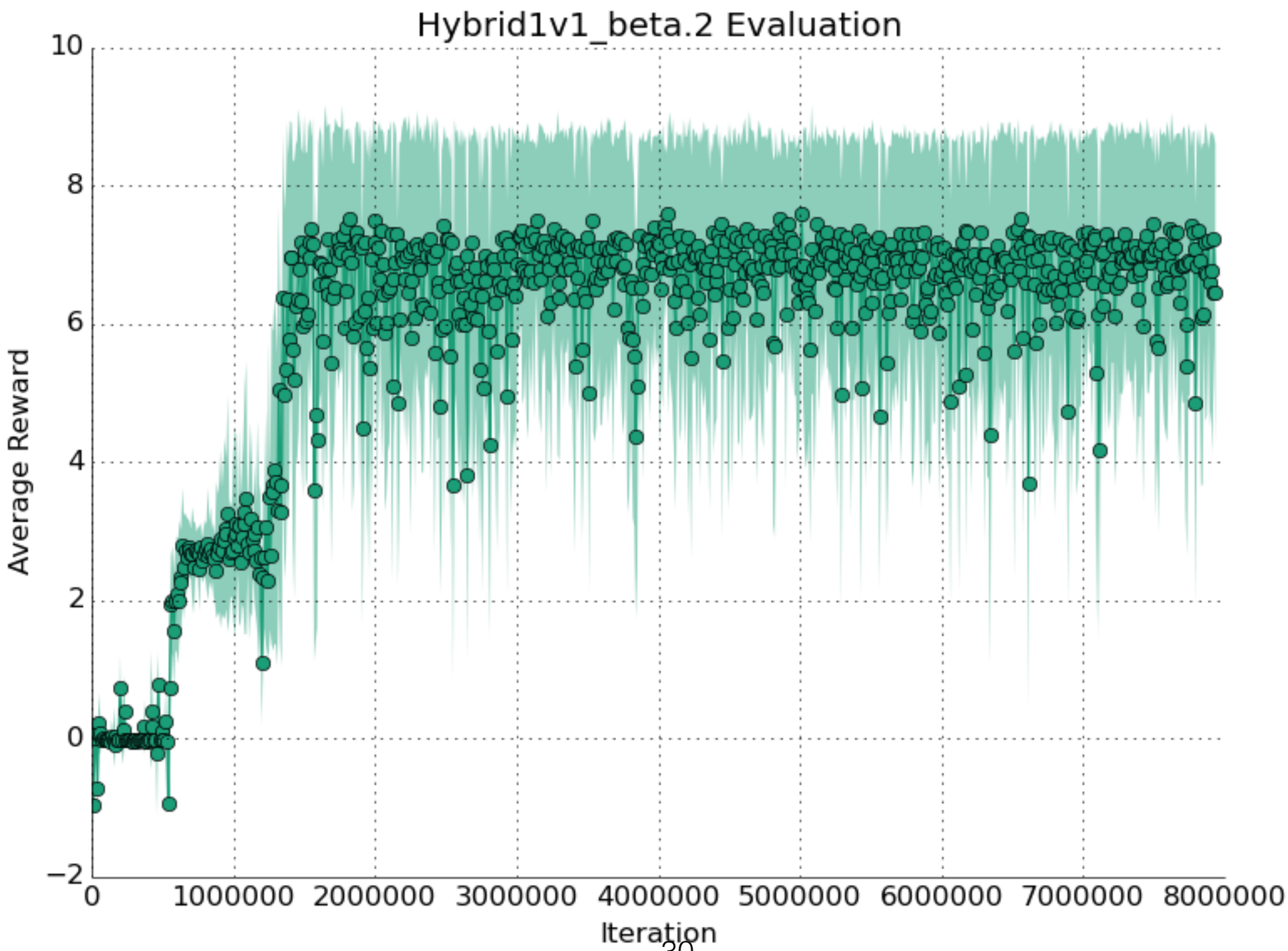
We evaluated 5 different β values: 0, .2, .5, .8, 1

$$y = \beta\ y_{\text{on-policy-MC}} + (1 - \beta)\ y_{\text{1-step-q-learning}}$$

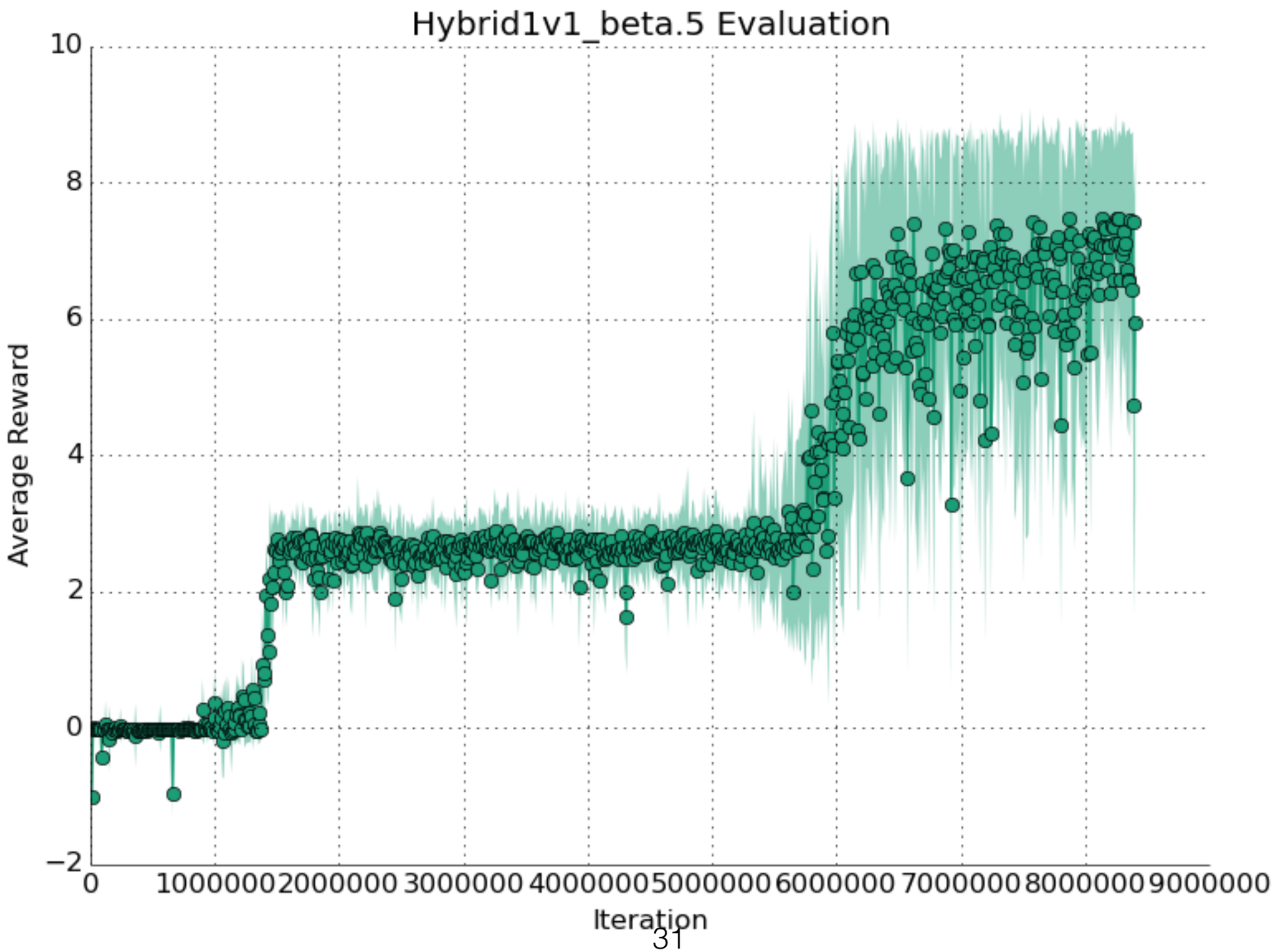
β = 0


Hybrid1v1_beta0 Evaluation
Average Reward
3
2
1
0
−1
−2
−3
−4
−5
0
1000000
2000000
3000000
4000000
5000000
6000000
Iteration

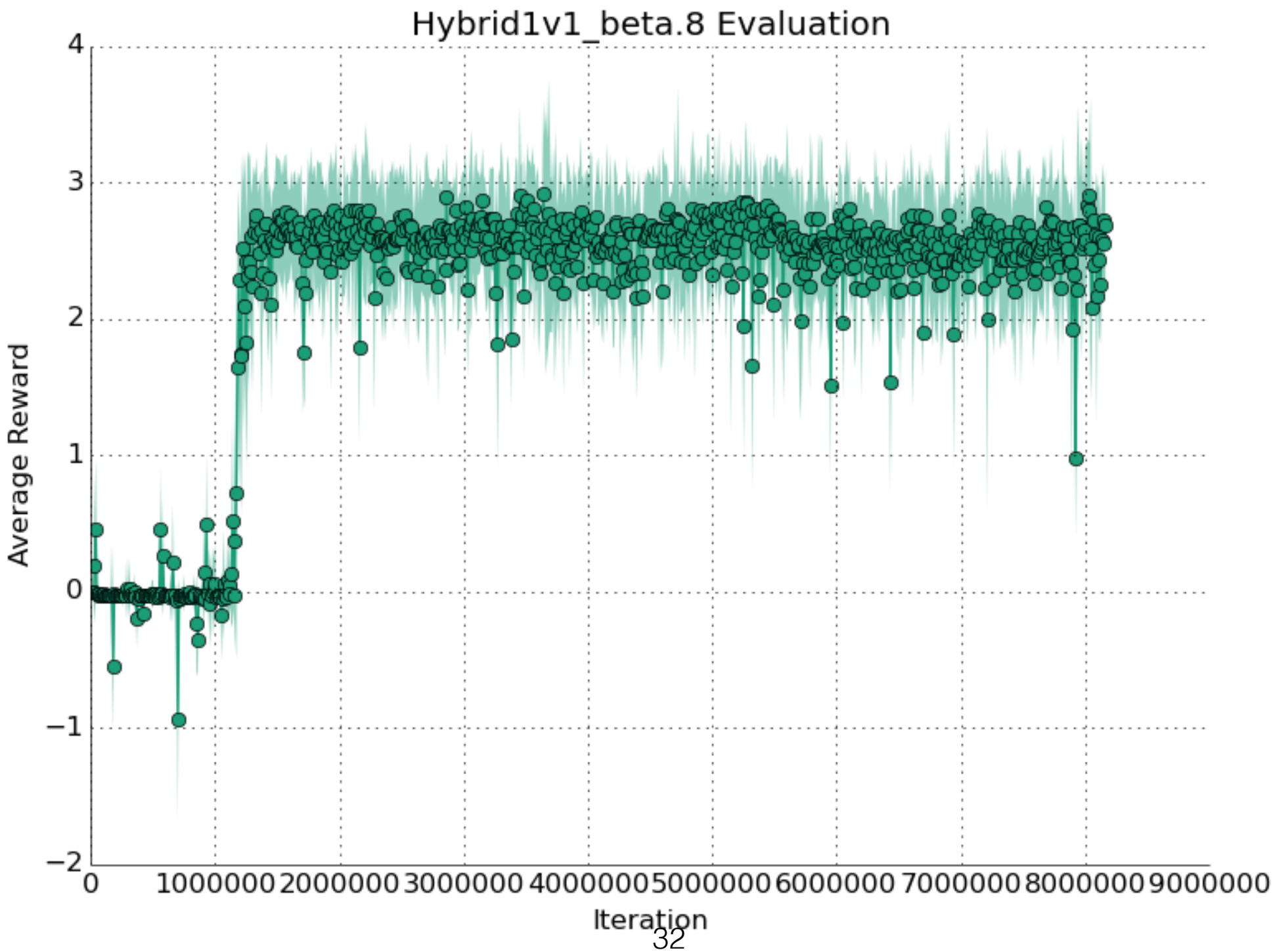
β = 0.2


Hybrid1v1_beta.2 Evaluation
Average Reward
Iteration
10
8
6
4
2
0
−2
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000

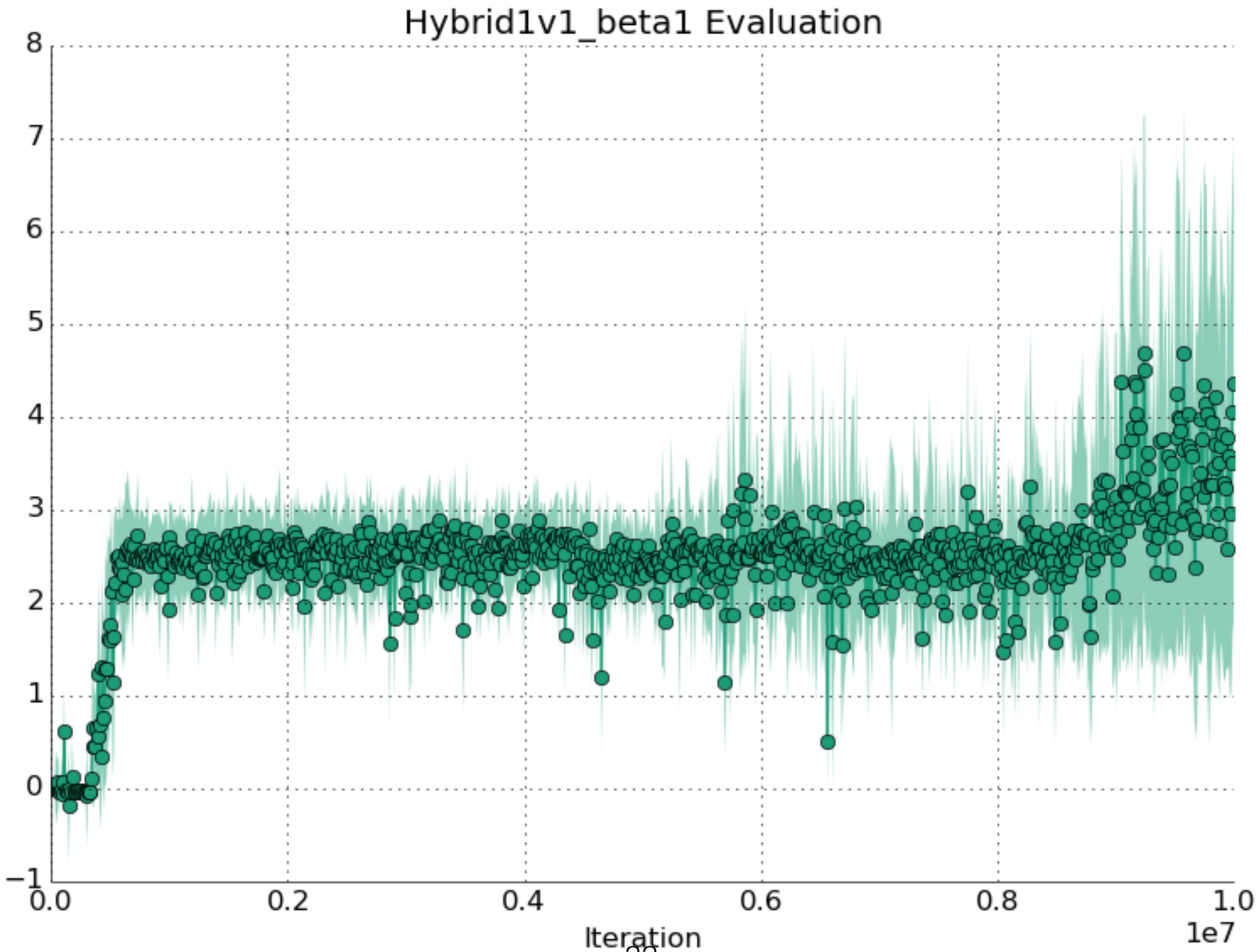
β = 0.5


Hybrid1v1_beta.5 Evaluation
Average Reward
Iteration
10
8
6
4
2
0
−2
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000

β = 0.8


Hybrid1v1_beta.8 Evaluation
Average Reward
Iteration
4
3
2
1
0
−1
−2
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000

β = 1.0


Hybrid1v1_beta1 Evaluation
Average Reward
Iteration
1e7
8
7
6
5
4
3
2
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0


34
1
8000
11
8000

Off-Policy Monte Carlo

For 1v1 experiments, a middle ground between on-policy and off-policy updates works best.

Purely off-policy updates can't learn; Purely on-policy updates take far too long to learn.

[*On-Policy vs. Off-Policy Updates for Deep Reinforcement Learning*, Hausknecht and Stone, DeepRL '16]

Thesis Question

How can the power of Deep Neural Networks be leveraged to extend Reinforcement Learning towards domains featuring partial observability, continuous parameterized action spaces, and sparse rewards?

Novel extension of DDPG to parameterized-continuos action space.

Method for efficiently mixing on-policy and off-policy update targets.

Outline

1. Background
2. Deep Reinforcement Learning
3. Multiagent Architectures
4. Communication

Deep Multiagent RL

Can multiple Deep RL agents cooperate to achieve a shared goal?

Examine several architectures:

Centralized: Single controller for multiple agents

Parameter Sharing: Layers shared between agents

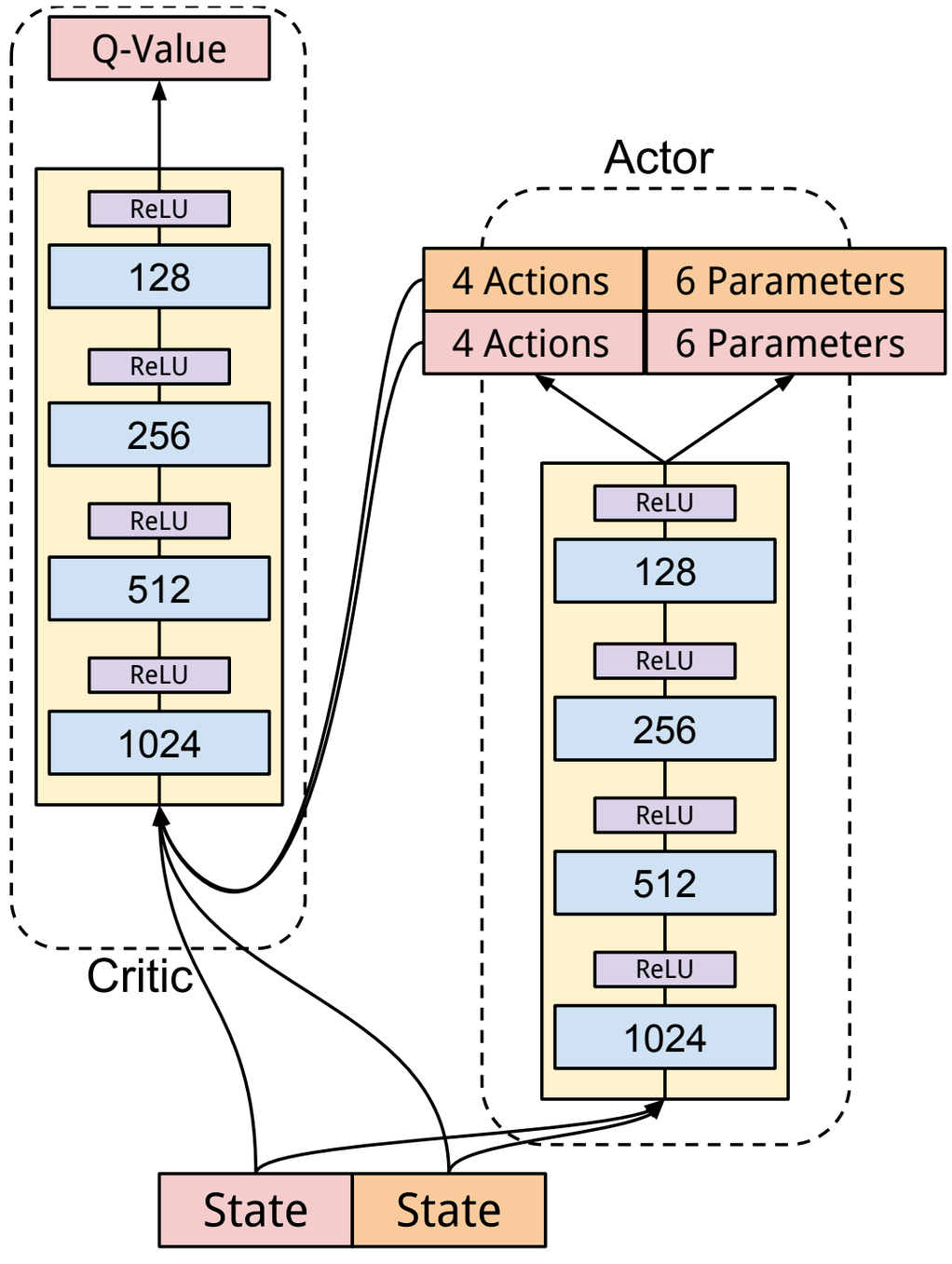
Memory Sharing: Shared replay memory

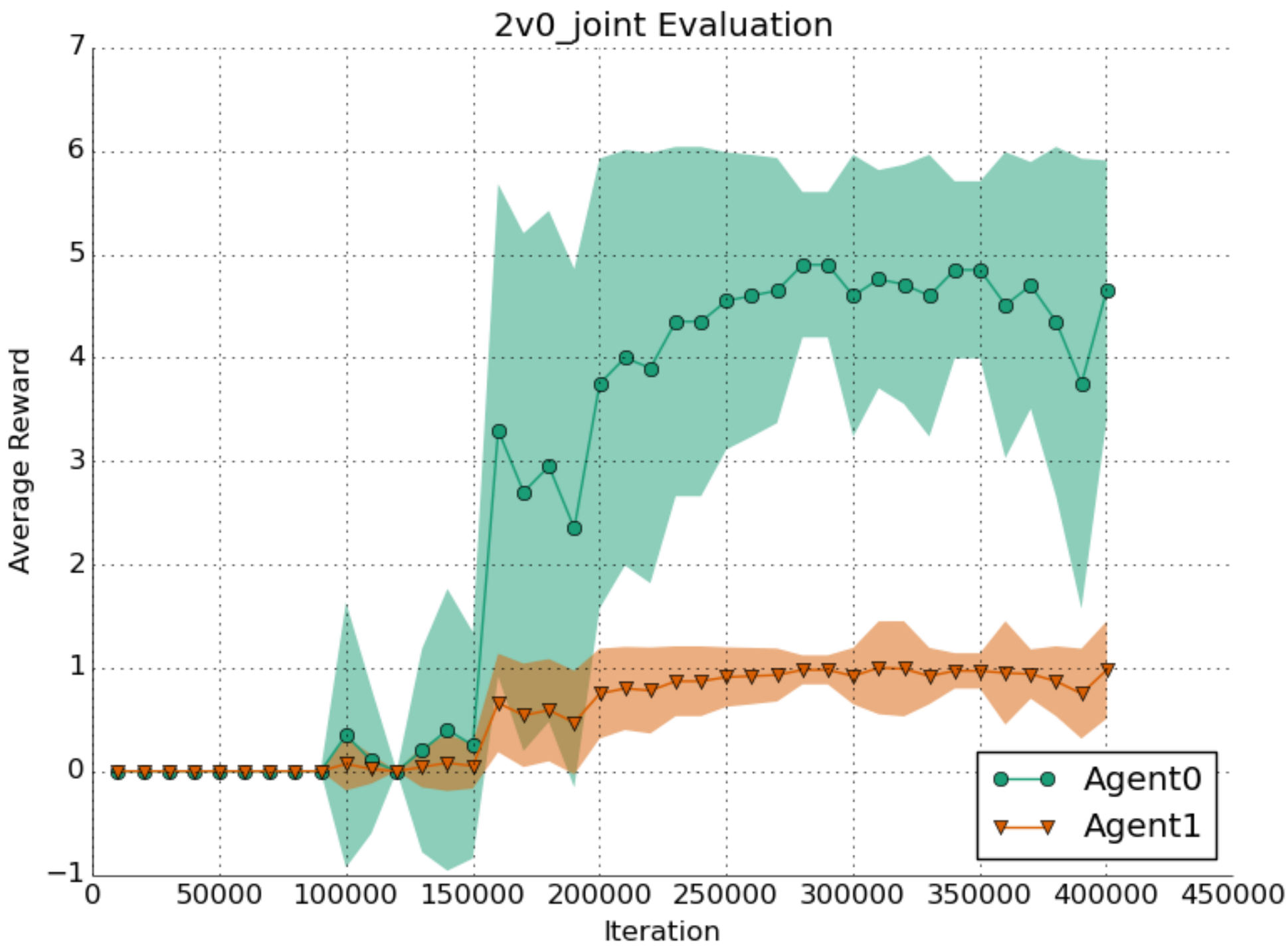
Centralized

Both agents are controlled by a single actor-critic

State & Action spaces are concatenated

Learning takes place in higher-dimensional joint state, action space




2v0_joint Evaluation
Average Reward
Iteration
7
6
5
4
3
2
1
0
−1
0
50000
100000
150000
200000
250000
300000
350000
400000
450000
Agent0
Agent1

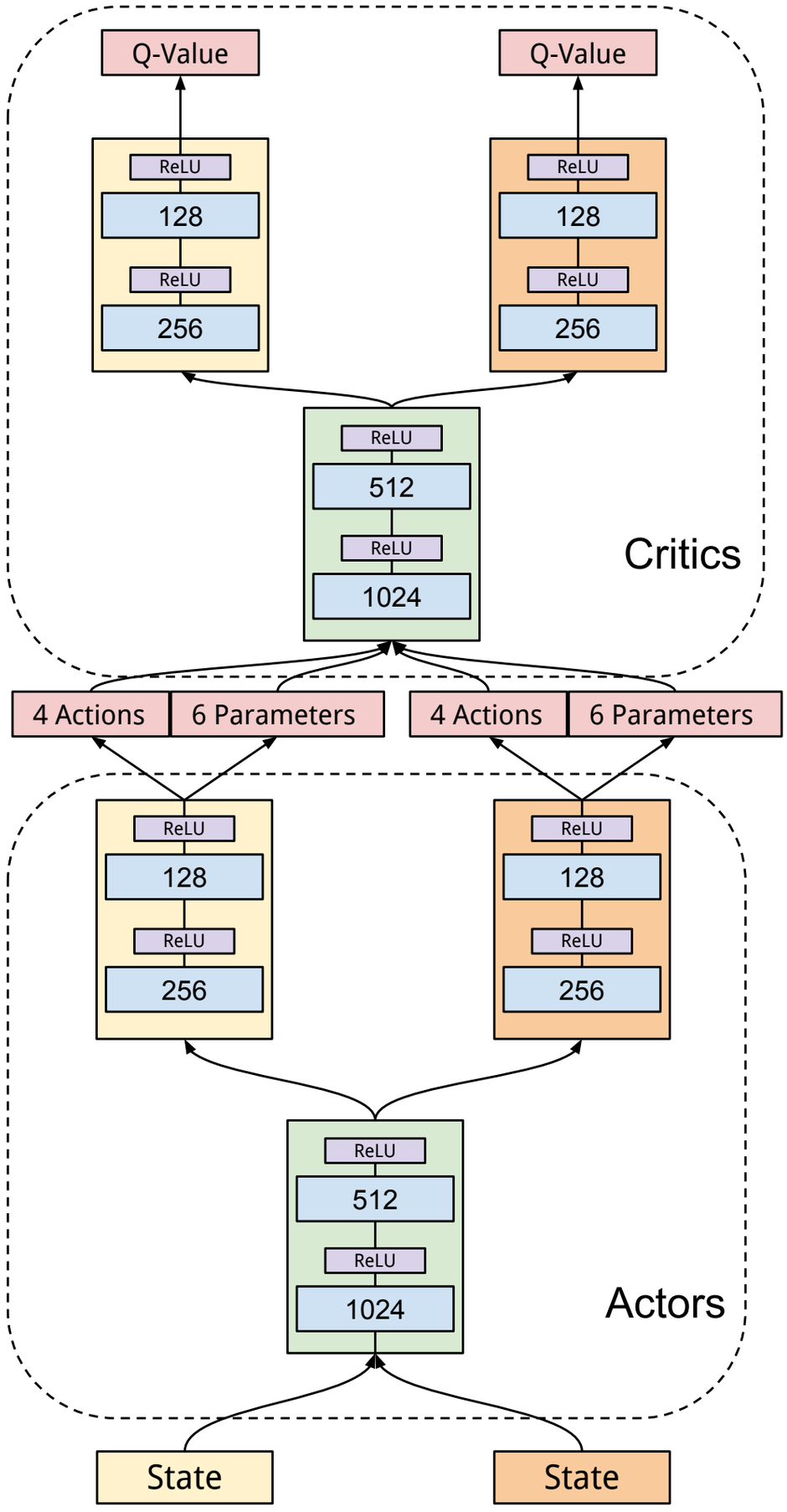

41
11
8000
7
8000

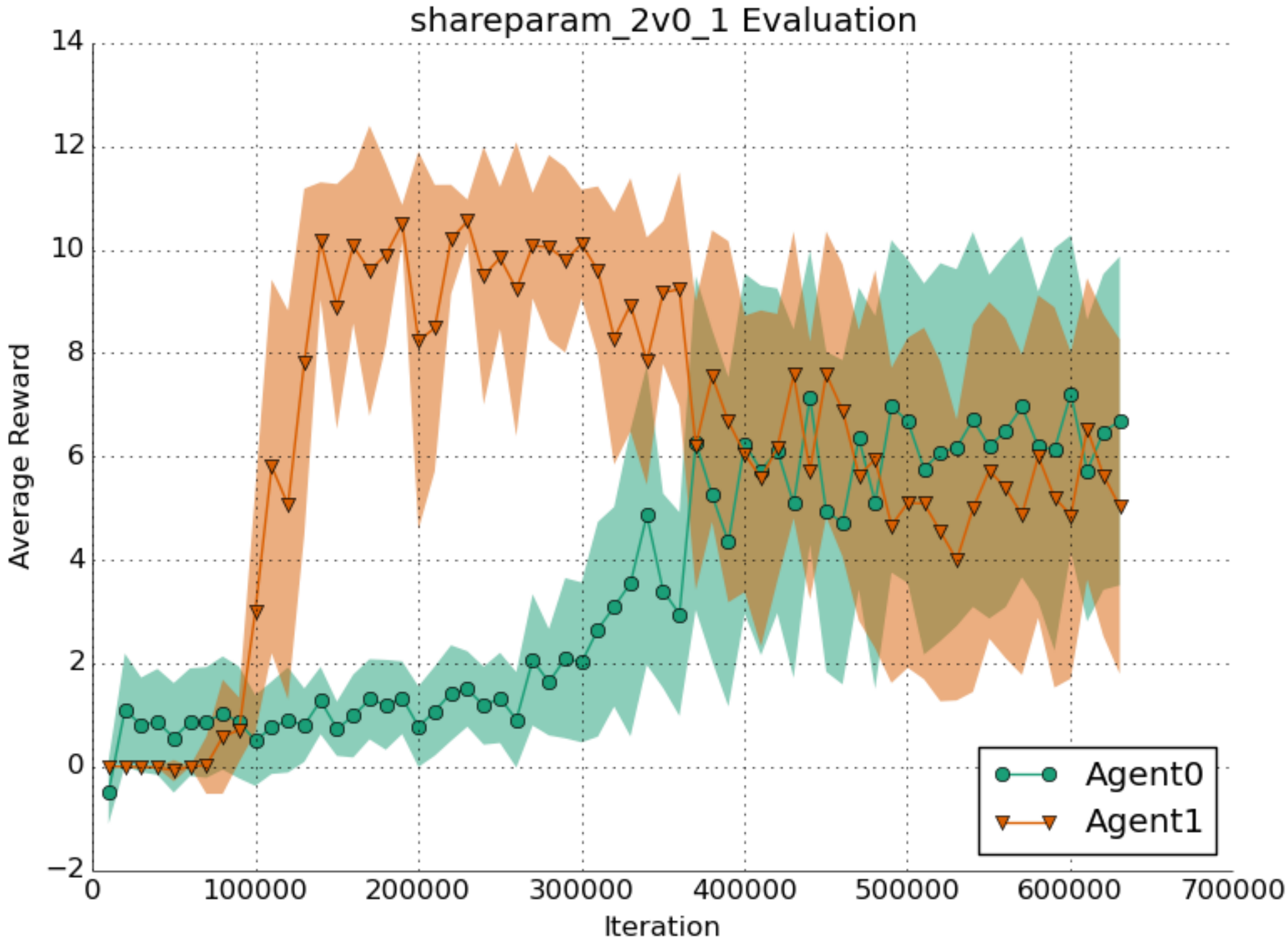
Parameter Sharing

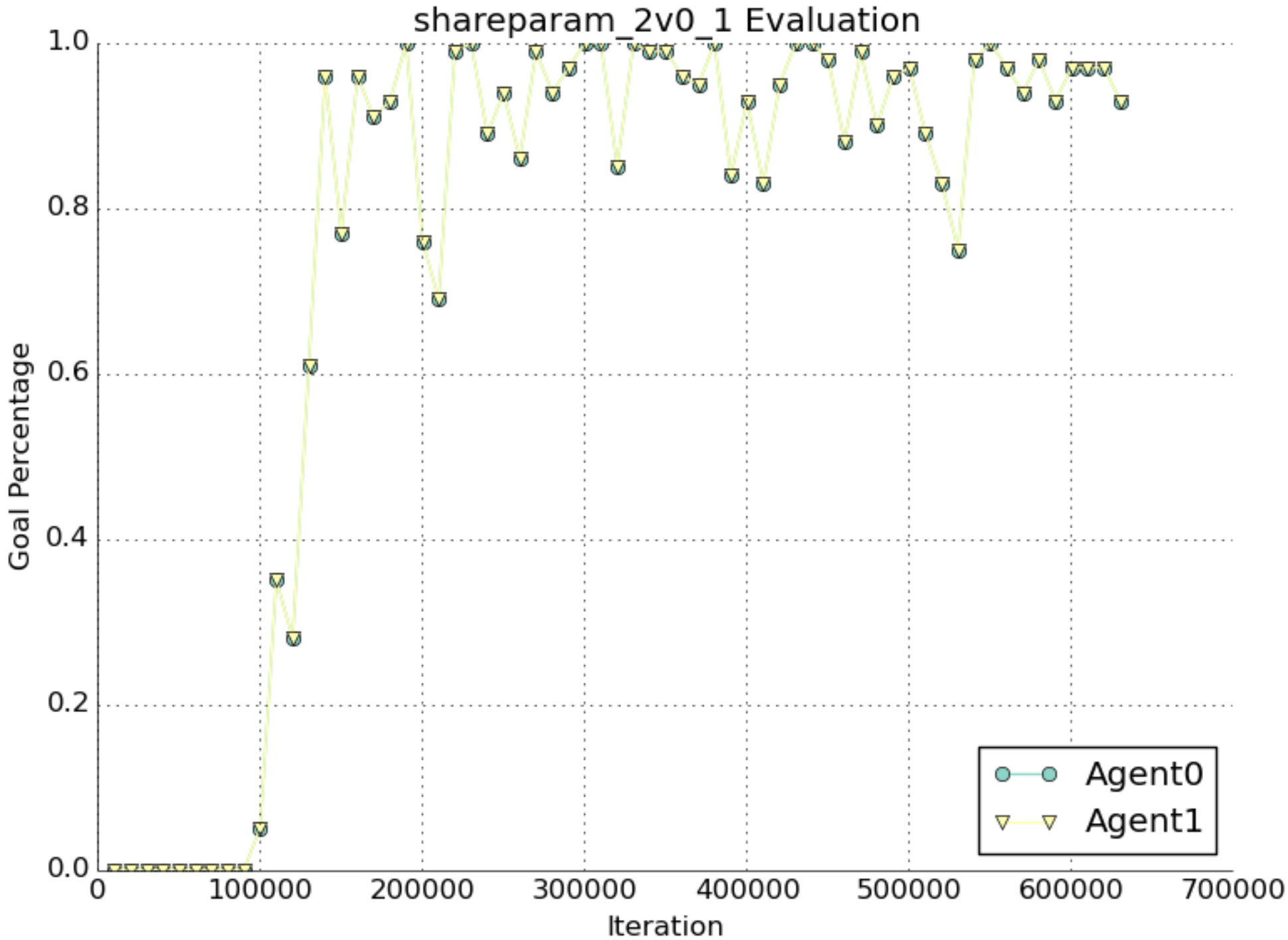
Shared weights between layers in Actor networks. Separate sharing between Critic networks.

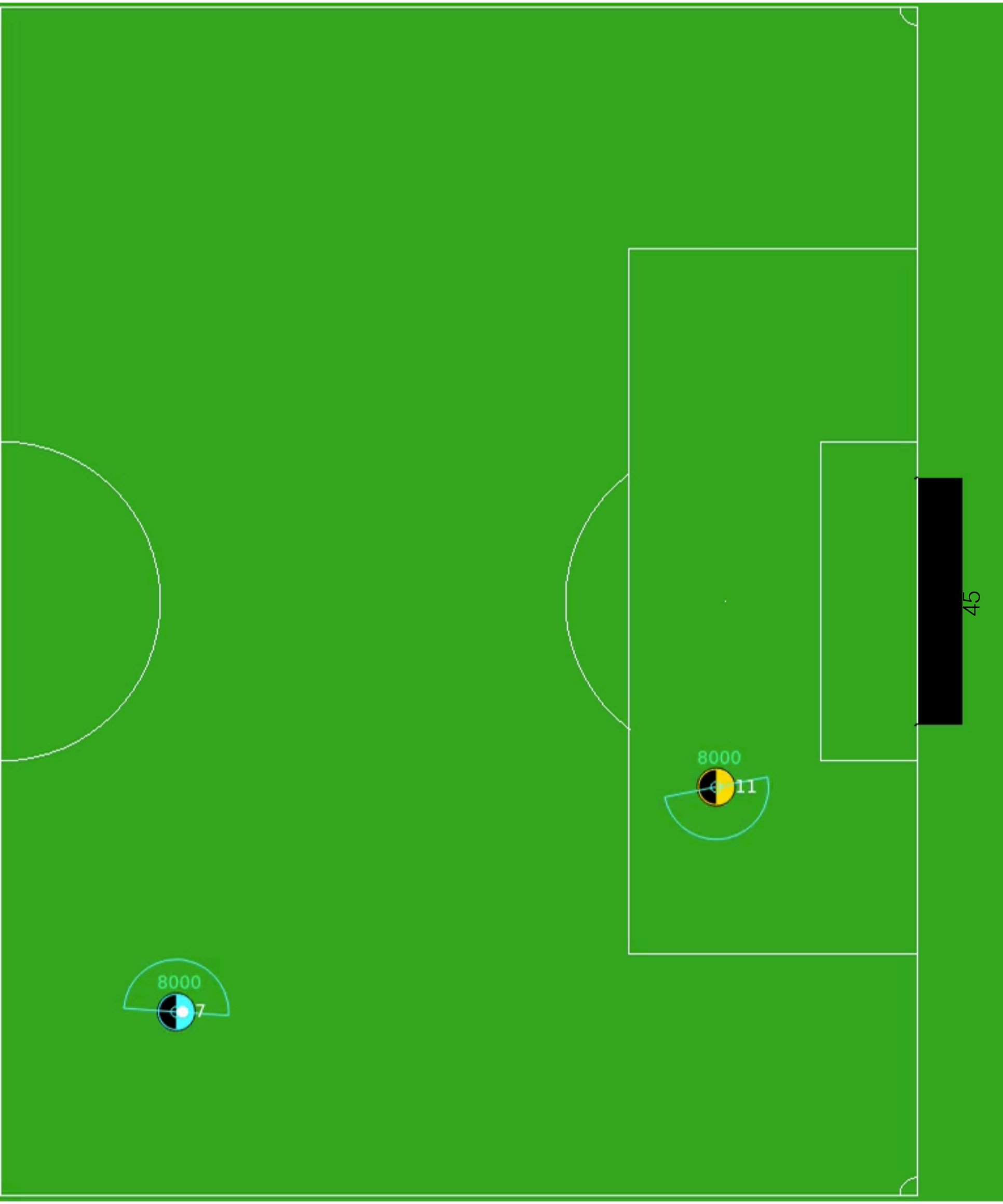
Reduces total number of parameters

Encourages both agents to participate even though 2v0 is solvable by a single agent.

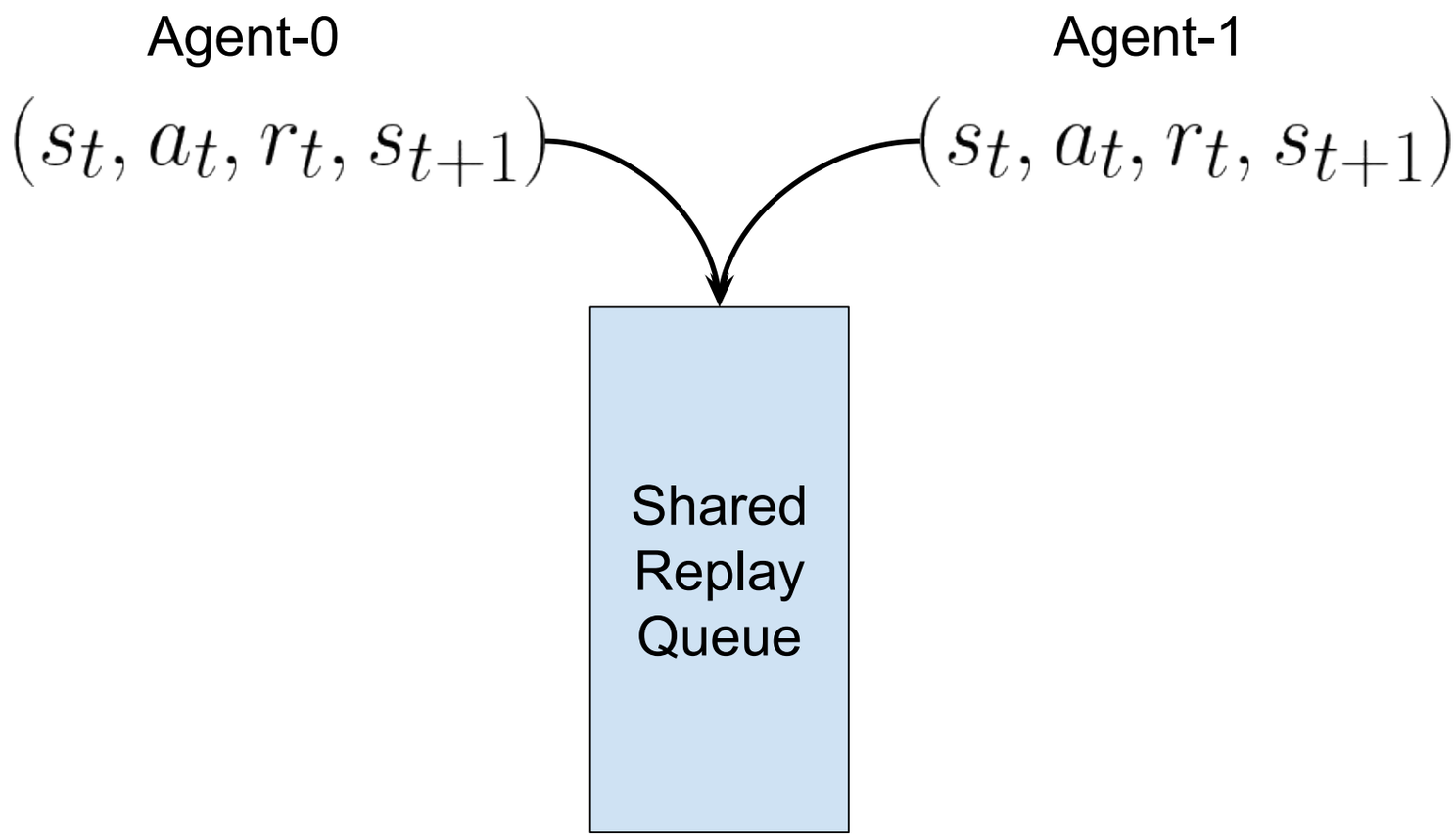



shareparam_2v0_1 Evaluation
Average Reward
Iteration
14
12
10
8
6
4
2
0
−2
0
100000
200000
300000
400000
500000
600000
700000
Agent0
Agent1


shareparam_2v0_1 Evaluation
Goal Percentage
1.0
0.8
0.6
0.4
0.2
0.0
0
100000
200000
300000
400000
500000
600000
700000
Iteration
Agent0
Agent1

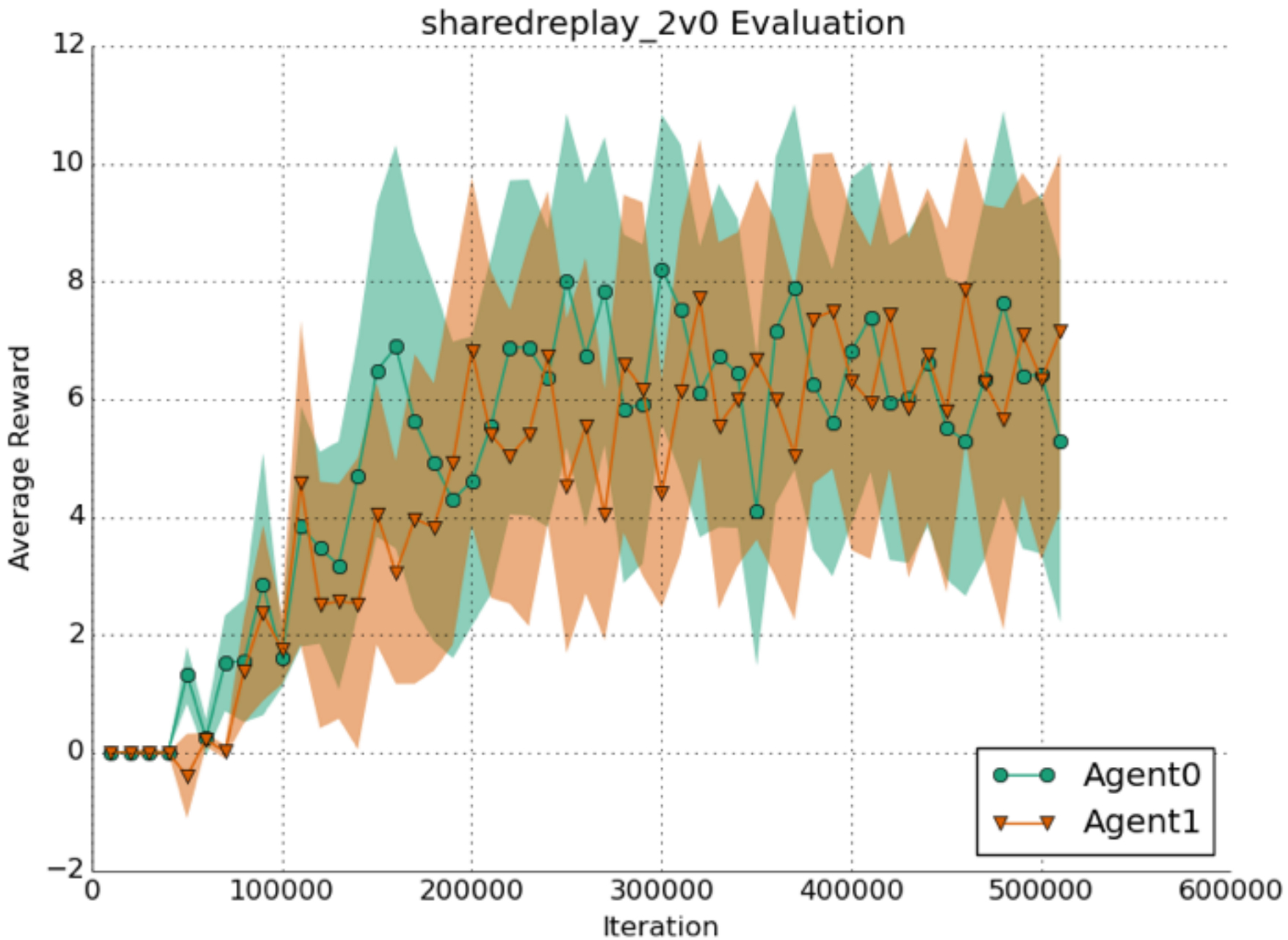

45
8000
11
8000
7

Memory Sharing



Both agents add experiences to a shared memory.
Both agents perform updates from the shared memory.
Parameters of agents are not shared.

Memory Sharing




48
8000
11
000

Multiagent Architectures

Centralized controller utilizes only a single agent.

Sharing parameters and memories encourages policy similarity, which can help all agents learn the task.

Memory sharing results least performance gap between agents.

Outline

1. Background
2. Deep Reinforcement Learning
3. Multiagent Architectures
4. Communication

Symbiosis in Nature

Clownfish and anemone

Crocodile and Egyptian Plover

Communication

In human society, cooperation can be achieved far faster than in nature, through communication.

How can learning agents use communication to achieve cooperation?

Desire a learned communication protocol that can:

1. Identify task-relevant information

2. Communicate meaningful information to the teammate

3. Remain stable enough that teammate can trust the meaning of messages

Related Work

- *Multiagent Cooperation and Competition with Deep Reinforcement Learning*; Tampuu et. al, 2015

- *Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks*; Foerster et al., 2016

- *Learning to Communicate with Deep Multi-Agent Reinforcement Learning*; Foerster et al., 2016

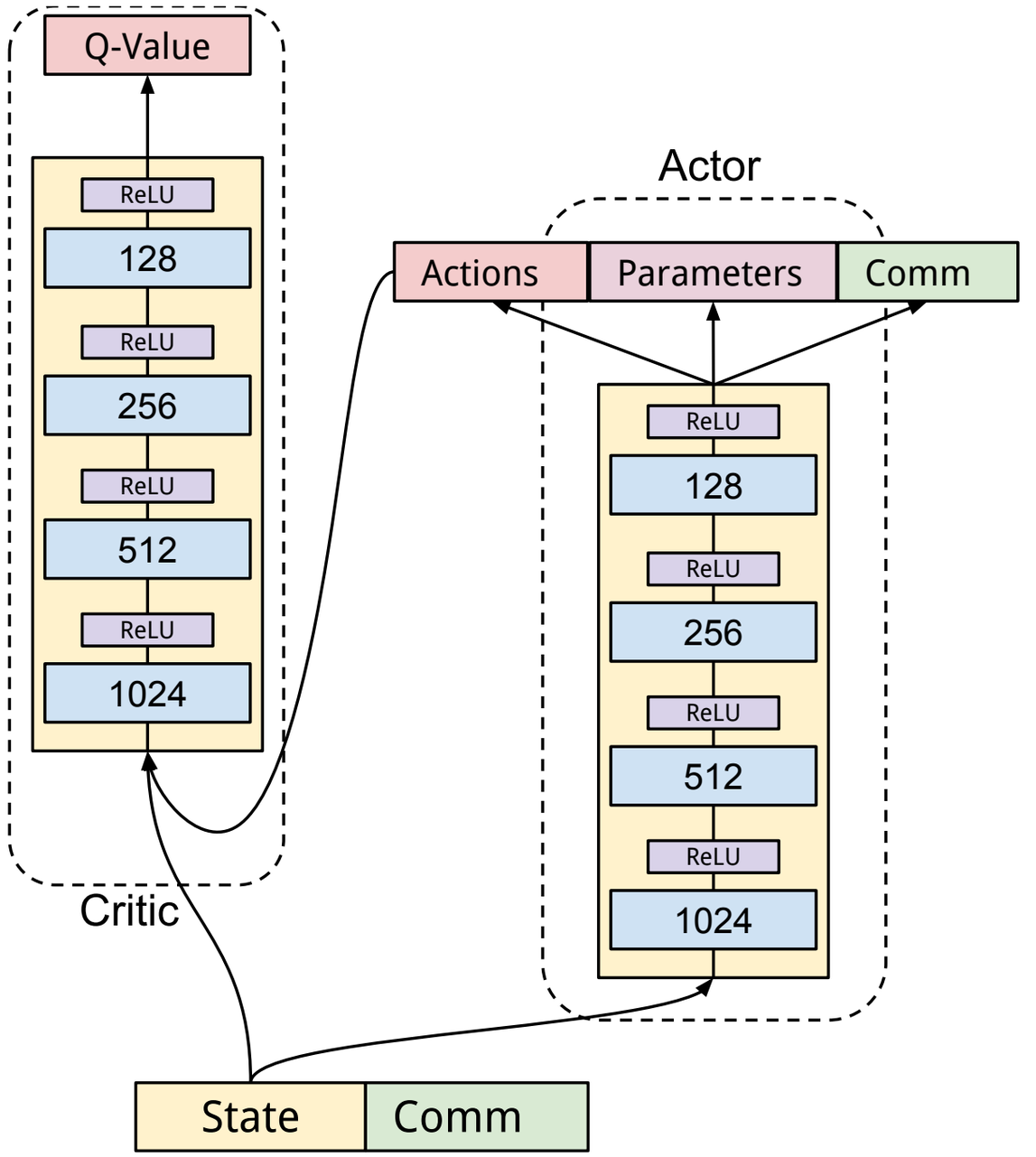
Baseline Communication Architecture

Continuous communication actions. Messages are real values.

No meaning attached to messages; no pre-defined communication protocol.

Incoming messages appended to state.

Messages updated in direction of higher Q-Values.

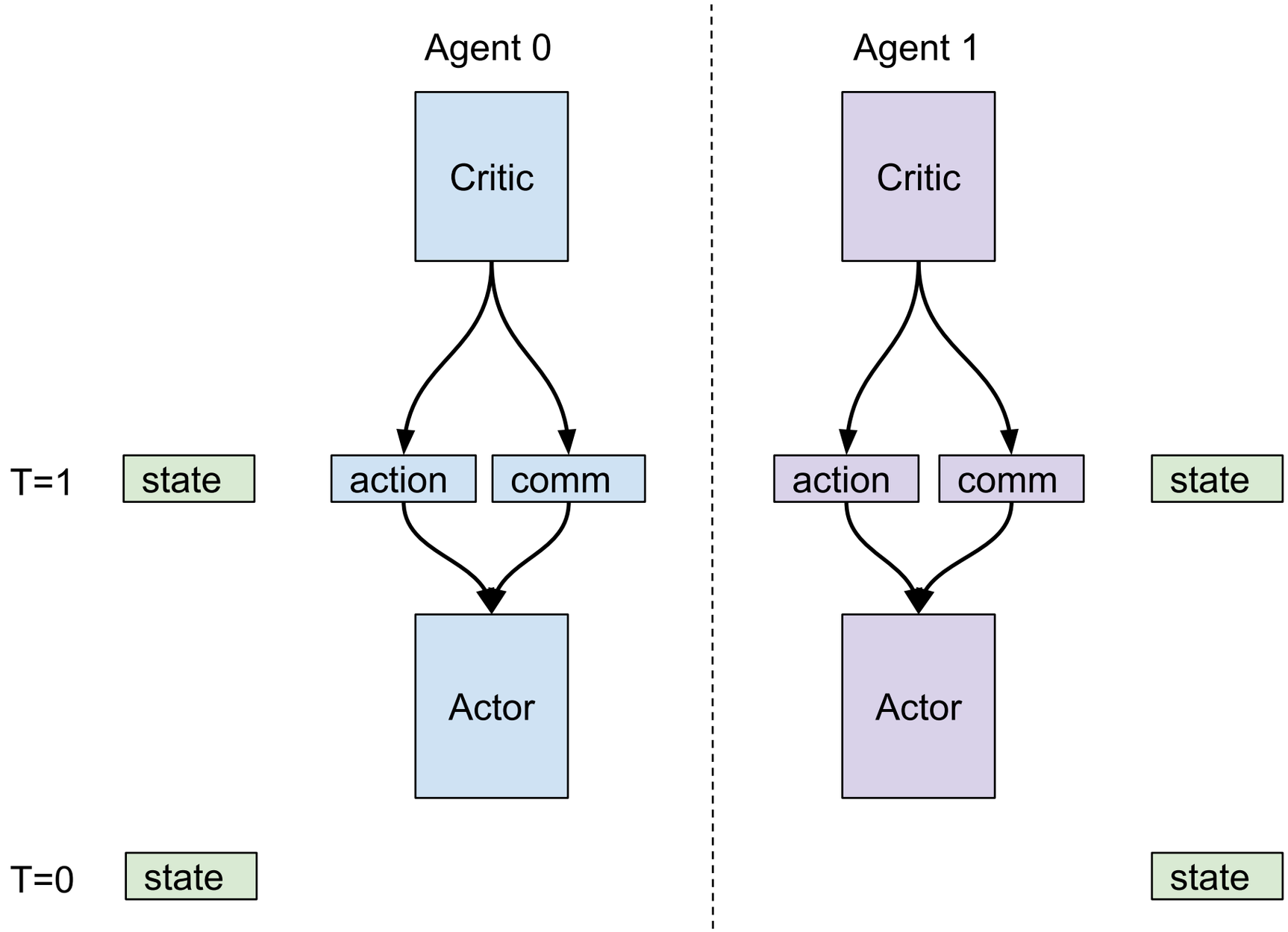


Teammate Comm Gradients

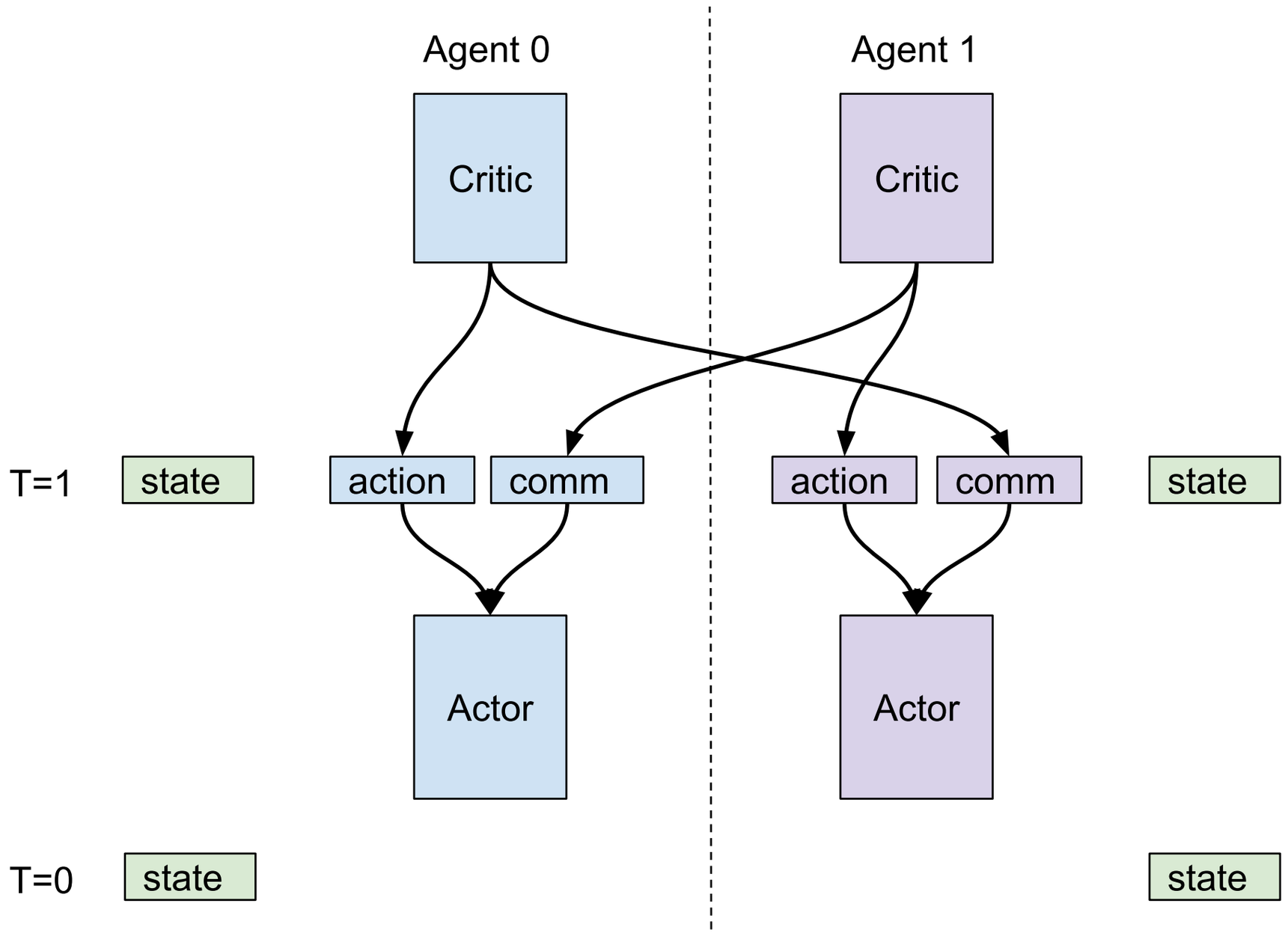
Same as baseline except Communication gradients exchanged with teammate.

Allows teammate to directly alter communicated messages in the direction of higher reward.

Baseline Comm Arch



Teammate Comm Gradients



Guess My Number Task

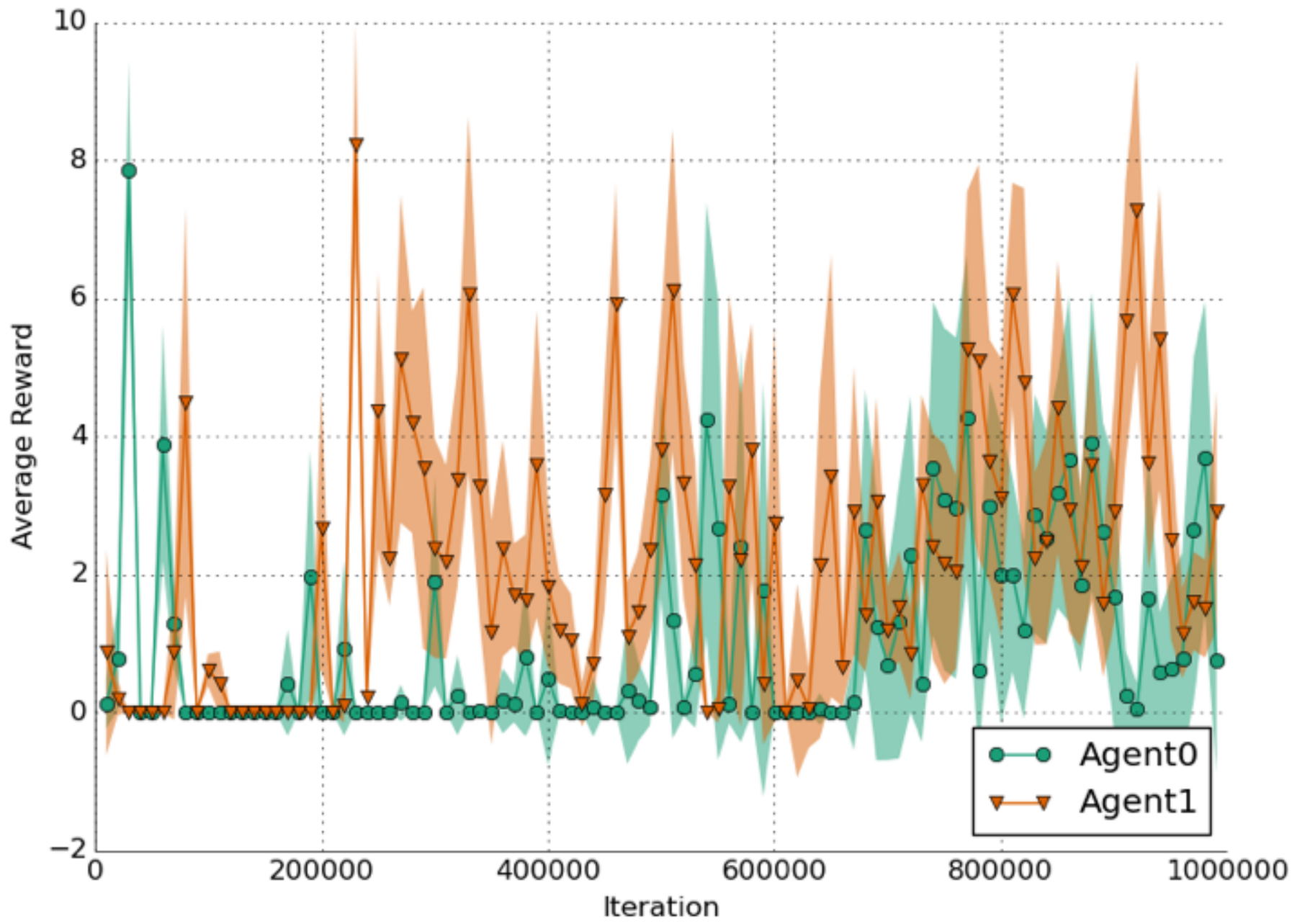
Each agent assigned secret number $h \in (-1, 1)$

Goal: teammate send a message $m \in \mathbb{R}^1$ close to secret number *h*

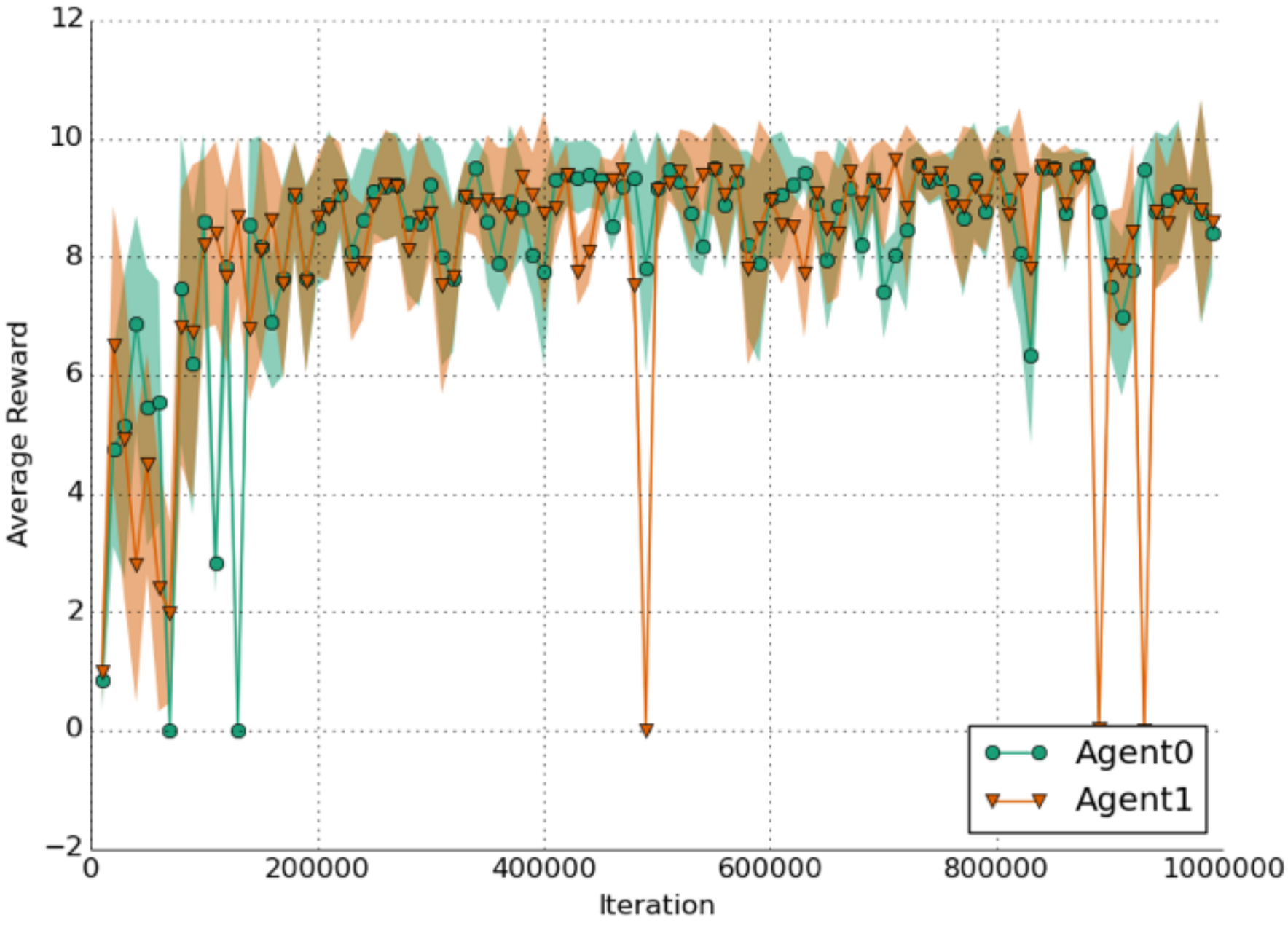
Reward: $r_t = \alpha / e^{\beta(h - m_{t-1})^2}$

Max reward when teammate message equals your secret number.

Baseline


10
8
6
4
2
0
−2
Average Reward
0
200000
400000
600000
800000
1000000
Iteration
Agent0
Agent1

Teammate Comm Grad



Blind Soccer

Blind Agent can hear but cannot see

Sighted Agent can see but cannot move

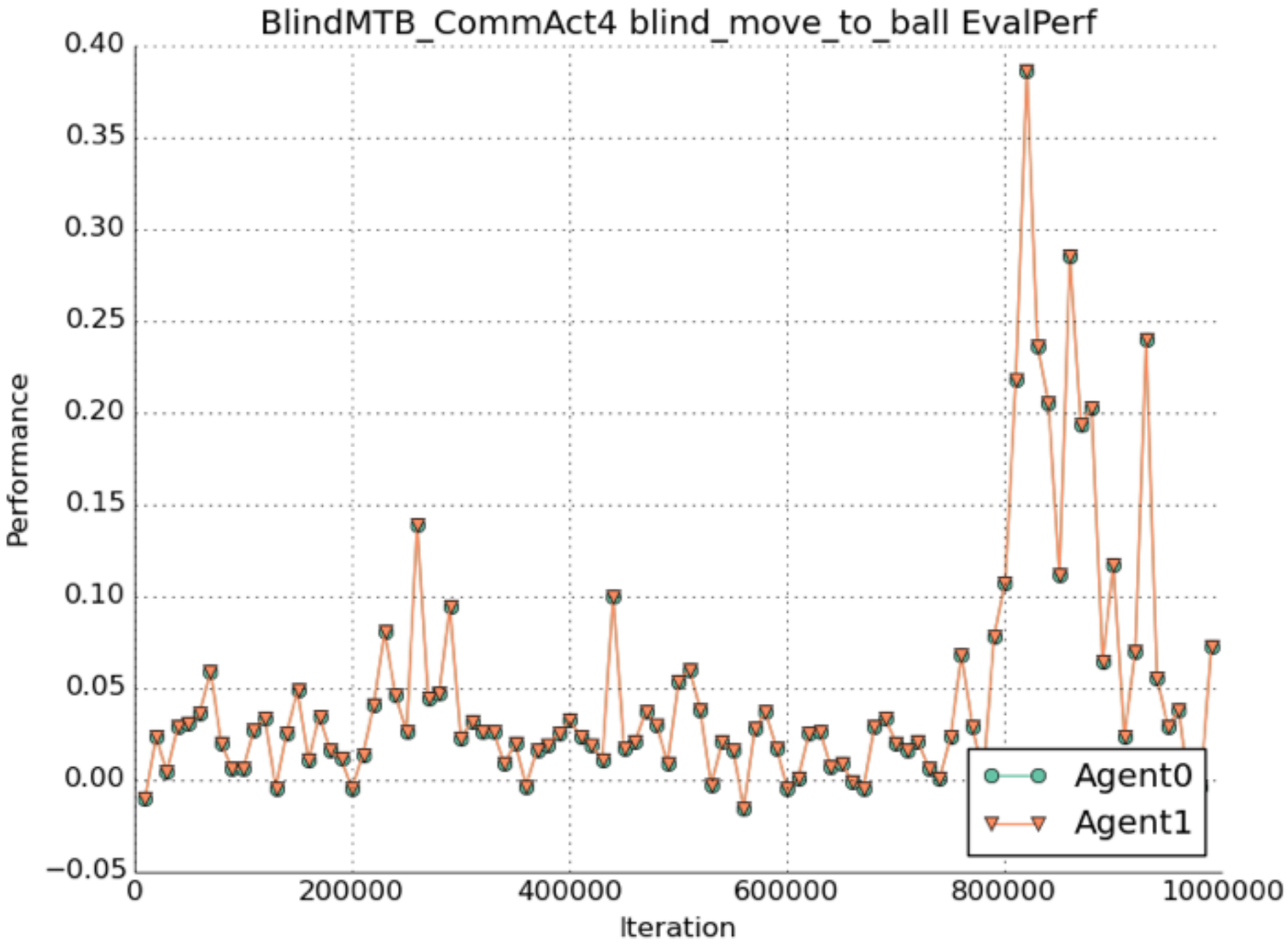
Goal: sighted agent must use communication to help blind agent locate and approach the ball

Rewards: $r_t = -\Delta d(\text{Blind Agent}, \text{Ball})$

Agents communicate using messages $m \in \mathbb{R}^4$


8000
7
8000
11

Baseline



Baseline architecture begins to solve the task, but the protocol is not stable enough and performance crashes.

Teammate Comm Gradients

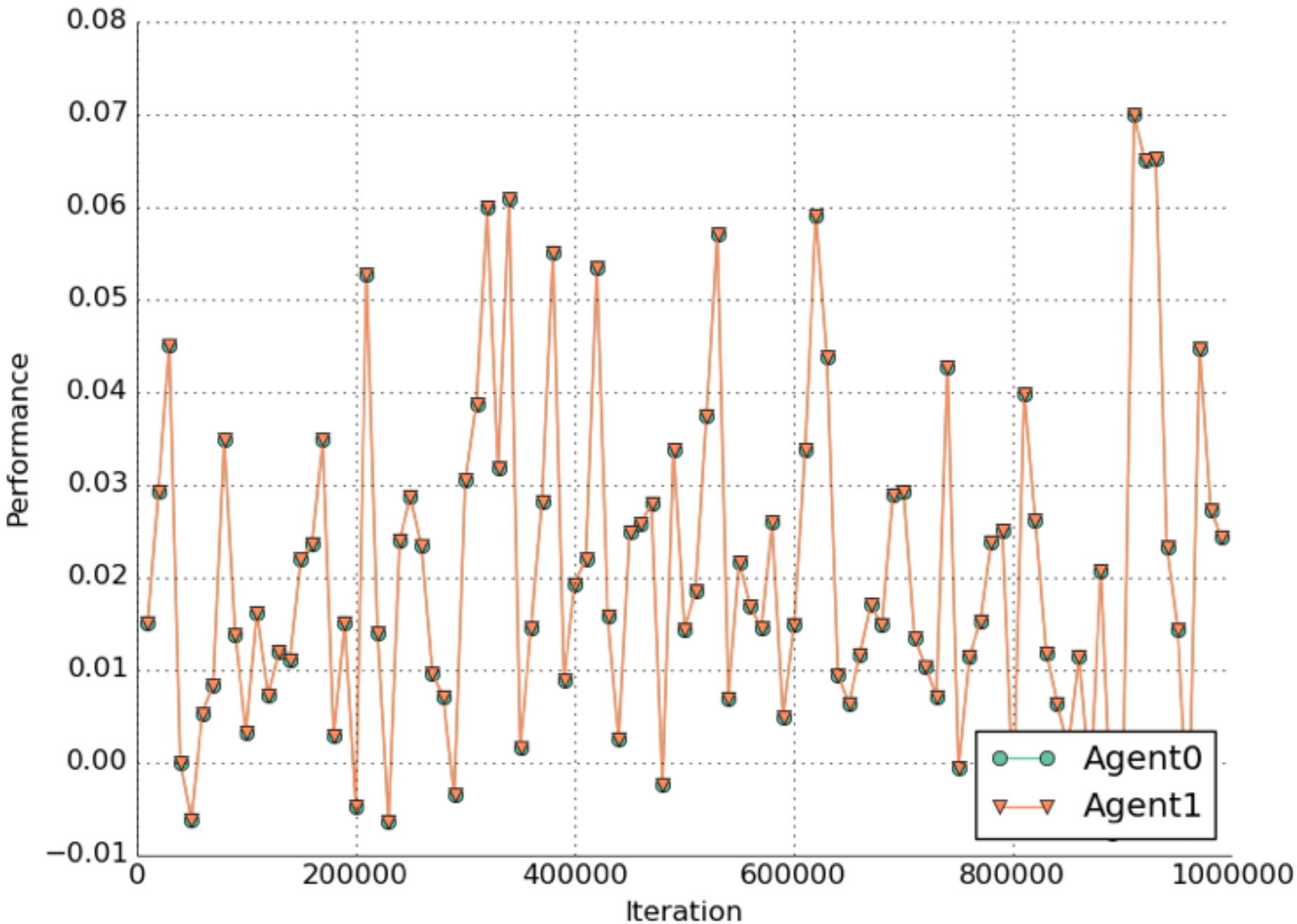
Fails to ground messages in the state of the environment.

Agents fabricate idealized messages that don’t reflect reality.

Example: blind agent wants the ball to be directly ahead

So it alters the sighted agents messages to say this, regardless of the actual location of the ball.

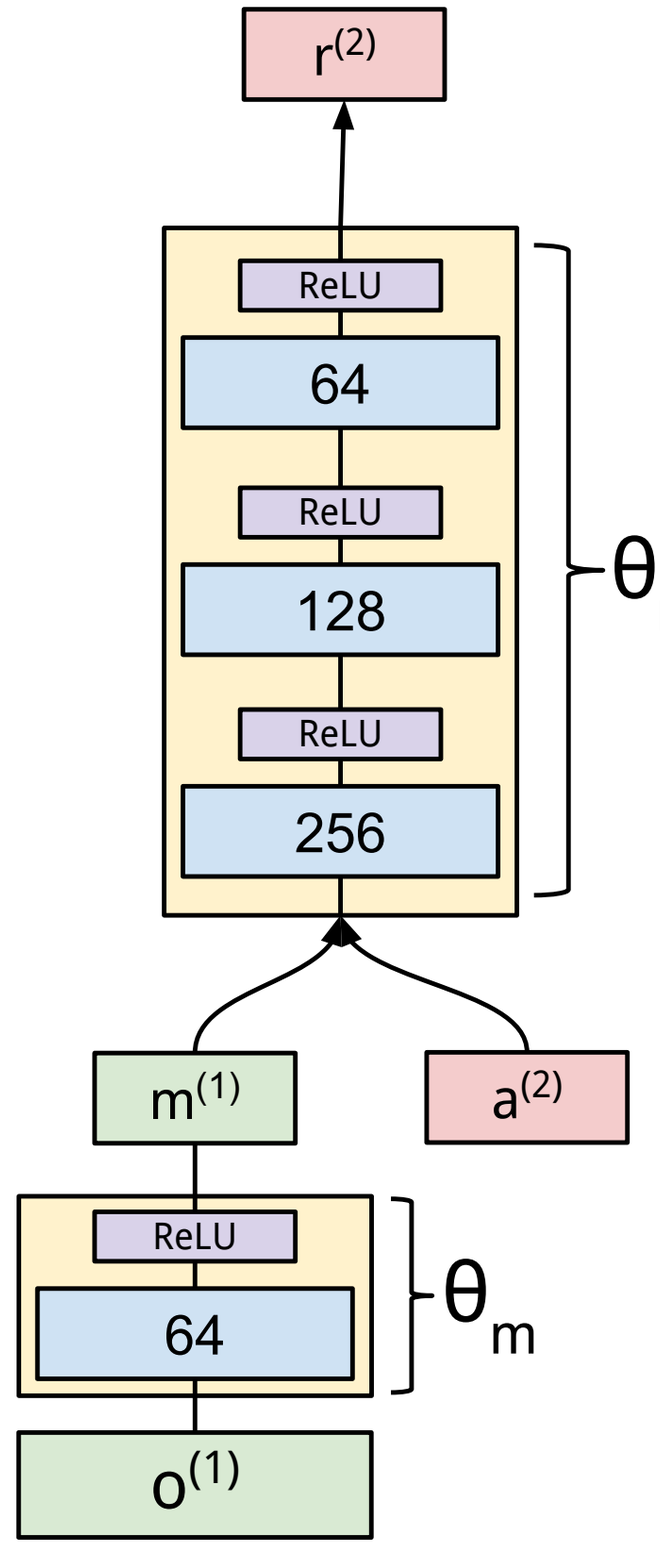
Teammate Comm Gradients



Grounded Semantic Network

GSN learns to extract information from the sighted agent's observations that is useful for predicting the blind agent's rewards.

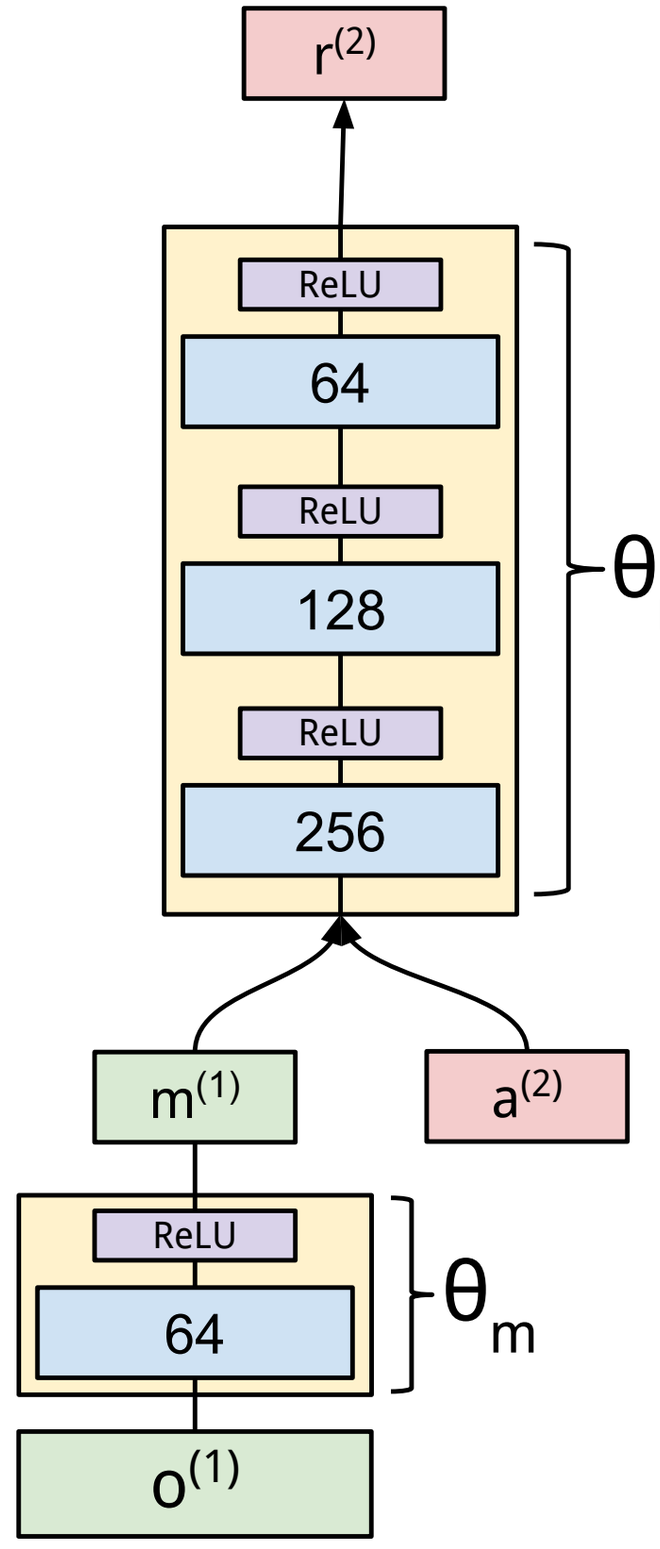
Intuition: We can use observed rewards to guide the learning of a communication protocol.



Grounded Semantic Network

Maps sighted agent's observation $o^{(1)}$ and blind teammate's action $a^{(2)}$ to blind teammate reward $r^{(2)}$

$$r^{(2)} = GSN(o^{(1)}, a^{(2)})$$



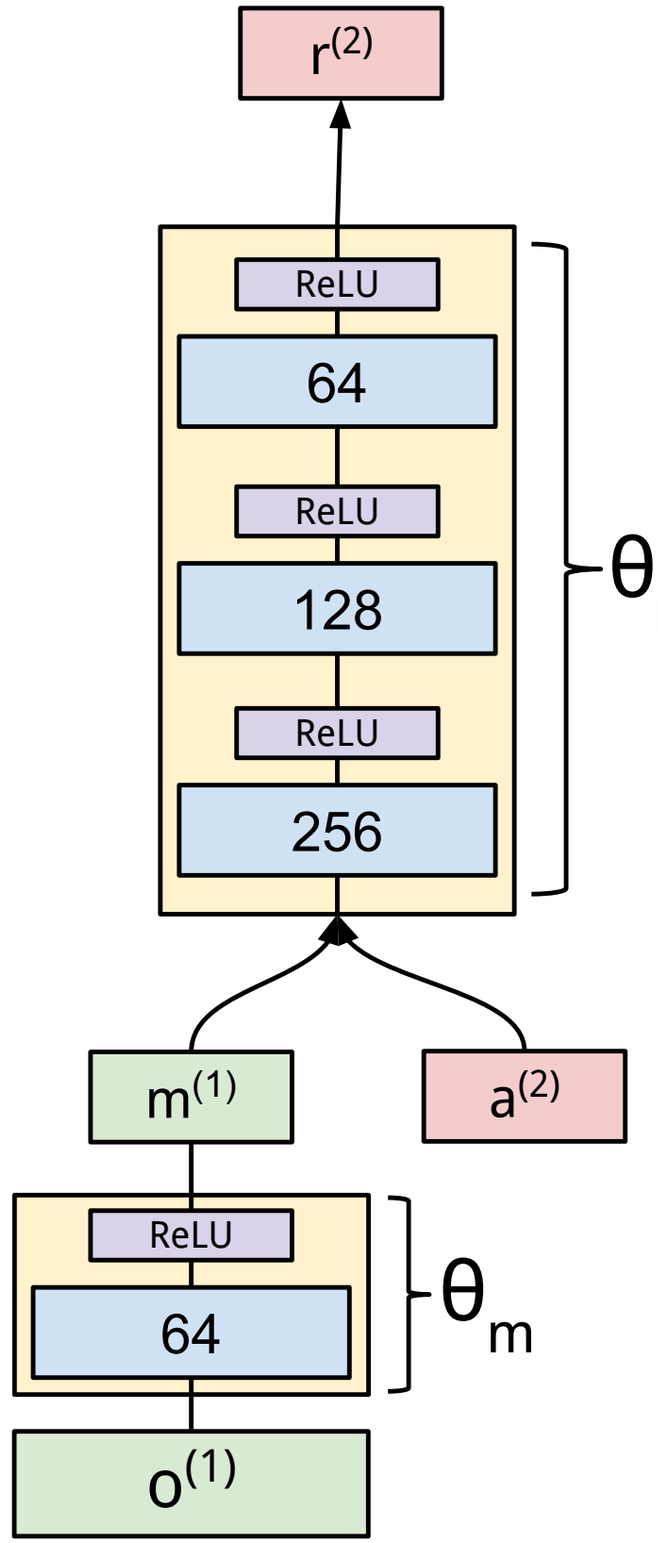
Grounded Semantic Network

Message encoder *M* and a reward model *R*:

$$\hat{r}^{(2)} = R\Big(M\big(o^{(1)};\theta_m\big), a^{(2)};\theta_r\Big)$$

Activations of layer $m^{(1)}$ form the message.

Intuition: $m^{(1)}$ will contain any salient aspects of $o^{(1)}$ that are relevant for predicting reward.



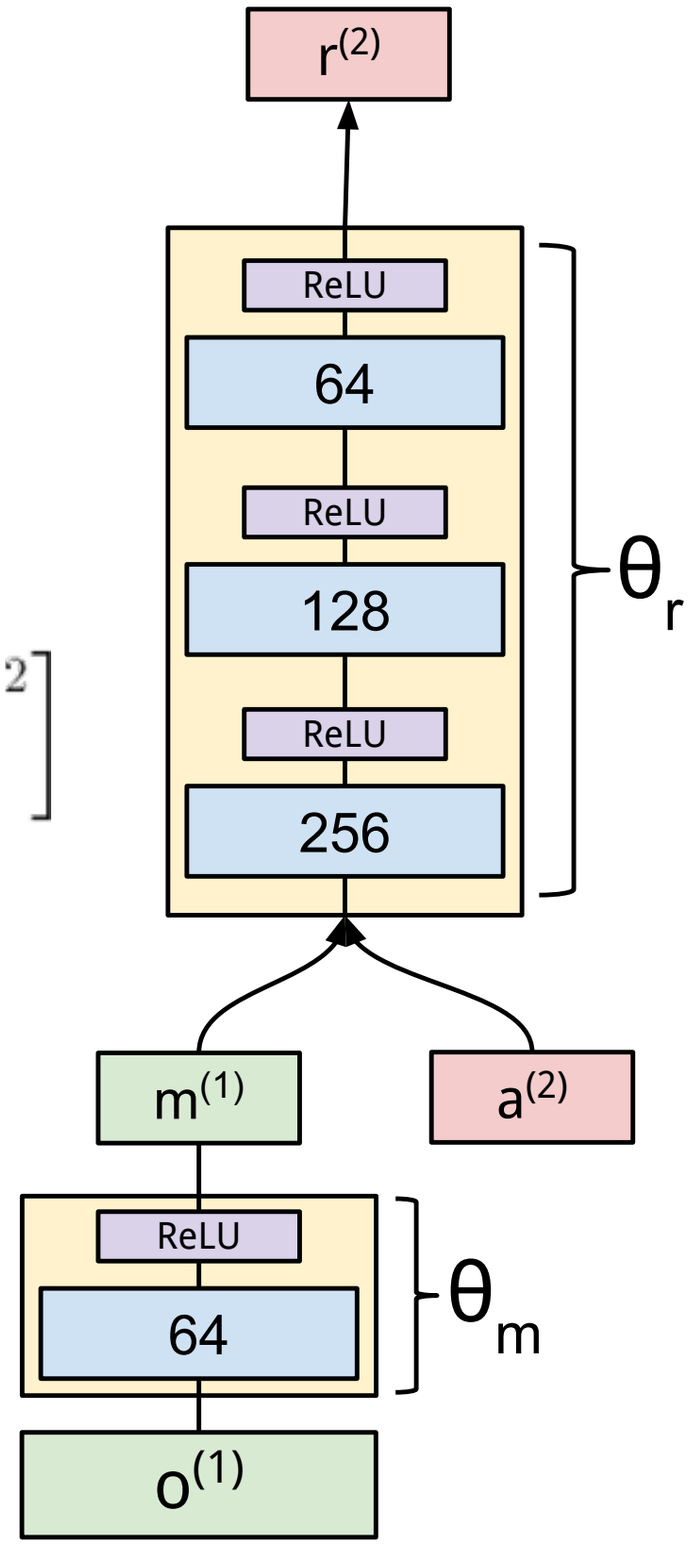
Grounded Semantic Network

Training minimizes supervised loss:

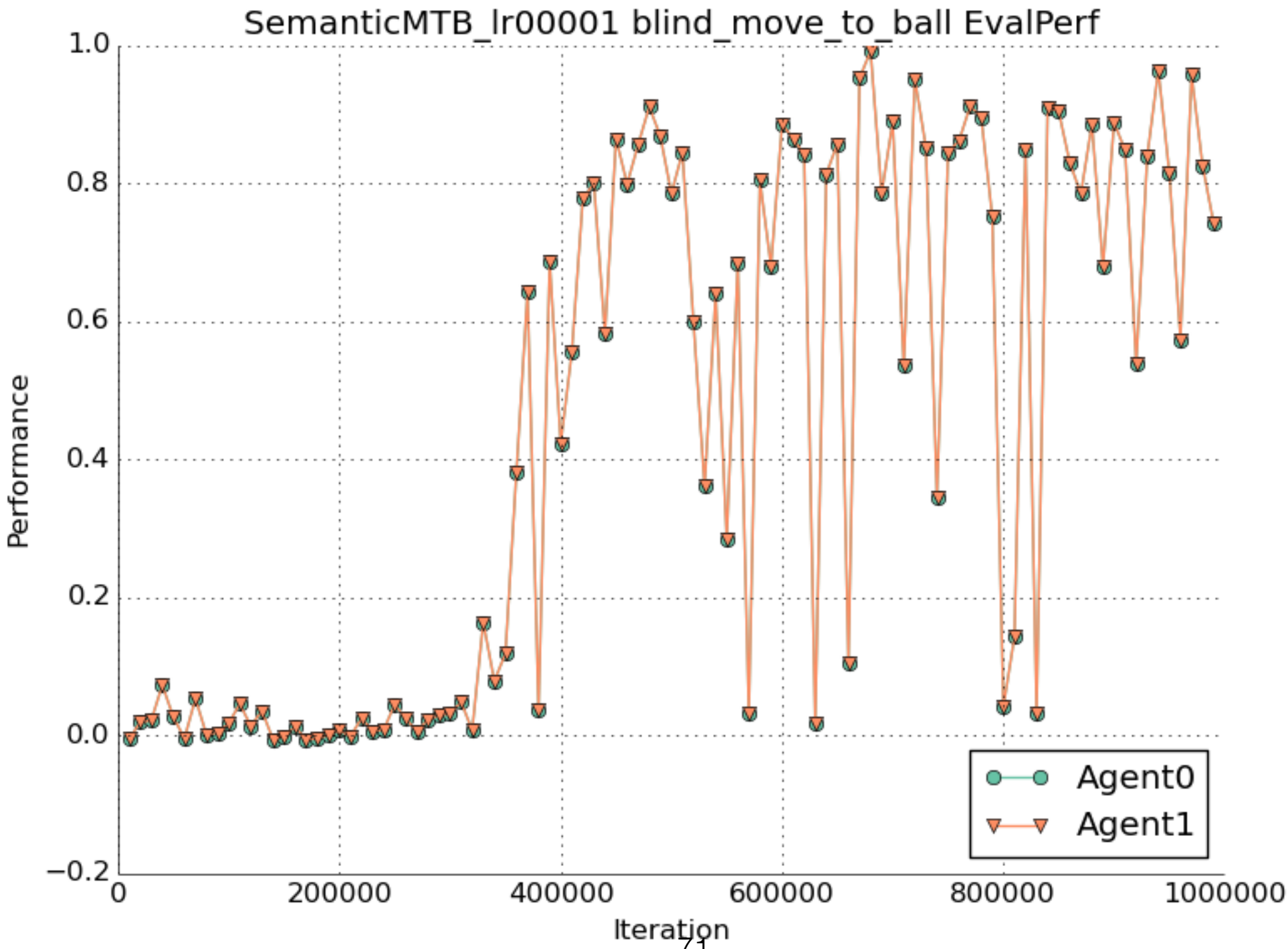
$$L(\theta_r, \theta_m) = \mathbb{E}_{(o^{(1)}, a^{(2)}, r^{(2)})}\left[\left(r^{(2)} - R\left(M(o^{(1)}; \theta_m), a^{(2)}; \theta_r\right)\right)^2\right]$$

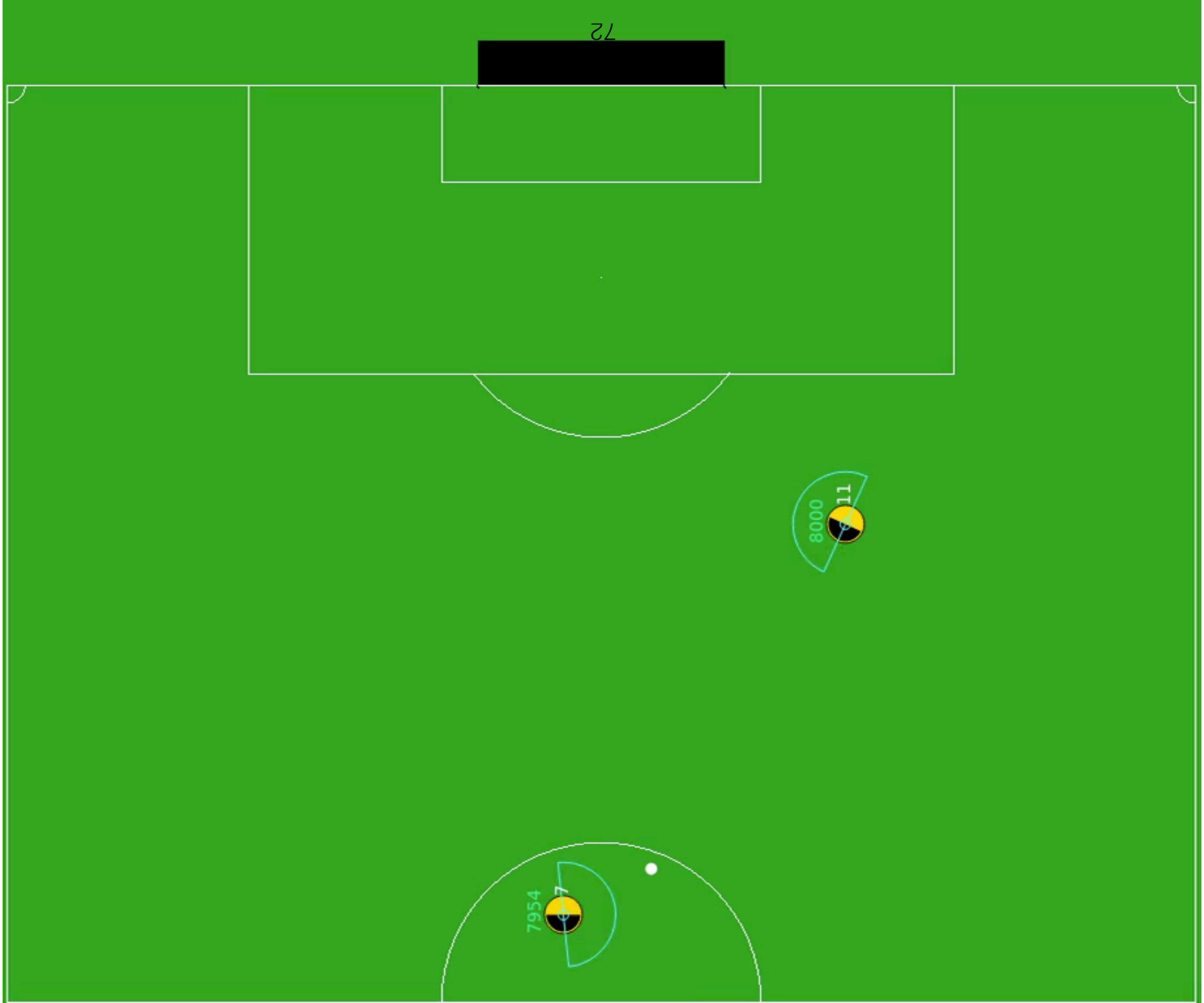
Evaluation requires only observation $o^{(1)}$ to generate message $m^{(1)}$

GSN is trained in parallel with agent. Uses learning rate 10x smaller than agent for stability.

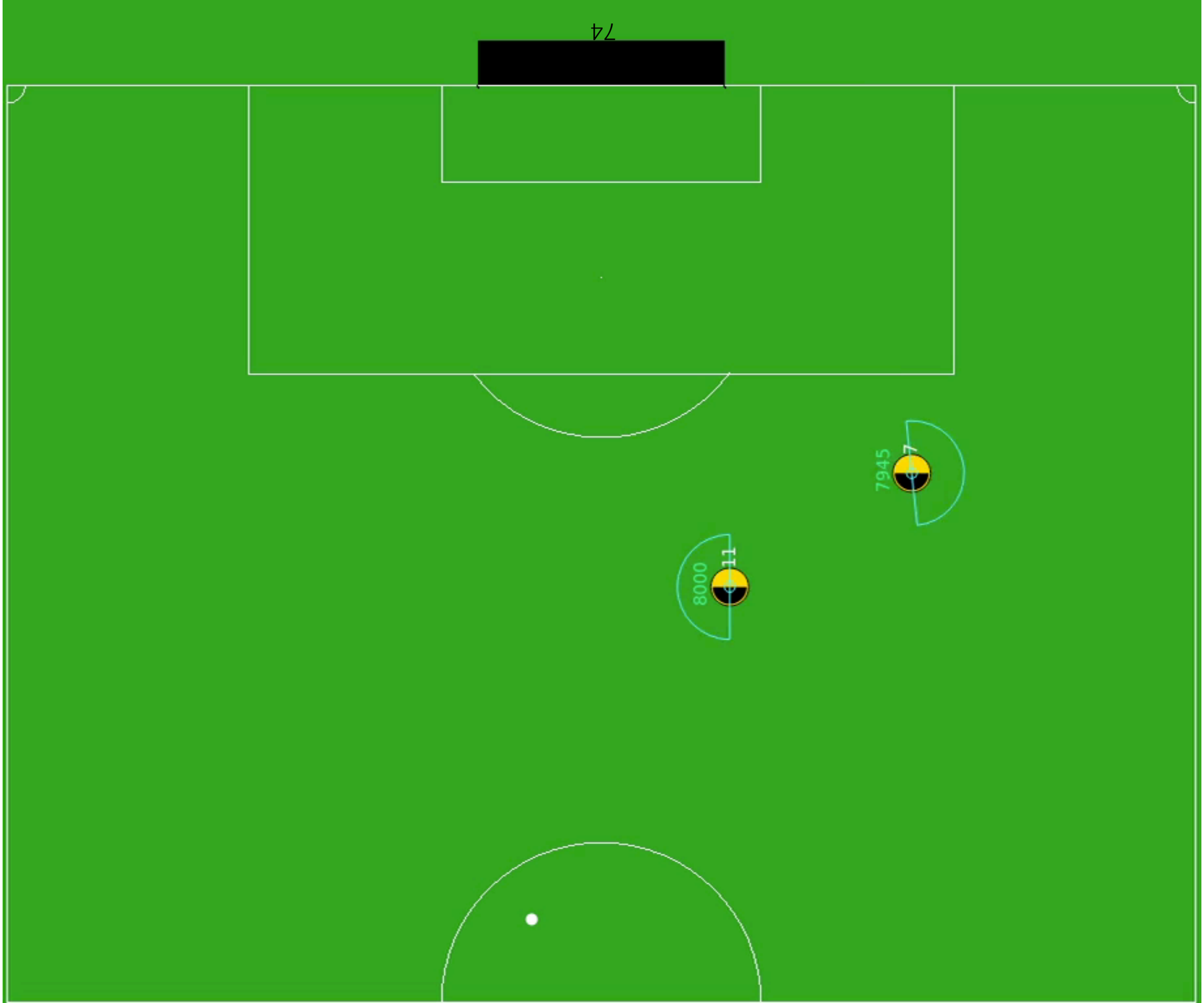


GSN


SemanticMTB_lr00001 blind_move_to_ball EvalPerf
Performance
Iteration
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200000
400000
600000
800000
1000000
Agent0
Agent1


72
8000
11
7954
7

Is communication really helping?


74
7945
11
8008

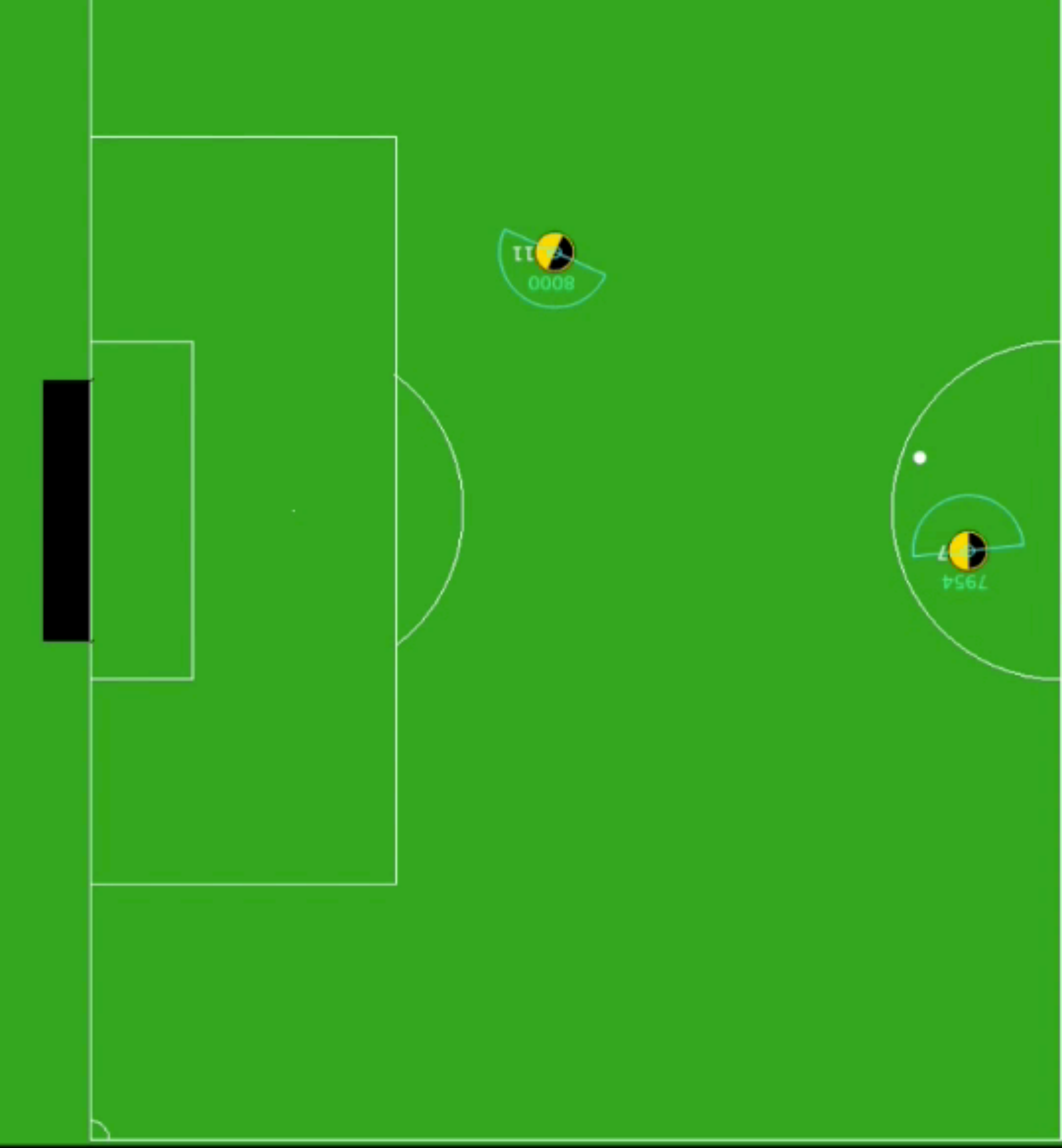
t-SNE Analysis

2D t-SNE projection of 4D messages sent by the sighted agent

Similar messages in 4D space are close in the 2D projection

Each dot is colored according to whether the blind agent Dashed or Turned

Content of messages strongly influences actions of blind agent


11
8000
7954

Desire a learned communication protocol that can:

1. Identify task-relevant information

2. Communicate meaningful information to the teammate

3. Remain stable enough that teammate can trust the meaning of messages

GSN fulfills these criteria. For more info see: [*Grounded Semantic Networks for Learning Shared Communication Protocols*] NIPS DeepRL Workshop ‘16

Communication Conclusions

Communication can help cooperation. It is possible to learn stable and informative communication protocols.

Teammate Communication Gradients is best in domains where reward is tied directly to the content of the messages.

GSN is ideal in domains in which communication needs to be used as a way to achieve some other objectives in the environment.

Thesis Question

How can Deep Reinforcement Learning agents learn to cooperate in a multiagent setting?

Showed that sharing parameters and replay memories can help multiple agents learn to perform a task.

Demonstrated communication can help agents cooperate in a domain featuring asymmetric information.

Future Work

Teammate modeling: Could such a model be used for planning or better cooperation?

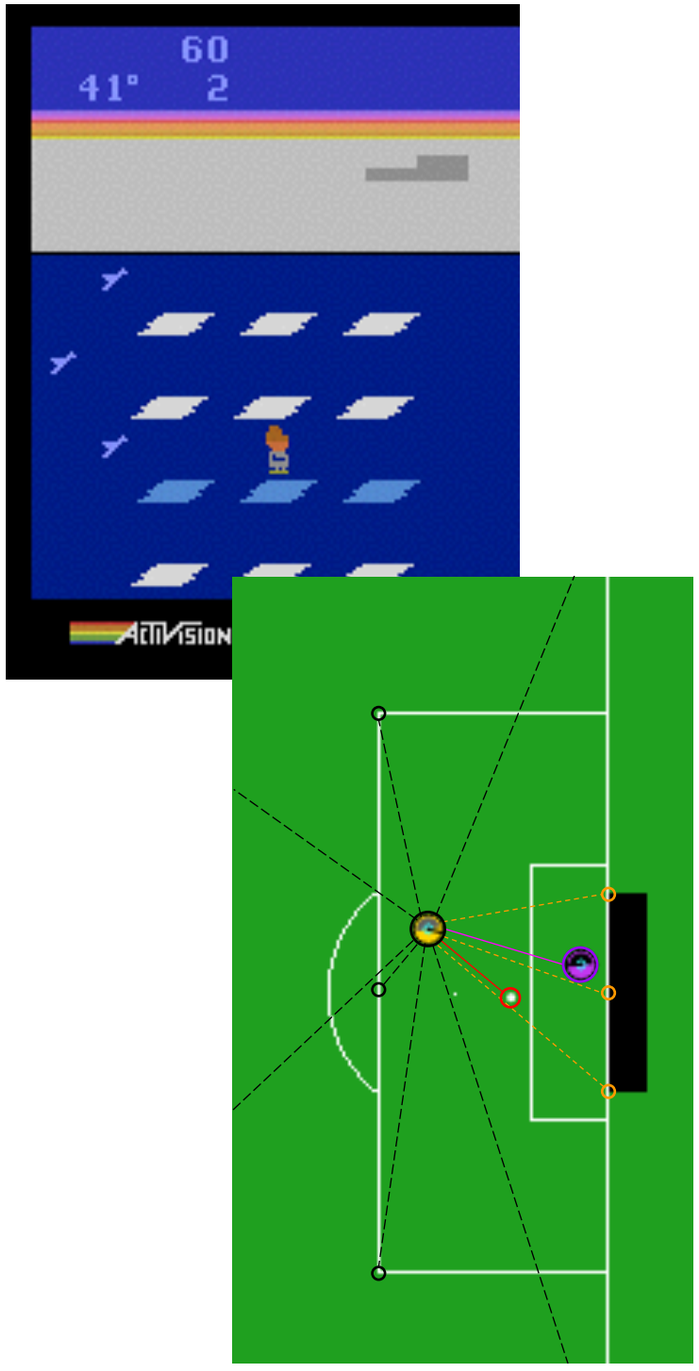
Embodied Imitation Learning: How can an agent learn from a teacher without directly observing the states or actions of the teacher?

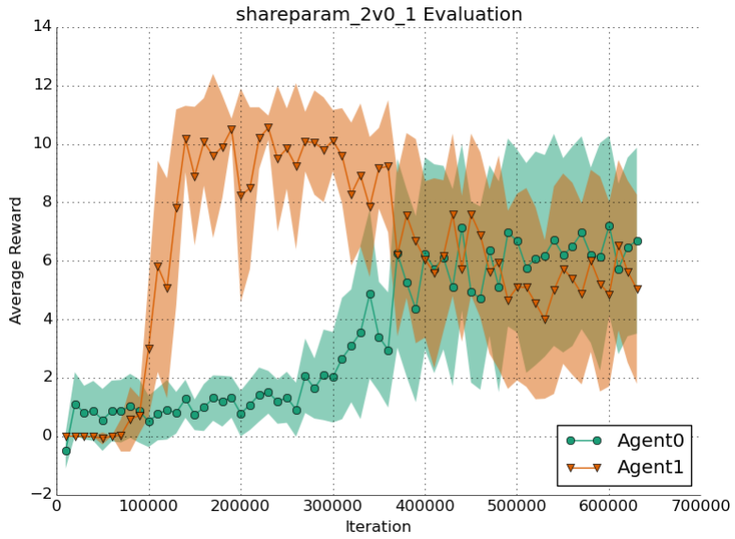
Adversarial multiagent learning: How to communicate in the presence of an adversary?

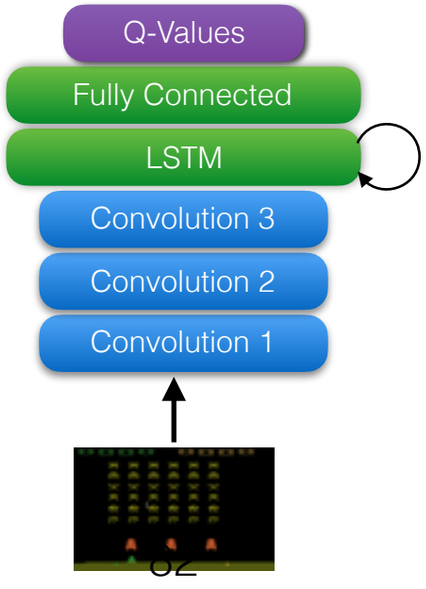
Contributions

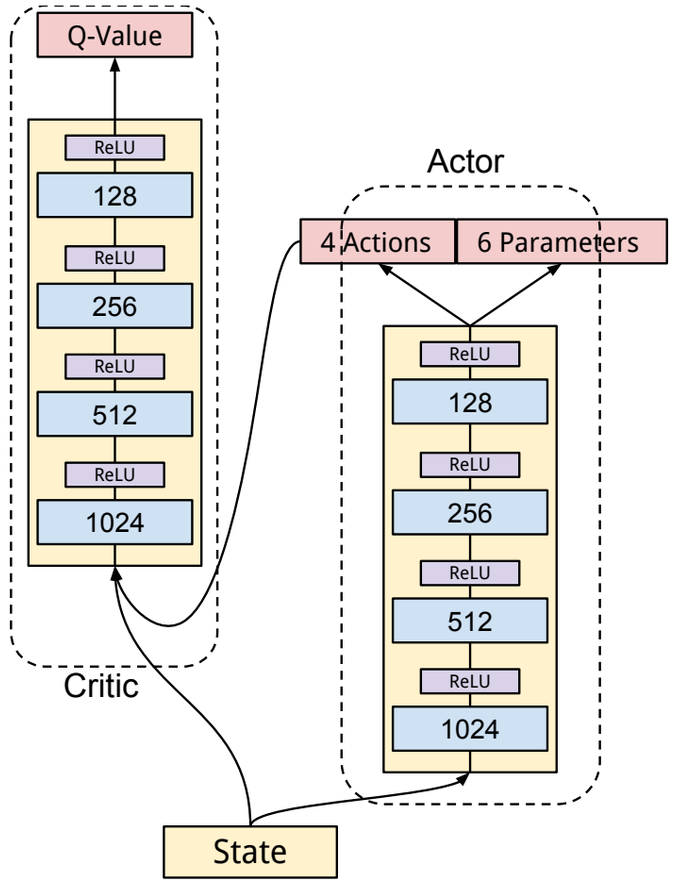
- Extended Deep RL algorithms to parameterized-continuous action space.
- Demonstrated that mixing bootstrap and Monte Carlo returns yields better learning performance.
- Introduced and analyzed parameter and memory sharing multiagent architectures.
- Introduced communication architectures and demonstrated that learned communication could help cooperation.
- Open source contributions: HFO, all learning agents

Thanks!


60
41°
2
ACTIVISION


shareparam_2v0_1 Evaluation
Average Reward
Iteration
Agent0
Agent1


Q-Values
Fully Connected
LSTM
Convolution 3
Convolution 2
Convolution 1


Q-Value
ReLU
128
ReLU
256
ReLU
512
ReLU
1024
Critic
Actor
4 Actions
6 Parameters
ReLU
128
ReLU
256
ReLU
512
ReLU
1024
State

Partially Observable MDP (POMDP)

Observation o_t

Action a_t

Reward r_t

Observations provide noisy or incomplete information

Memory may help to learn a better policy

Atari Environment

Reward is change in game score

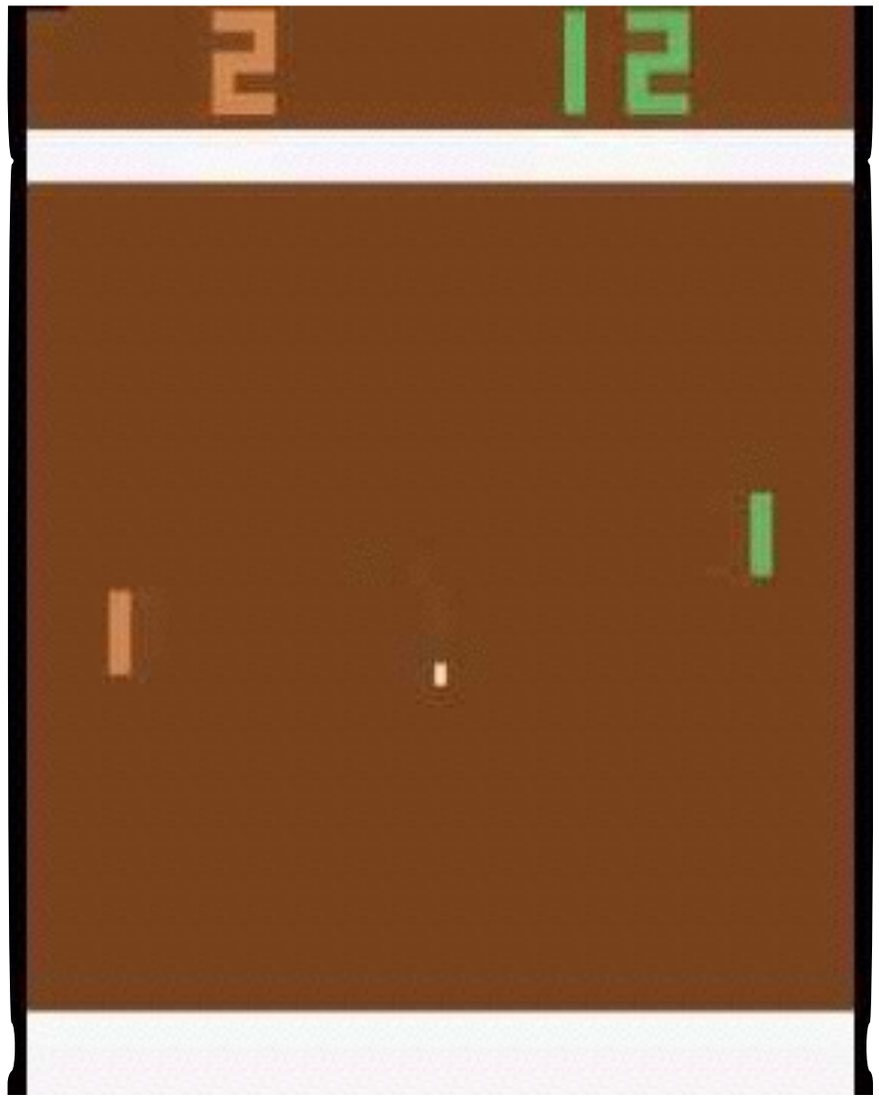
Resolution 160x

18 discrete actions

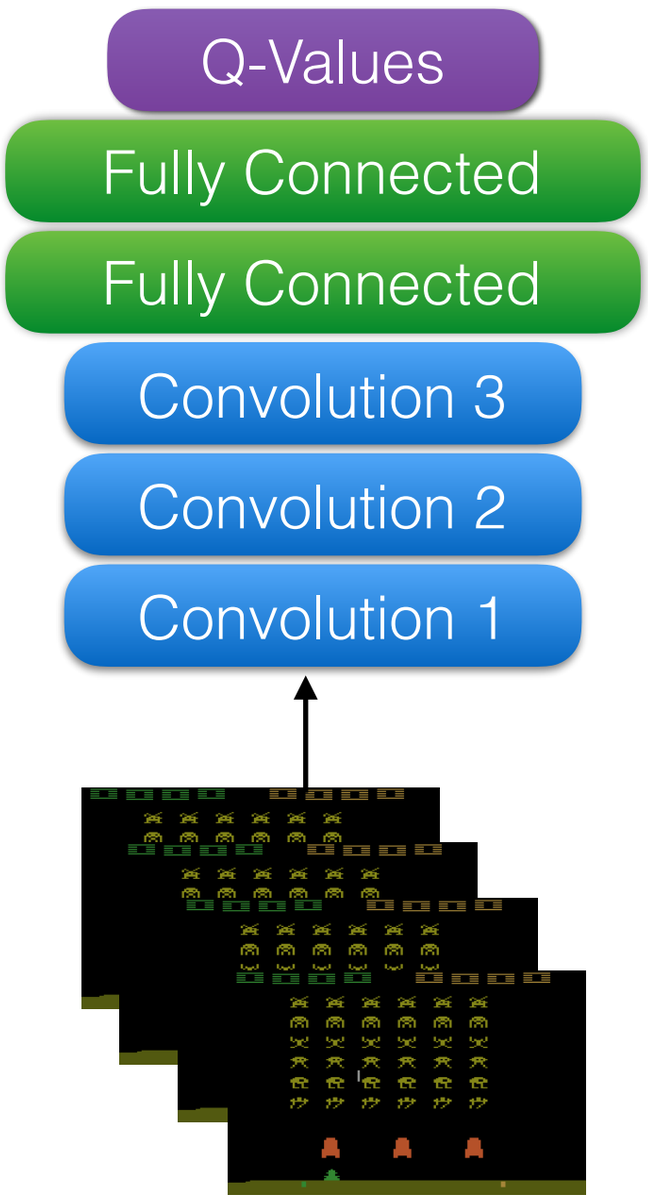
Atari: MDP or POMDP?

Depends on the number game screens used in the state representation.

Many games PO with a single frame.



Deep Q-Network (DQN)



Neural network estimates Q-Values Q(s,a) for all 18 actions:

$$Q(s|\theta) = (Q_{s,a_1} \cdots Q_{s,a_n})$$

Learns via temporal difference:

$$L_i(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(Q(s_t|\theta) - y_i \right)^2 \right]$$

$$y_i = r_t + \gamma max(Q(s_{t+1}|\theta))$$

Accepts the last 4 screens as input.

Flickering Atari

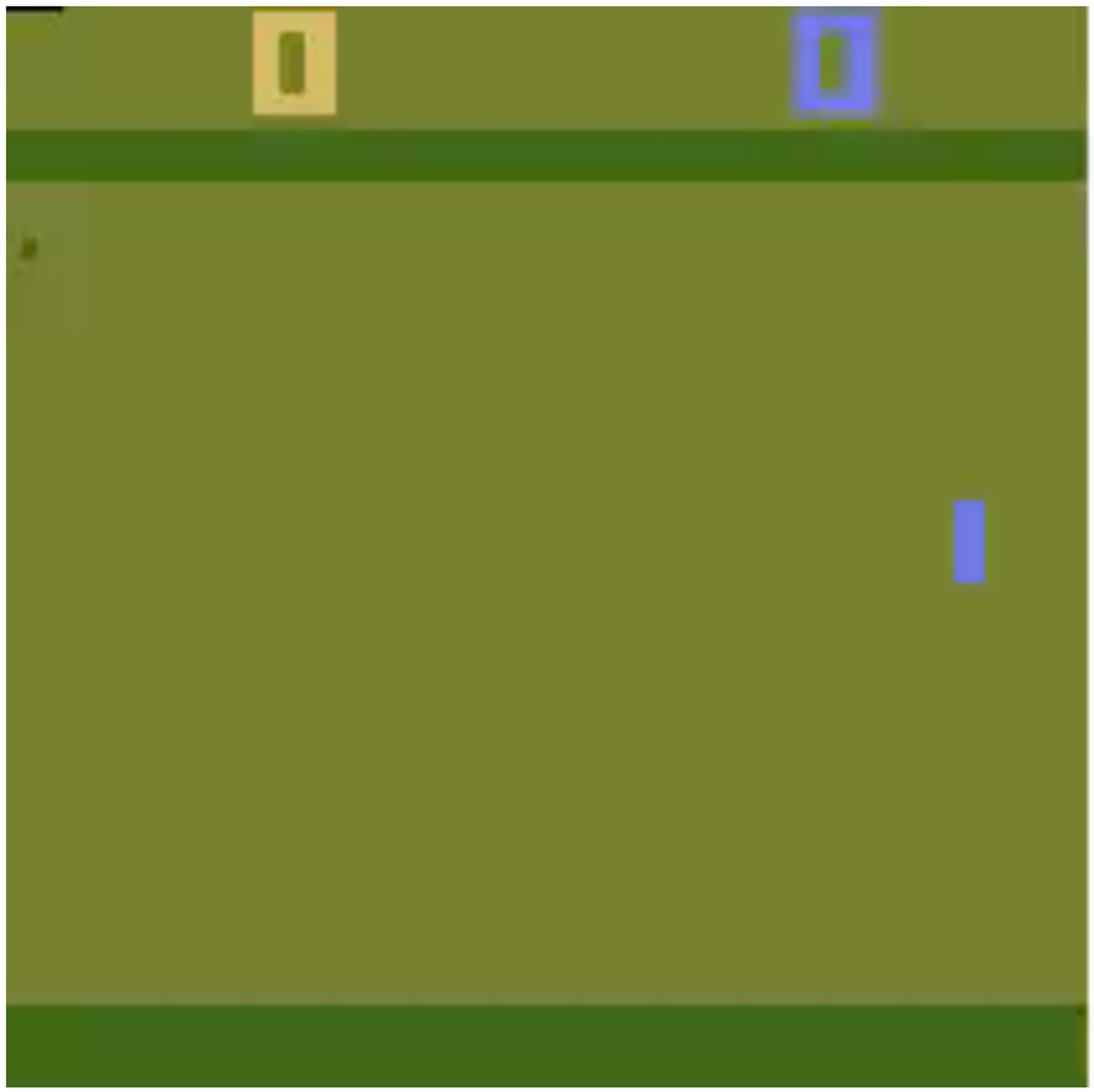
How well does DQN perform on POMDPs?

Induce partial observability by stochastically obscuring the game screen

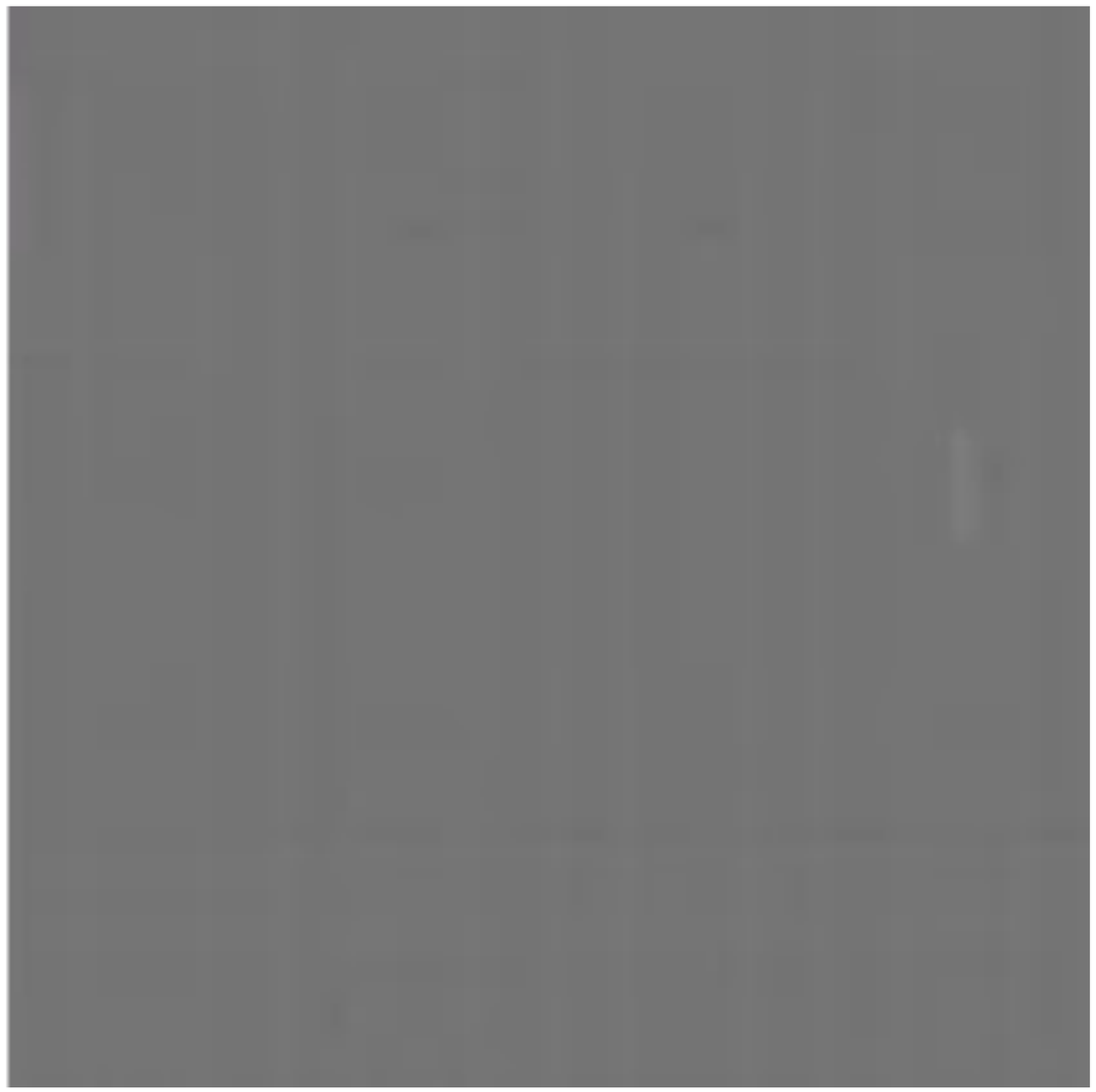
$$o_t = \begin{cases} s_t & \text{with } p = \frac{1}{2} \\ < 0, \ldots, 0 > & \text{otherwise} \end{cases}$$

Game state must be inferred from past observations

DQN Pong

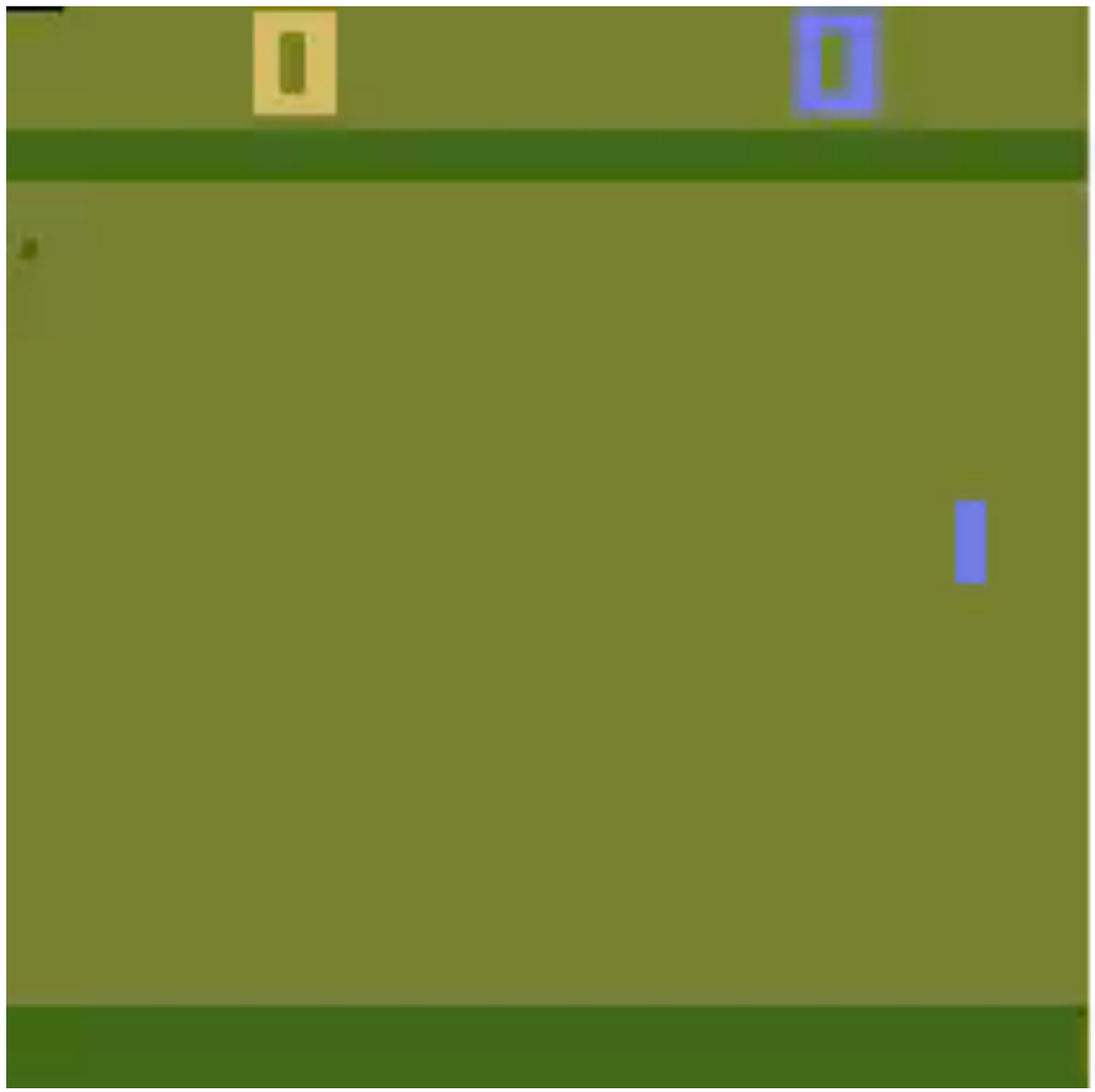


True Game Screen

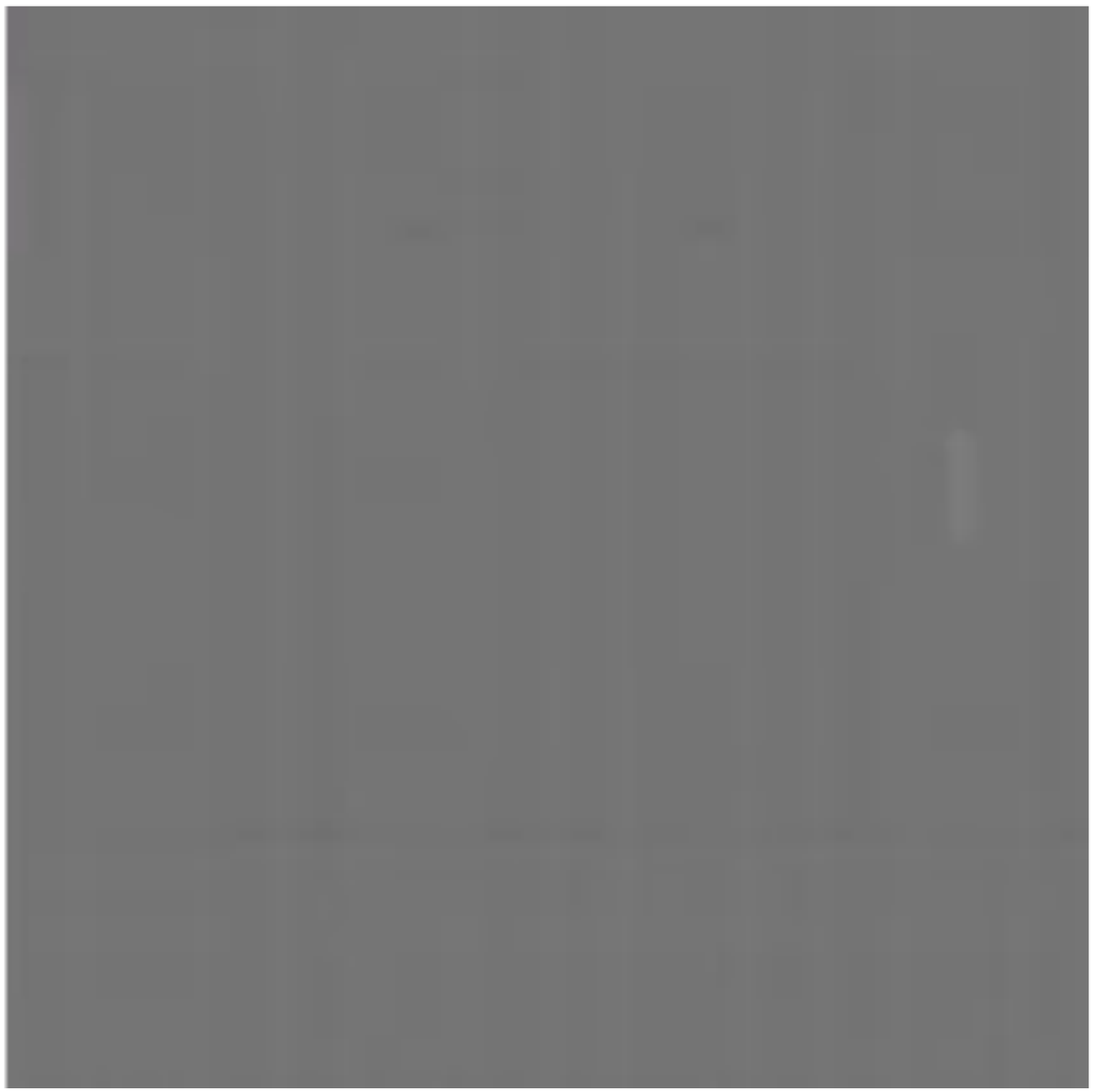


Observed Game Screen

DQN Flickering Pong

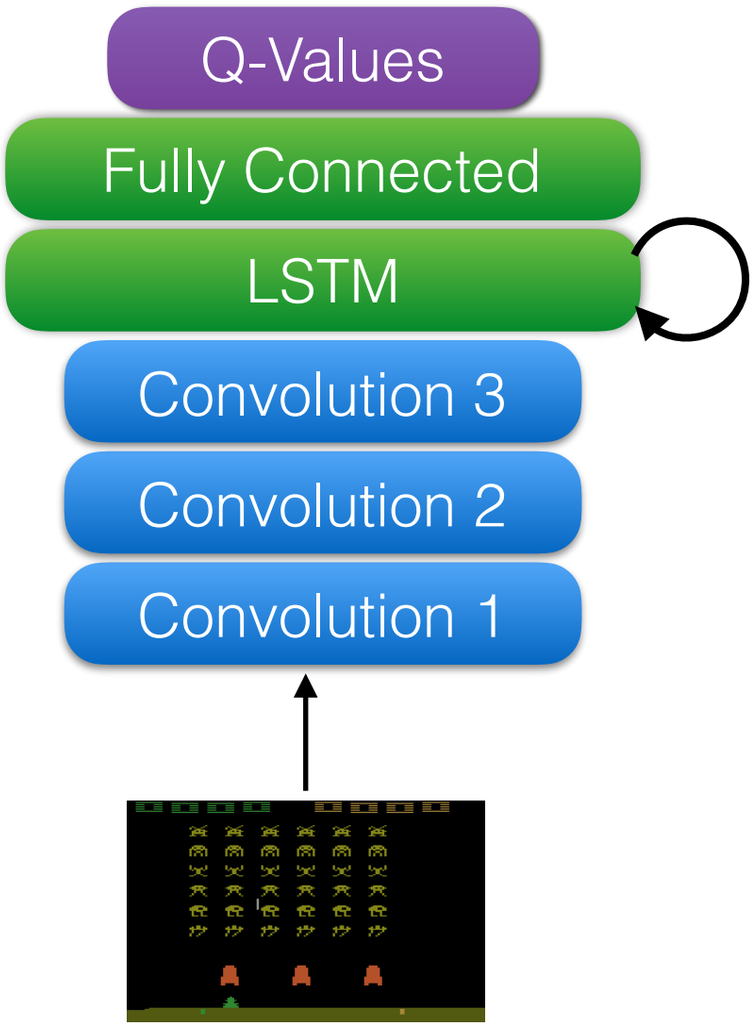


True Game Screen



Observed Game Screen

Deep Recurrent Q-Network



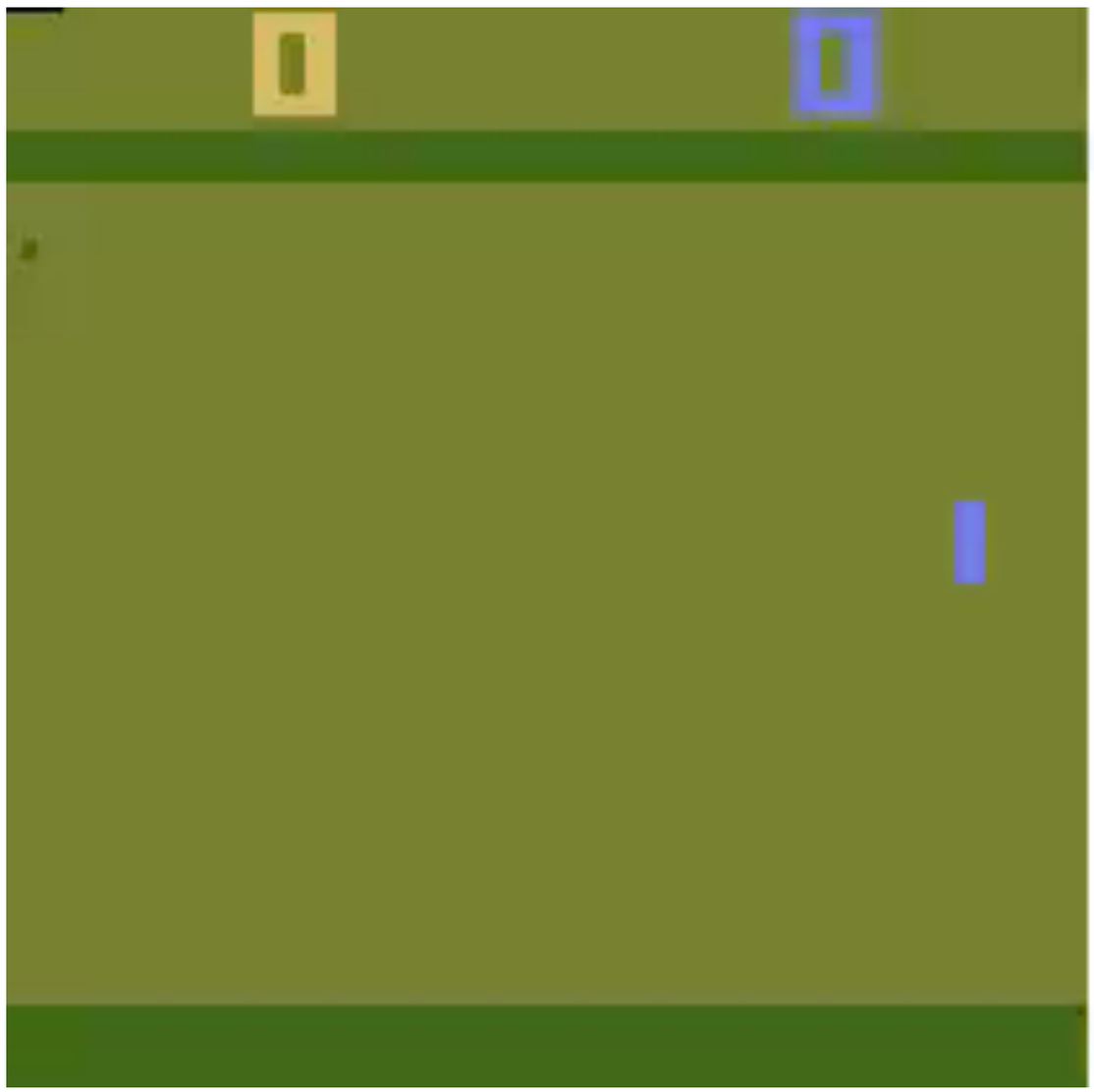
Uses a Long Short Term Memory (LSTM) to selectively remember past game screens.

Architecture identical to DQN except:

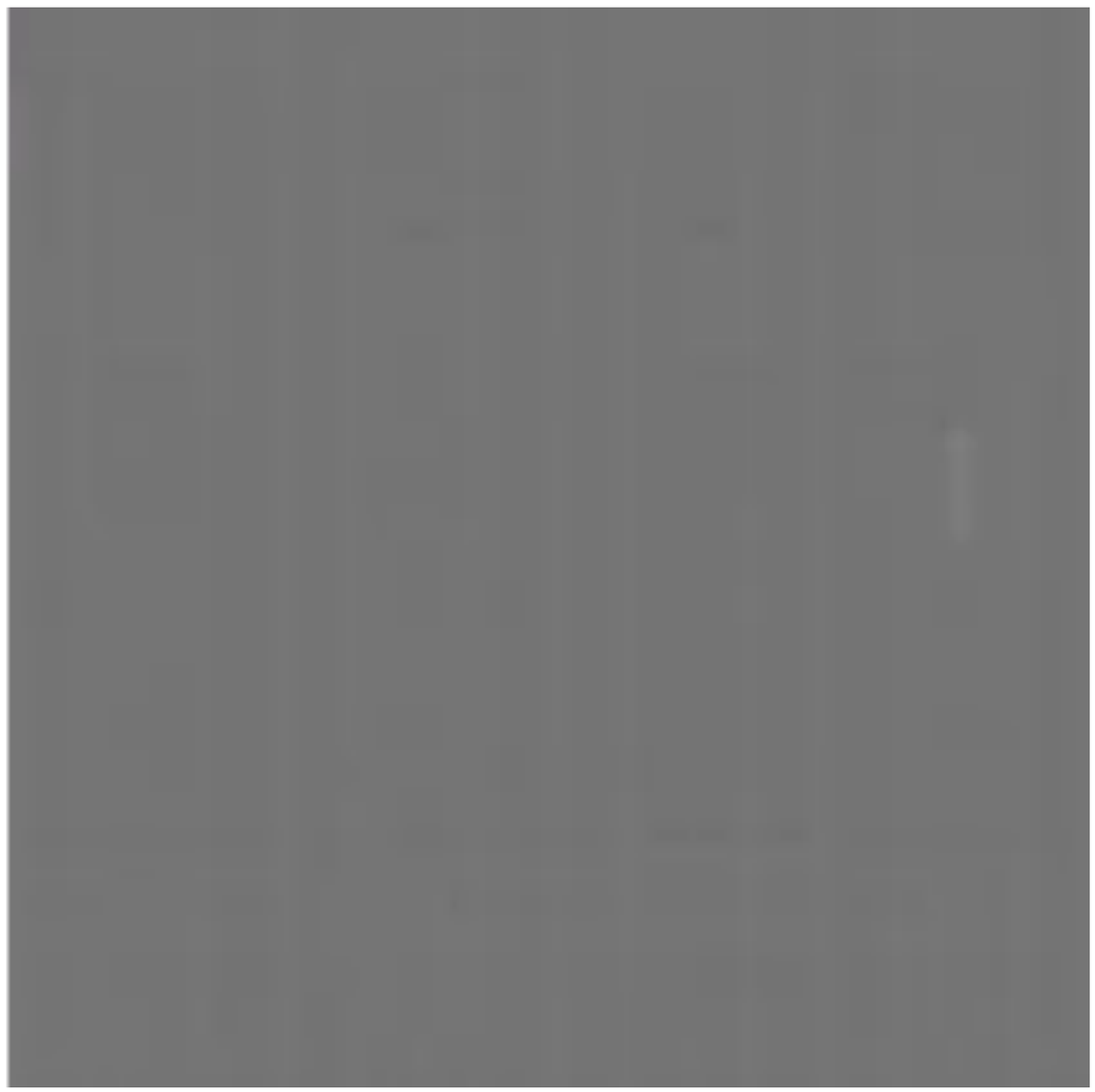
1. Replaces FC layer with LSTM
2. Single frame as input each timestep

Trained end-to-end using BPTT for last 10 timesteps.

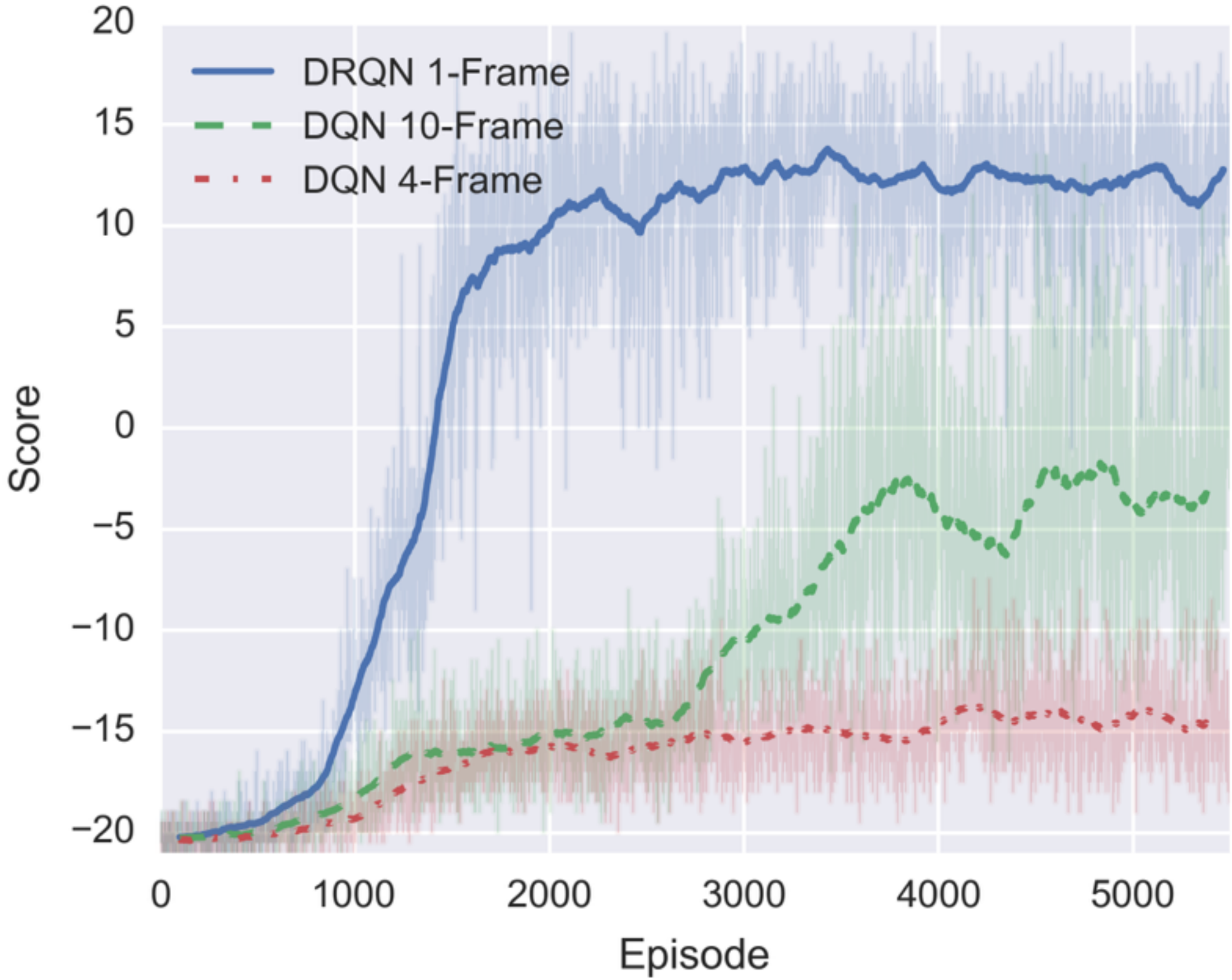
DRQN Flickering Pong



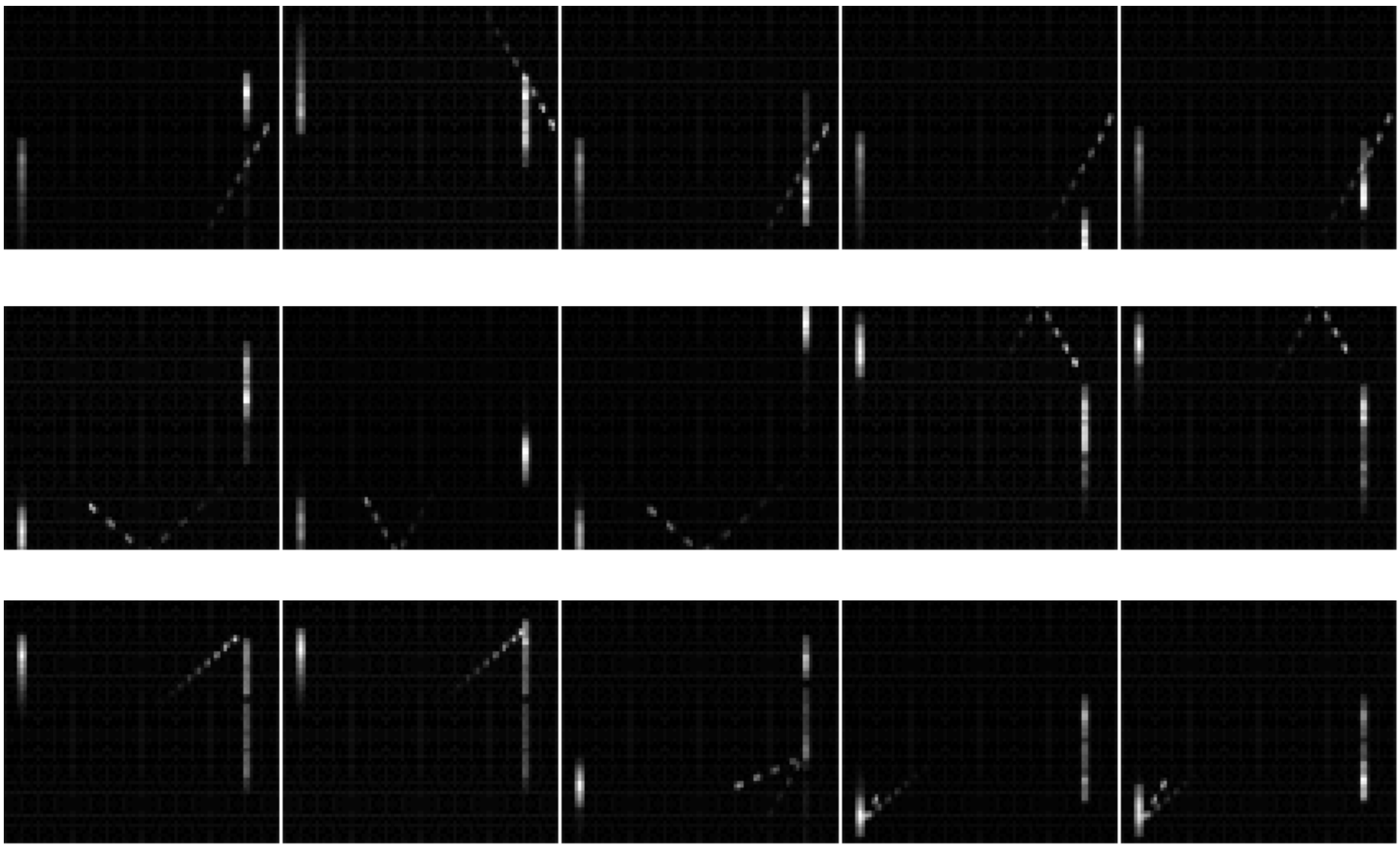
True Game Screen



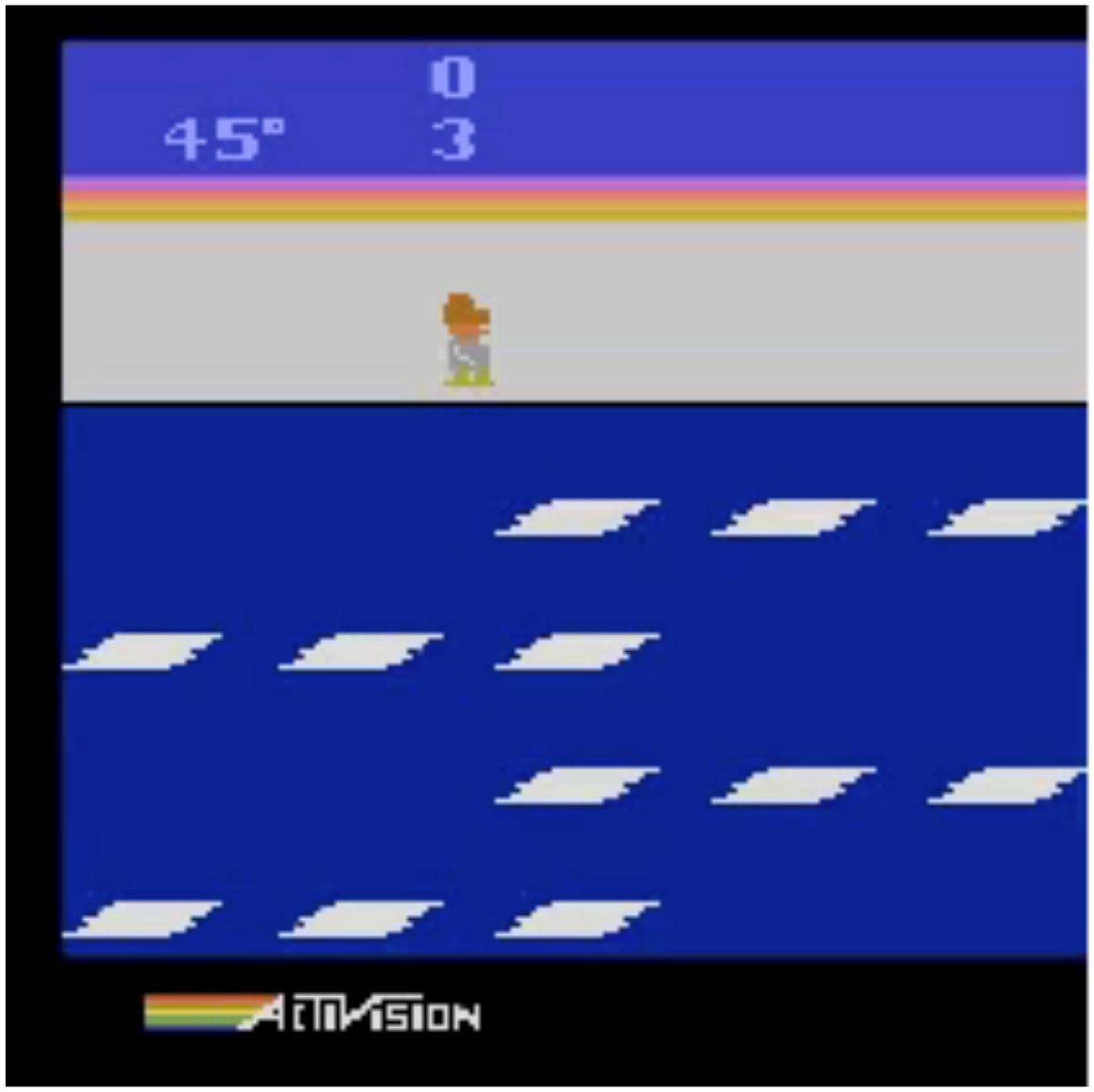
Observed Game Screen

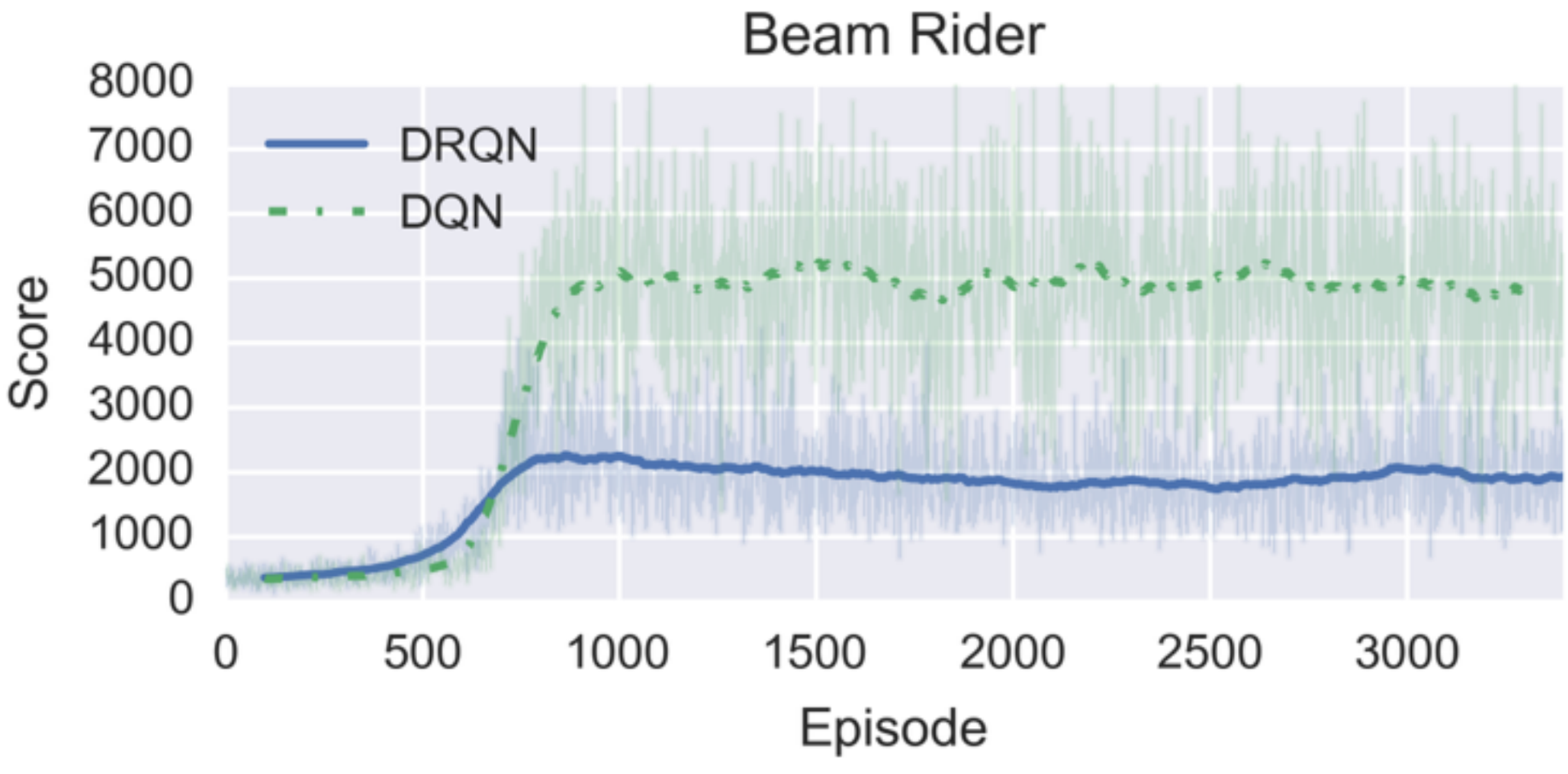

DRQN 1-Frame
DQN 10-Frame
DQN 4-Frame
Score
20
15
10
5
0
−5
−10
−15
−20
0
1000
2000
3000
4000
5000
Episode

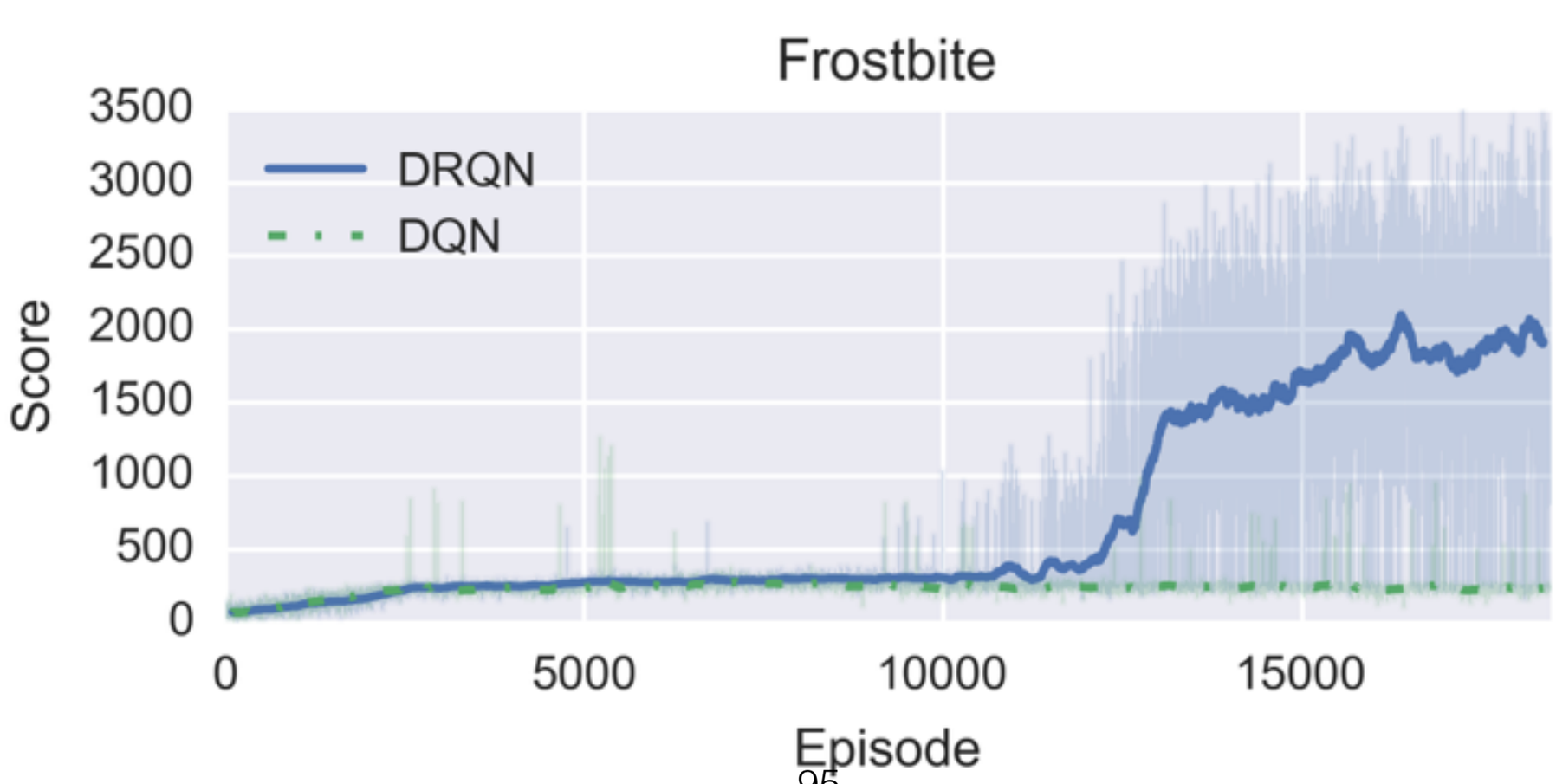
LSTM infers velocity



DRQN Frostbite




Beam Rider
8000
7000
6000
5000
4000
3000
2000
1000
0
Score
DRQN
DQN
0
500
1000
1500
2000
2500
3000
Episode


Frostbite
3500
3000
2500
2000
1500
1000
500
0
Score
DRQN
DQN
0
5000
10000
15000
Episode

Extensions

DRQN has been extended in several ways:

- Addressable Memory: *Control of Memory, Active Perception, and Action in Minecraft;* Oh et al. in ICML ’16

- Continuous Action Space: *Memory Based Control with Recurrent Neural Networks;* Heess et al., 2016

[*Deep Recurrent Q-Learning for Partially Observable MDPs,* Hausknecht et al, 2015; ArXiv]

Bounded Action Space

HFO's continuous parameters are bounded

Dash(direction, power)
Turn(direction)
Tackle(direction)
Kick(direction, power)

Direction in [-180,180], Power in [0, 100]

Exceeding these ranges results in no action

If DDPG is unaware of the bounds, it will invariably exceed them

Bounded DDPG

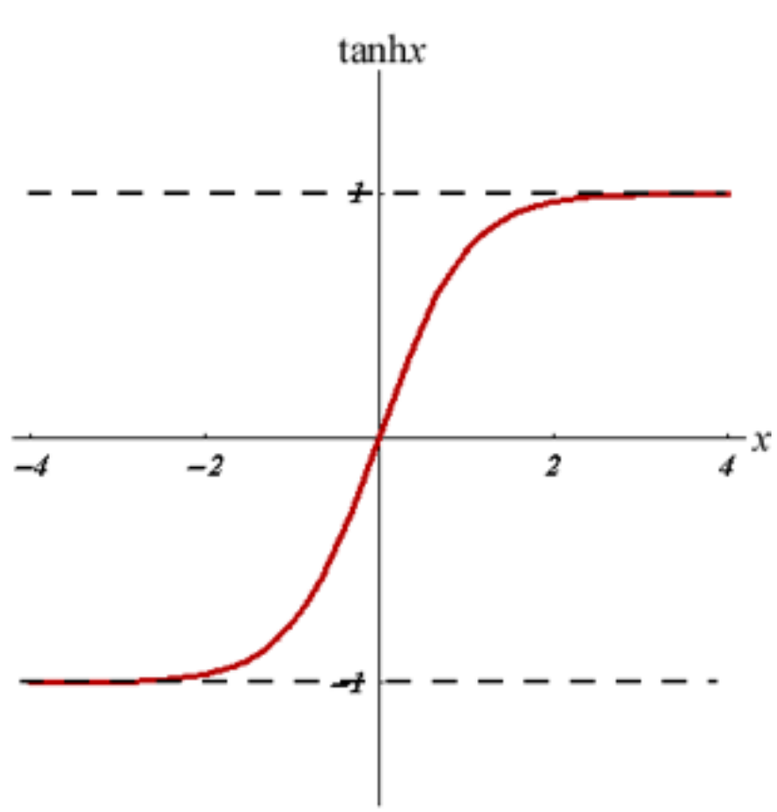
We examine 3 approaches for bounding the DDPG's action space:

1. Squashing Function
2. Zero Gradients
3. Invert Gradients

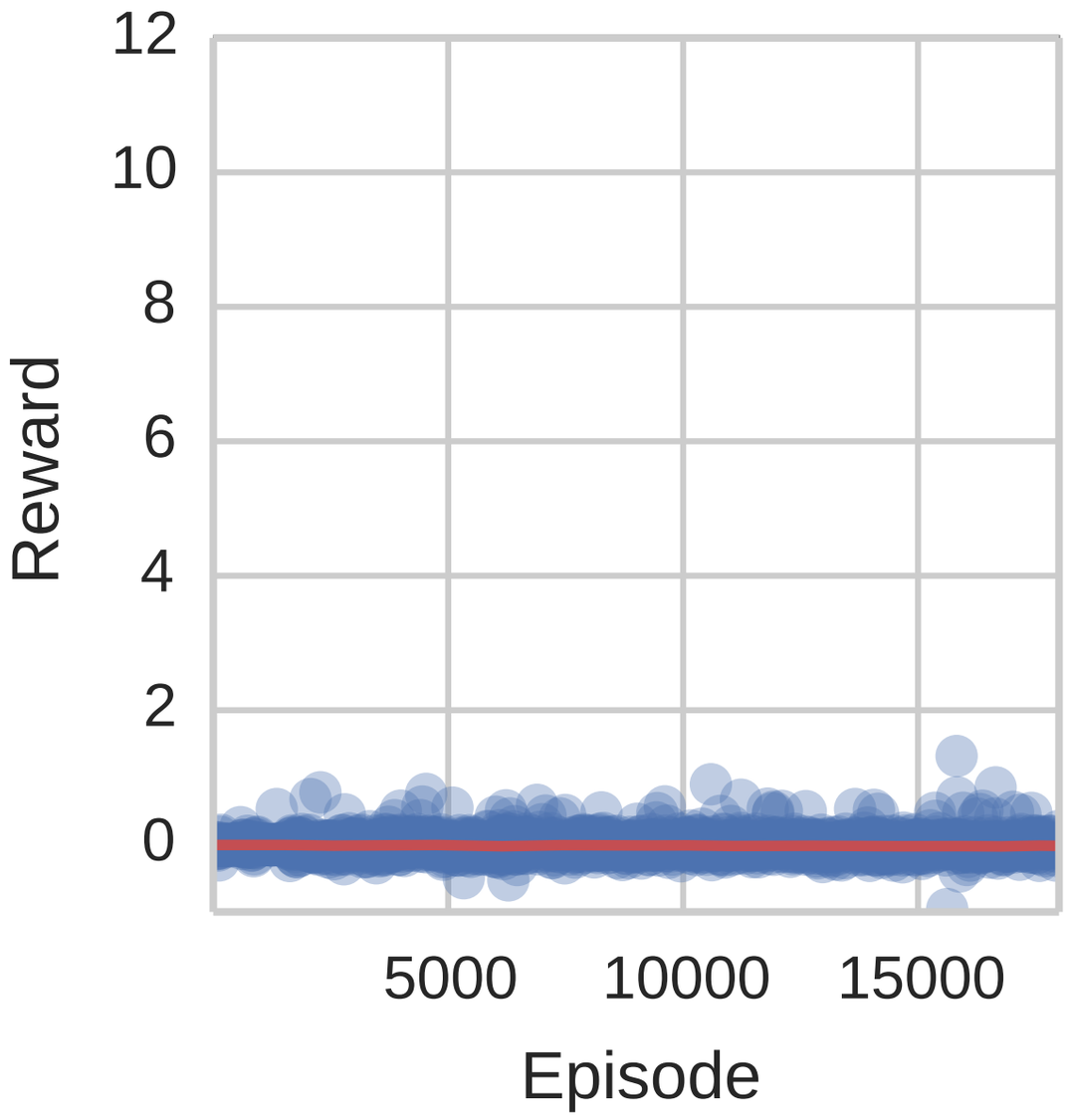
Squashing Function

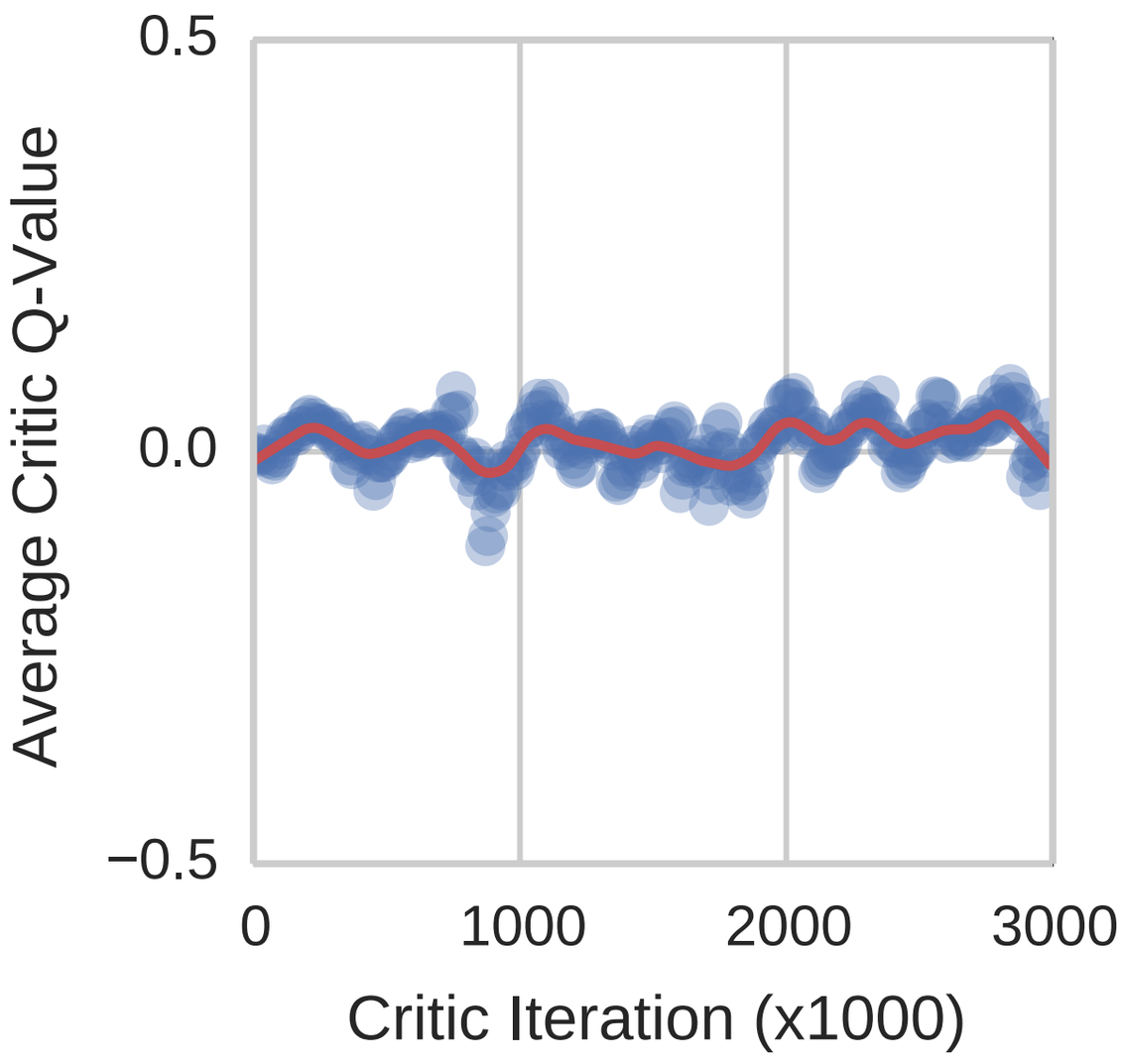
1. Use Tanh non-linearity to bound parameter output

2. Rescale into desired range



Squashing Function





Zeroing Gradients

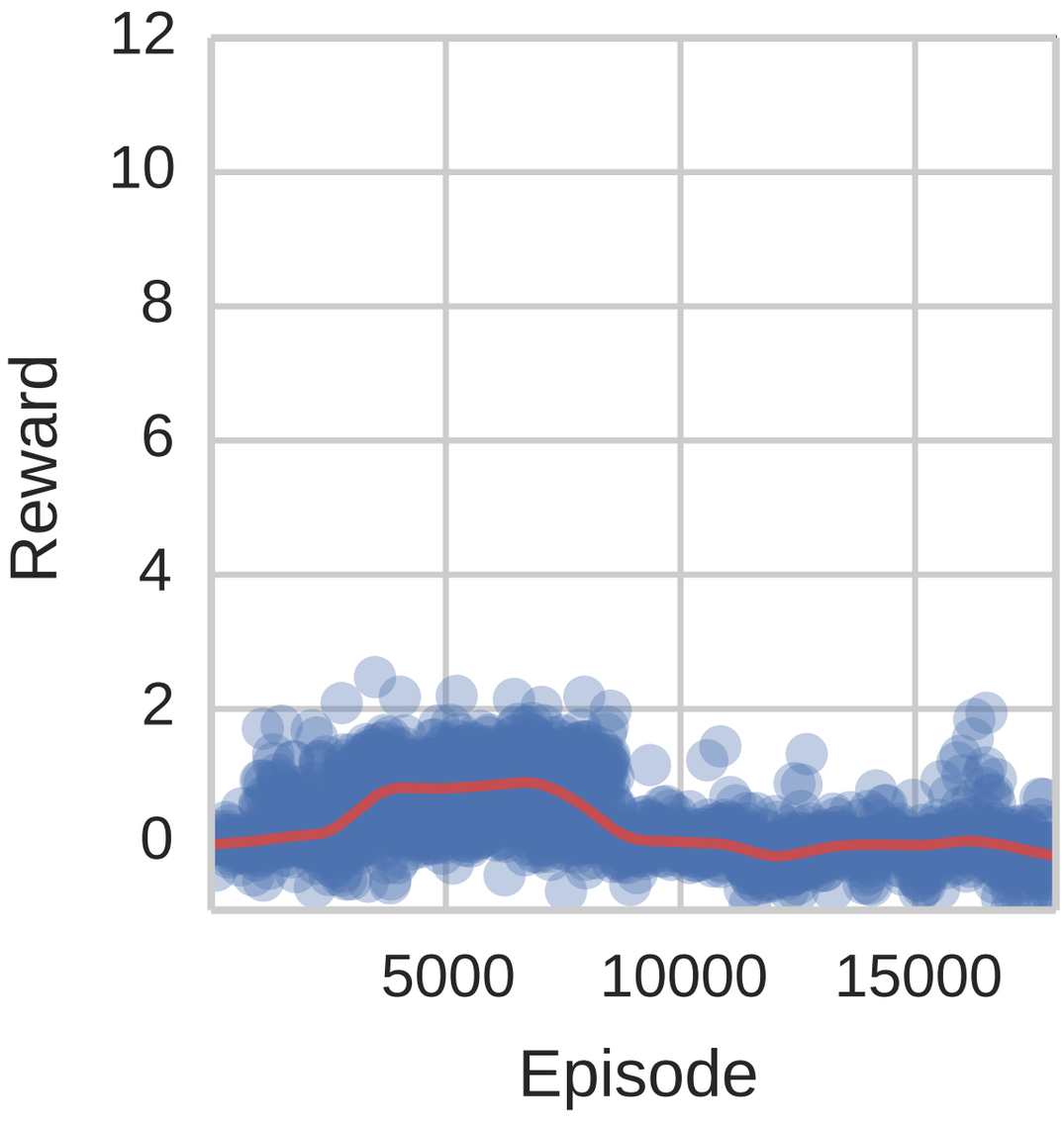
Each continuous parameter has a range: [p_{min}, p_{max}]

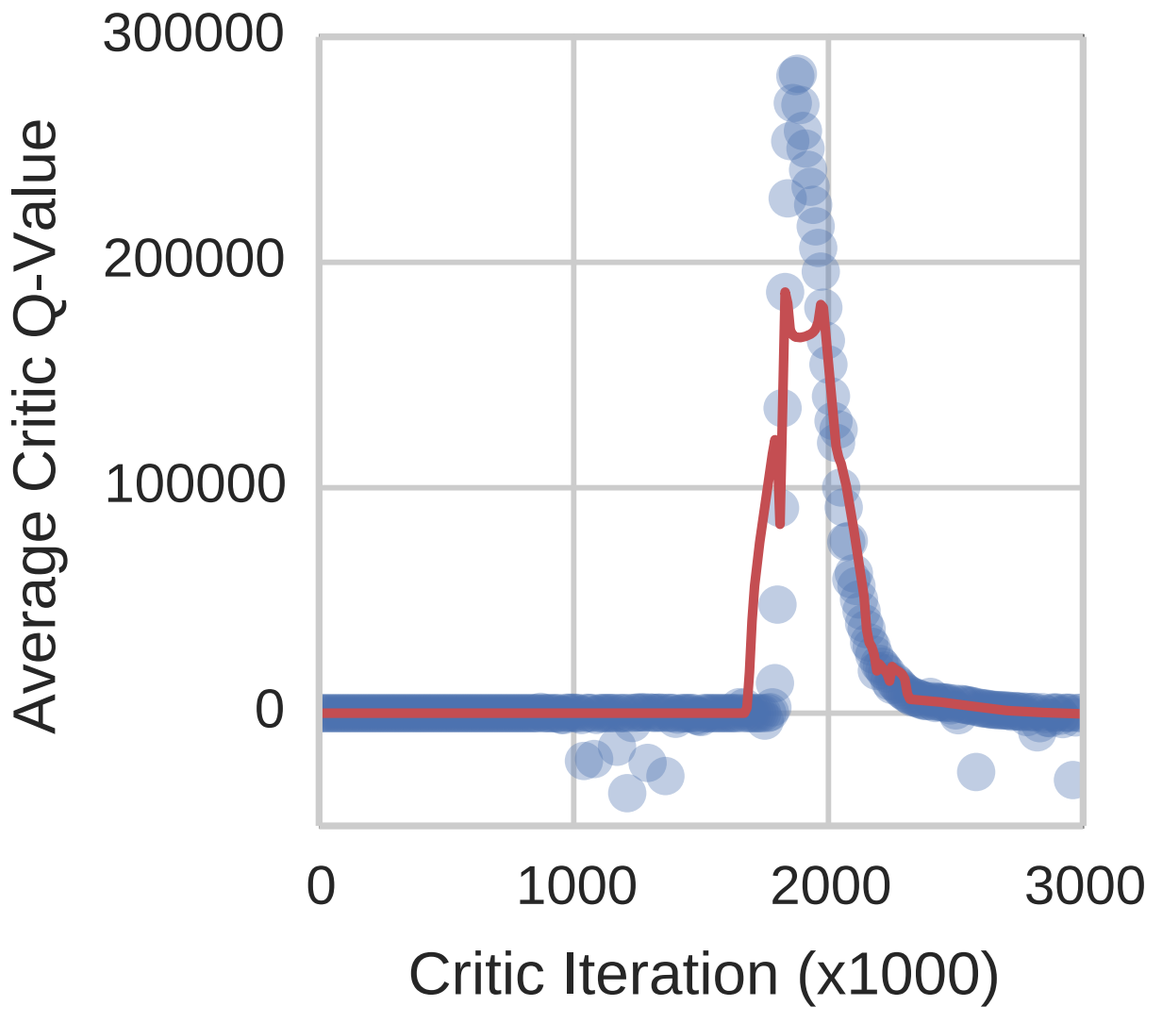
Let p denote current value of parameter, and ∇_p the suggested gradient.

Then:

$$\nabla_p = \begin{cases} \nabla_p & \text{if } p_{\min} < p < p_{\max} \\ 0 & \text{otherwise} \end{cases}$$

Zeroing Gradients





Inverting Gradients

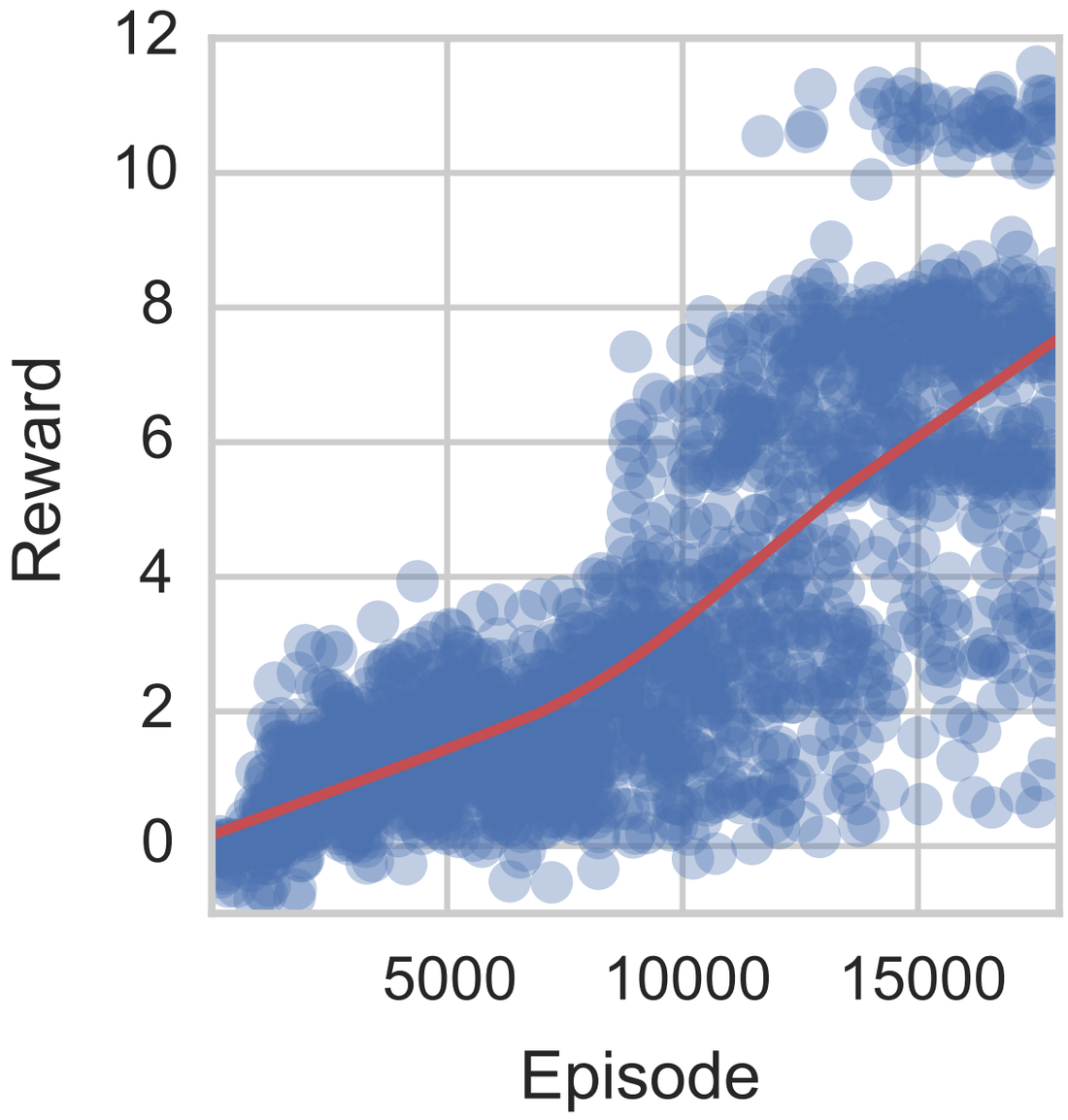
For each parameter:

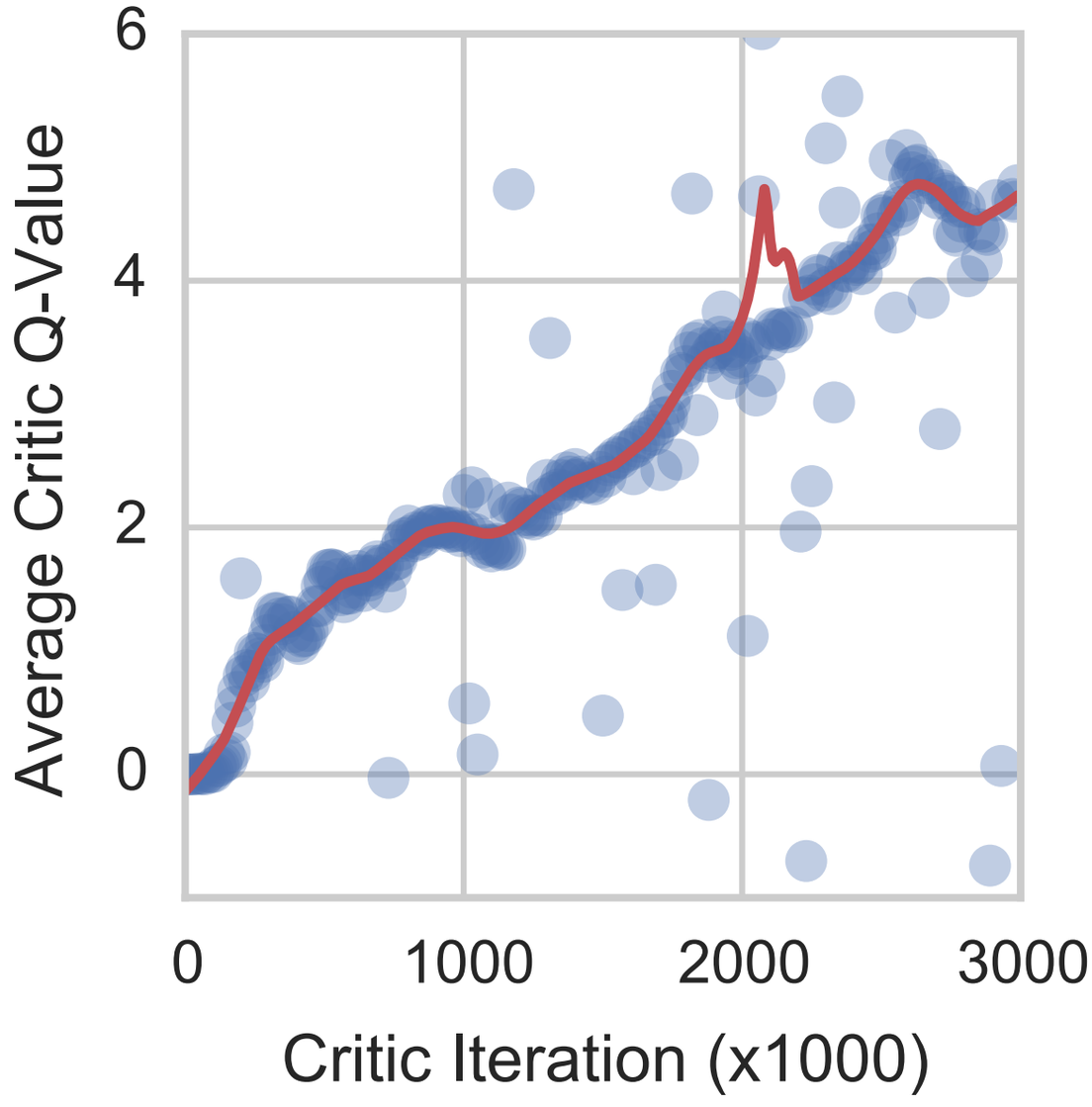
$$\nabla_p = \nabla_p \cdot \begin{cases} (p_{\max} - p)/(p_{\max} - p_{\min}) & \text{if } \nabla_p \text{ suggests increasing } p \\ (p - p_{\min})/(p_{\max} - p_{\min}) & \text{otherwise} \end{cases}$$

Allows parameters to approach the bounds of the ranges without exceeding them

Parameters don’t get “stuck” or saturate

Inverting Gradients



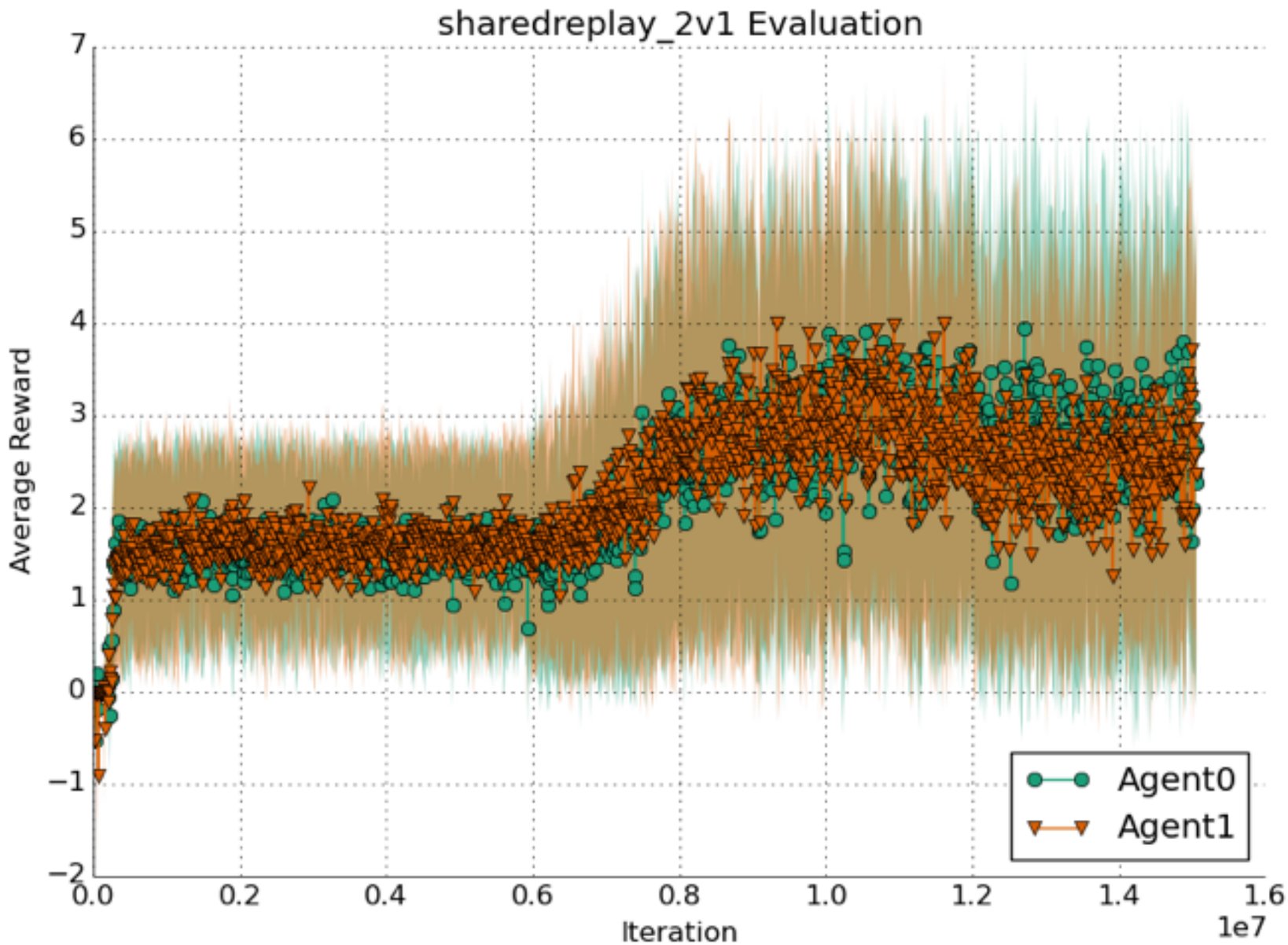


2v1

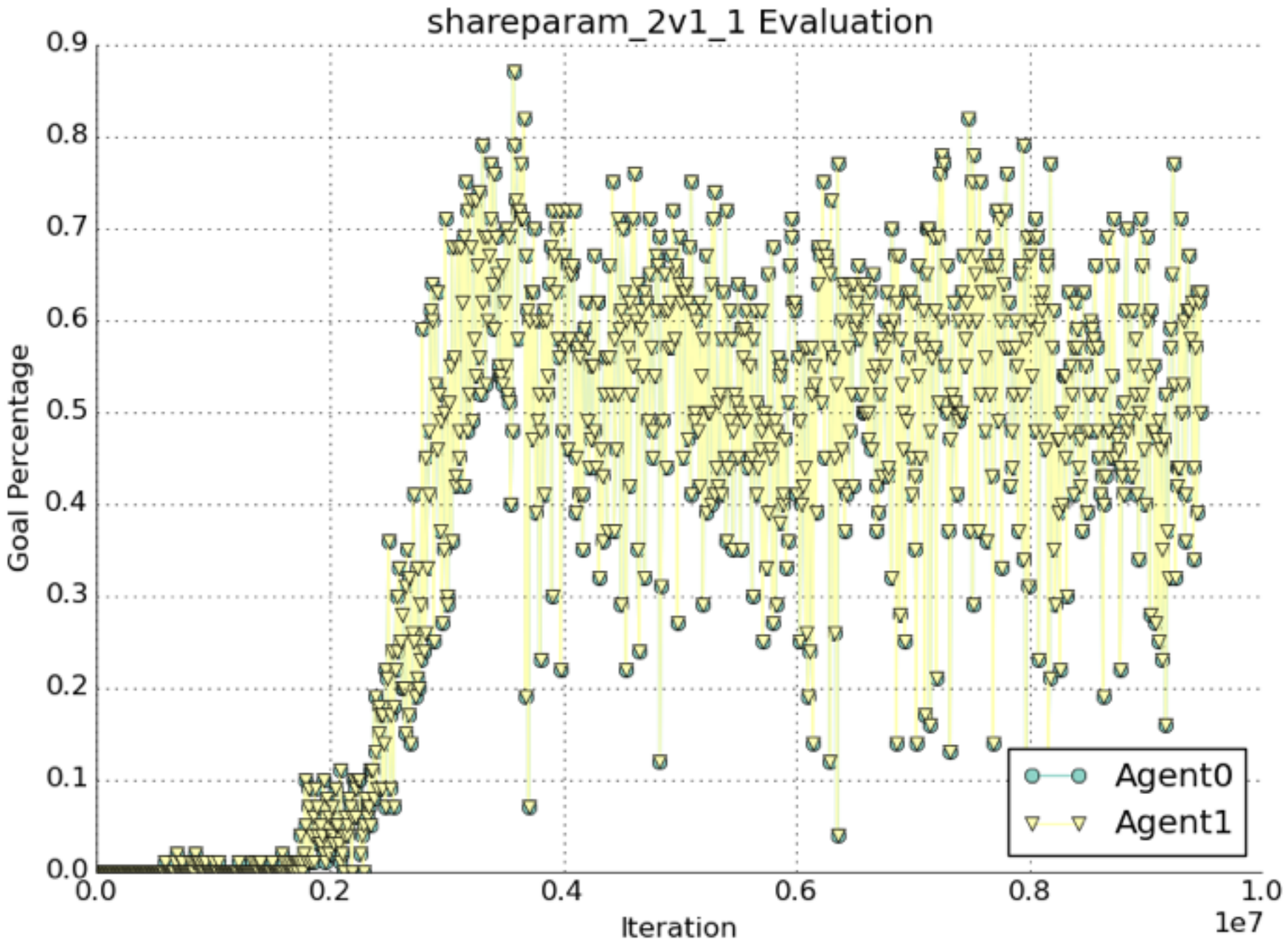
- Can agents learn cooperative behaviors like passing and cross kicks?

- Hypothesis: Cross kicks can help achieve more reliable scoring in 2v1 setting. Can sharing architectures learn such behaviors?

2v1


sharedreplay_2v1 Evaluation
Average Reward
Iteration
1e7
Agent0
Agent1
7
6
5
4
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

2v1


shareparam_2v1_1 Evaluation
Goal Percentage
Iteration
1e7
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0.0
0.2
0.4
0.6
0.8
1.0
Agent0
Agent1


108
1
8000
7
8000
11
8000

2v1

- Both memory sharing and parameter sharing result in reasonably high goal percentage

- Agents do not learn passes or cross kicks and instead rely on individual attacks

Curriculum Learning

- Motivation: it's difficult to design unbiased reward functions for complex tasks.

- Easier to break a complex task into many subtasks, learn each subtask, and then use the skills to address the complex task.

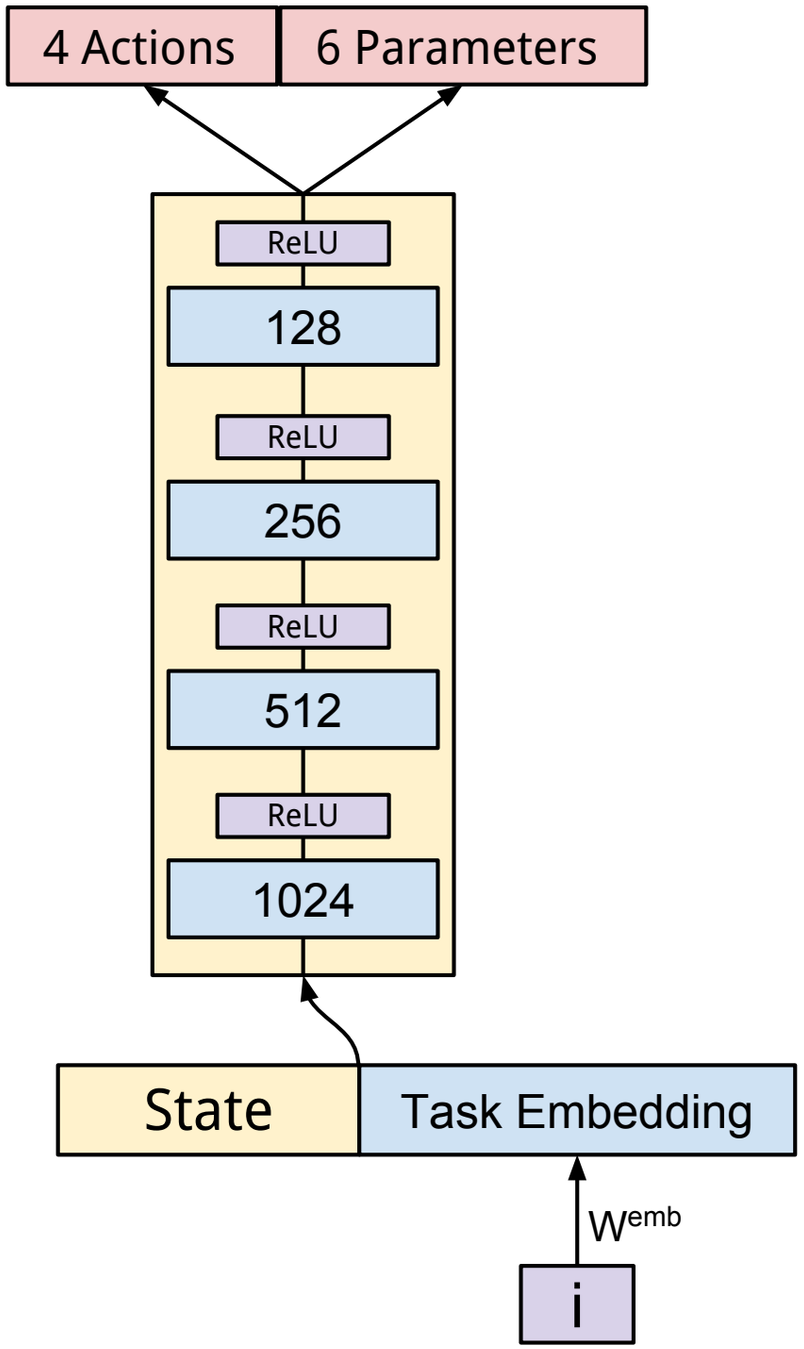
- Given: Complex target task with sparse reward function, curriculum of tasks with non-sparse reward.

- Goal: Learn how to perform all tasks in curriculum including the target task.

State Embed Architecture

Each task in curriculum is represented as an embedding vector.

Task embedding vector is concatenated with agent's observation.

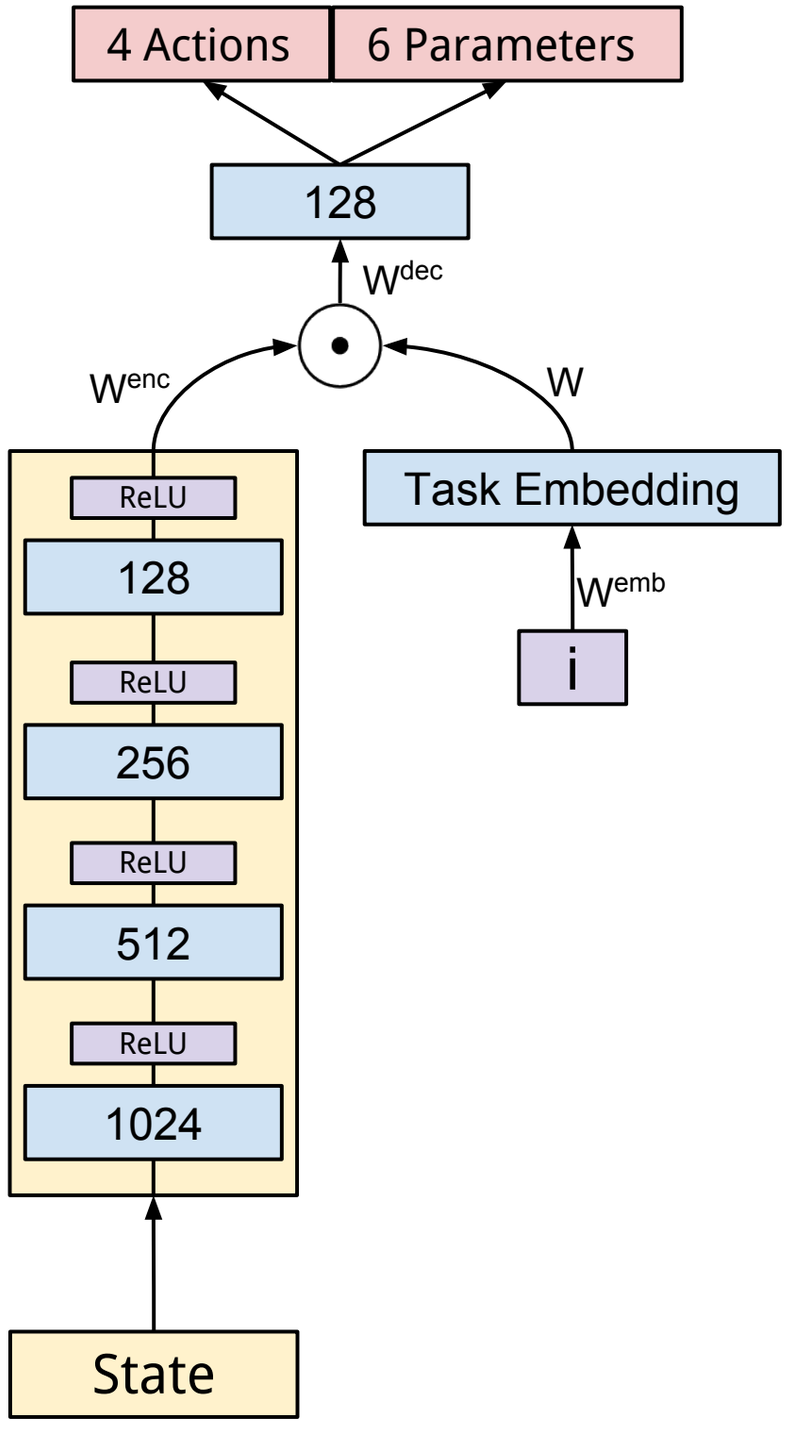


Weight Embed Architecture

Each task in curriculum is represented as an embedding vector.

Weight embedding architecture conditions the activations of a particular layer on the task embedding.

Allows a single network to learn many tasks and act uniquely in each task.



Curriculum

- Target Task: Score on Goal
- Curriculum: Move to Ball, Kick to Goal
- Each task in curriculum corresponds to one skill in the target task.
- Tasks are represented using an embedding vector.

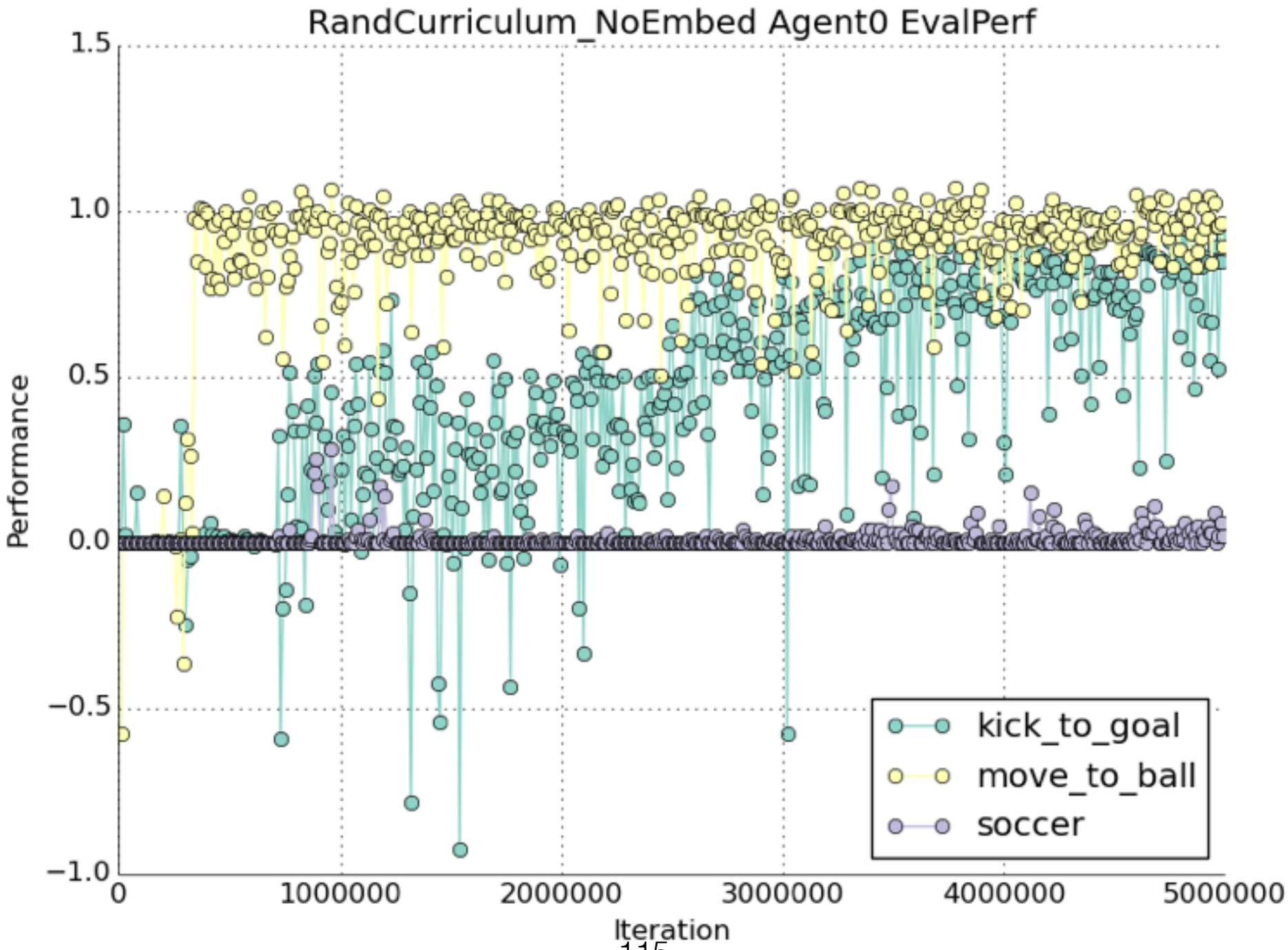
Curriculum Ordering

- The order of training tasks has an impact on ultimate performance.

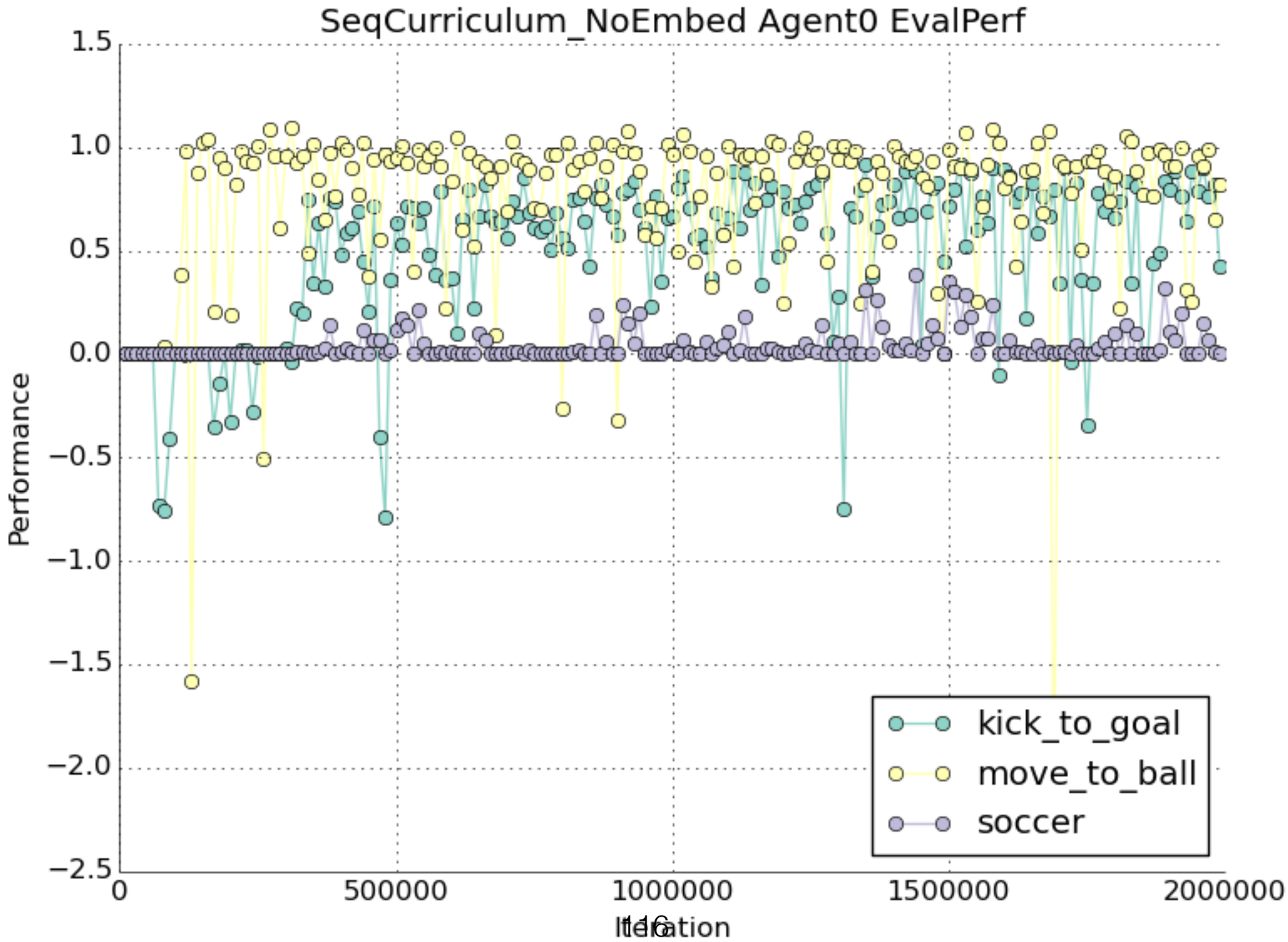
- Random curriculum: Presents a random task in the curriculum at each episode

- Sequential curriculum: Easiest tasks presented first, then harder tasks. Each task trained until convergence.

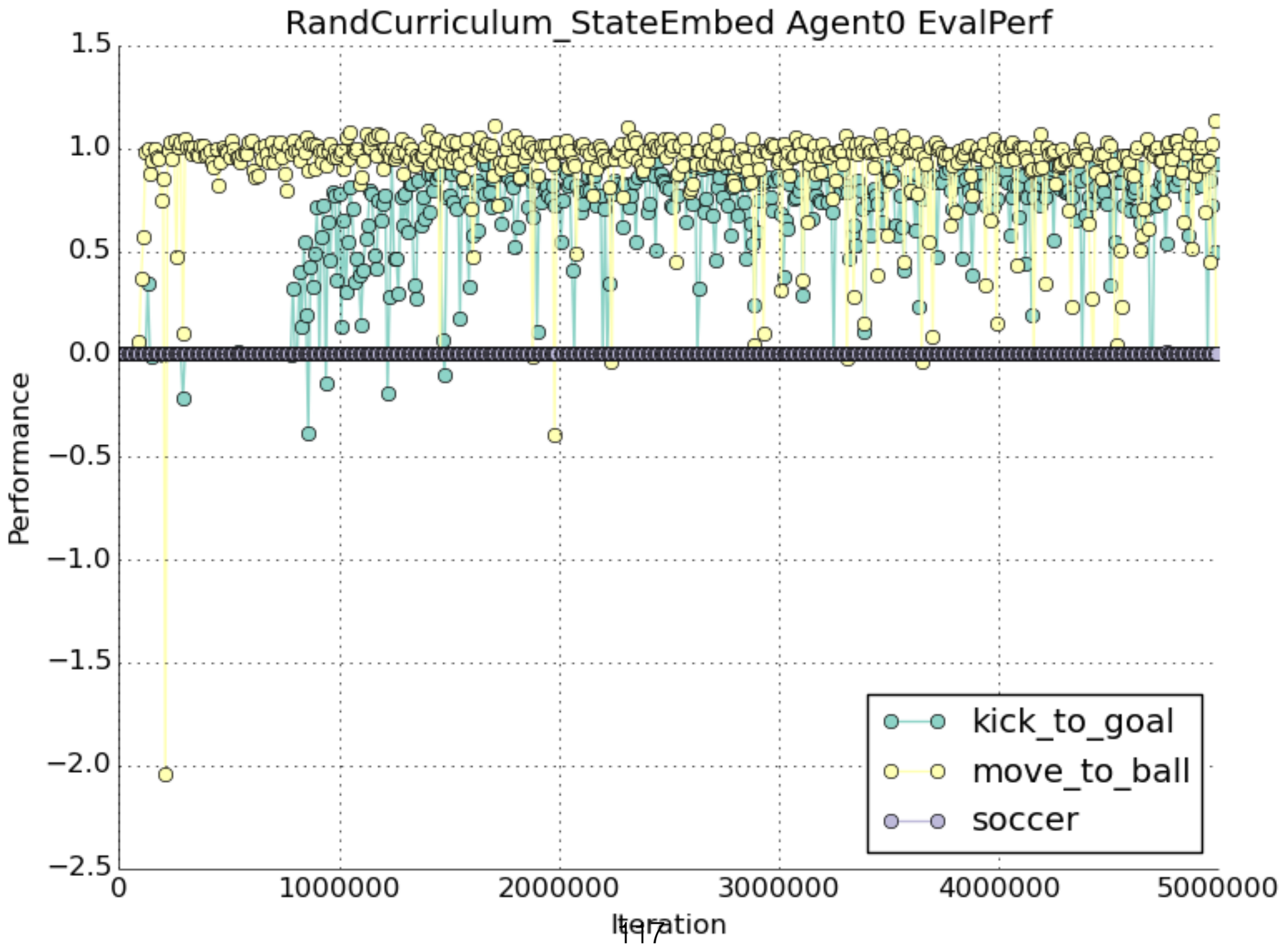
Random Curriculum, No Embedding



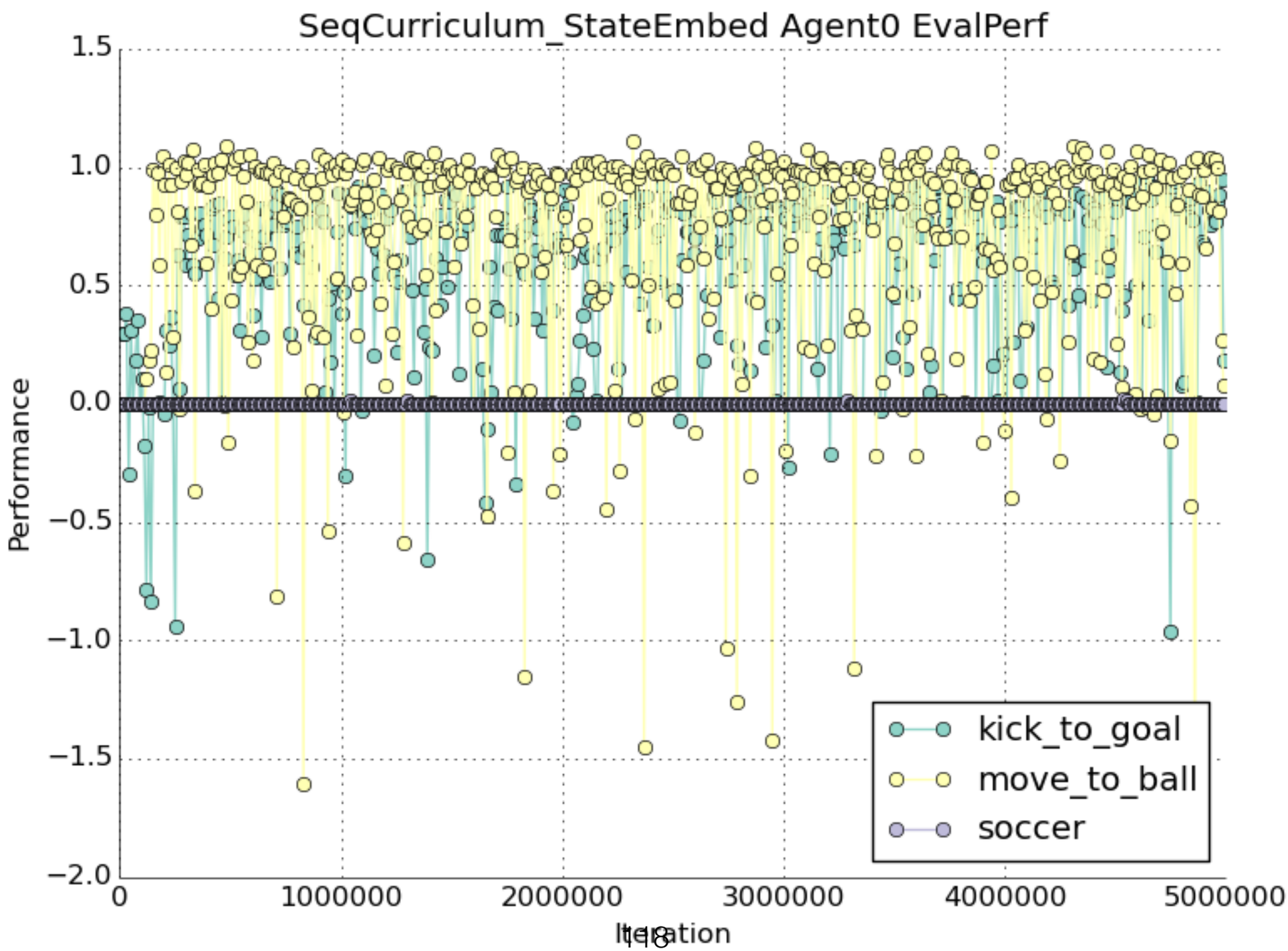
Seq Curriculum, No Embedding



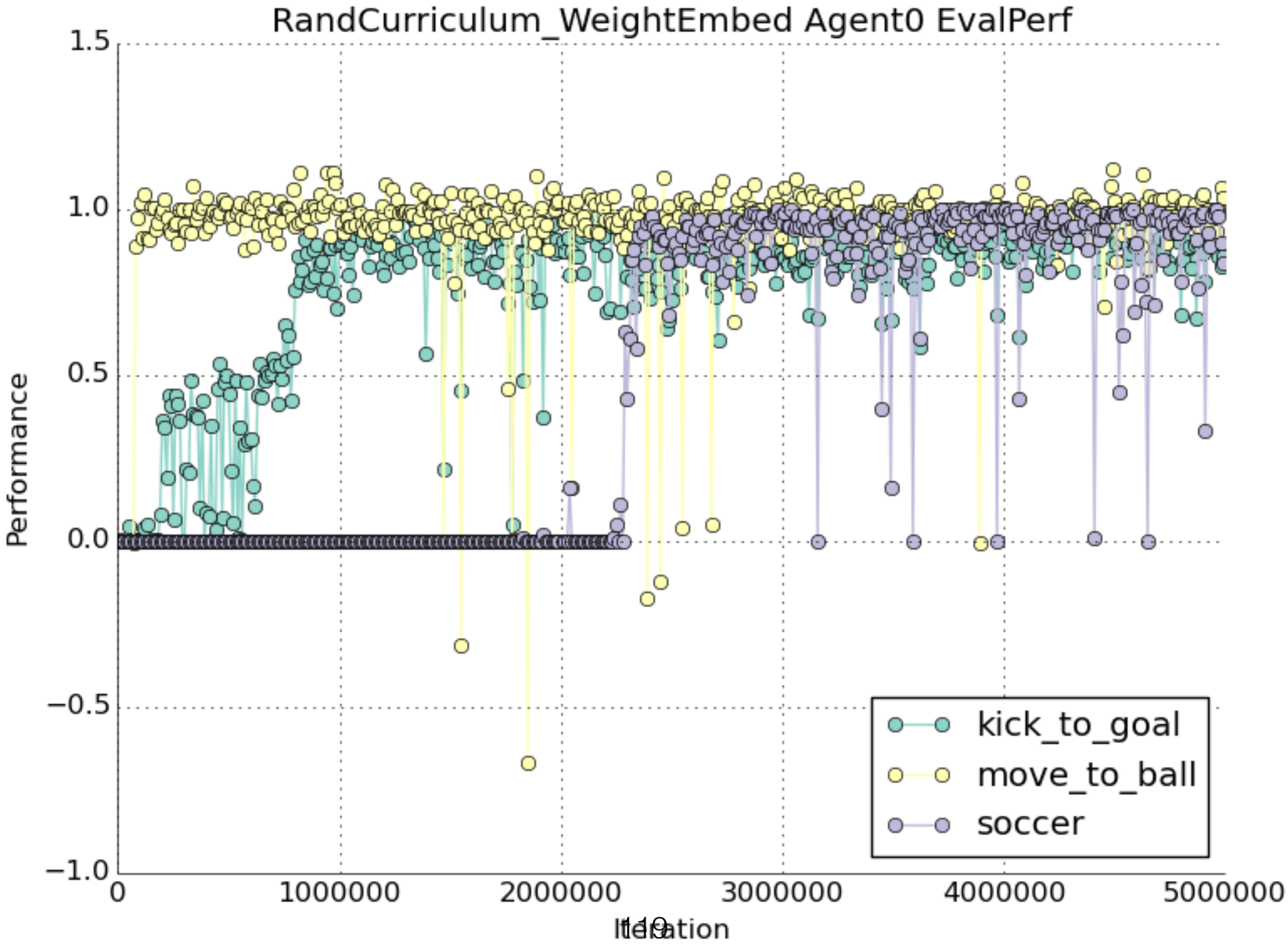
Random Curriculum, State Embedding



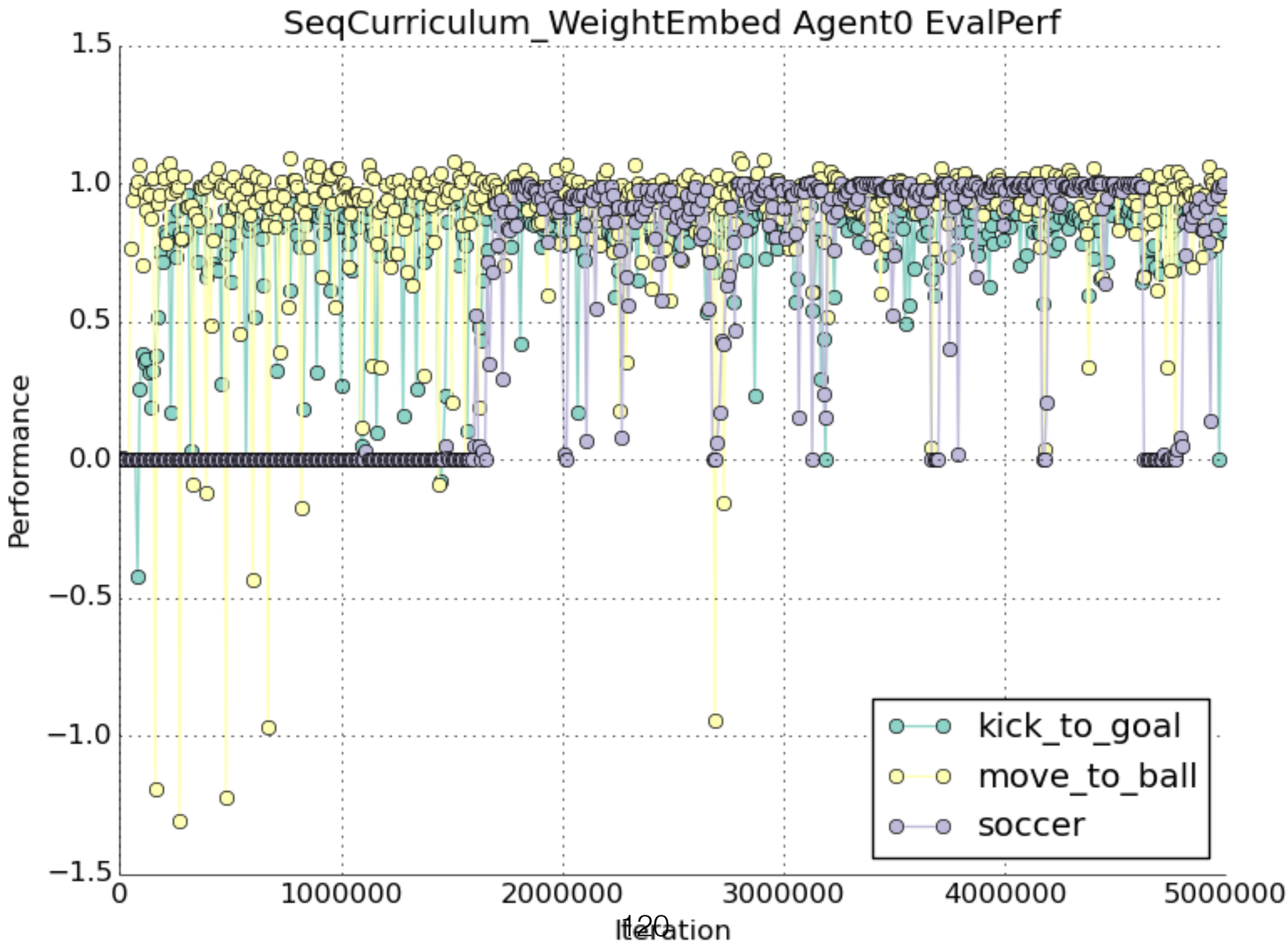
Seq Curriculum, State Embedding



Random Curriculum, Weight Embedding



Seq Curriculum, Weight Embedding



Curriculum Learning

- Agents can reuse learned skills to perform the soccer task which features a sparse goal reward

- Agents must continue to revisit all training tasks or they will forget previous skills

- Ablation experiments show that all tasks are necessary for the soccer curriculum