Deep RL and Multi-Agent RL

Sriyash Poddar

poddarsriyash@gmail.com

https://sriyash.me

* Slides taken from CS 285 by S. Levine and MARL tutorial at EACL by V.D. Silva

Deep RL with Q-Functions

Recap: Q-learning

full fitted Q-iteration algorithm:

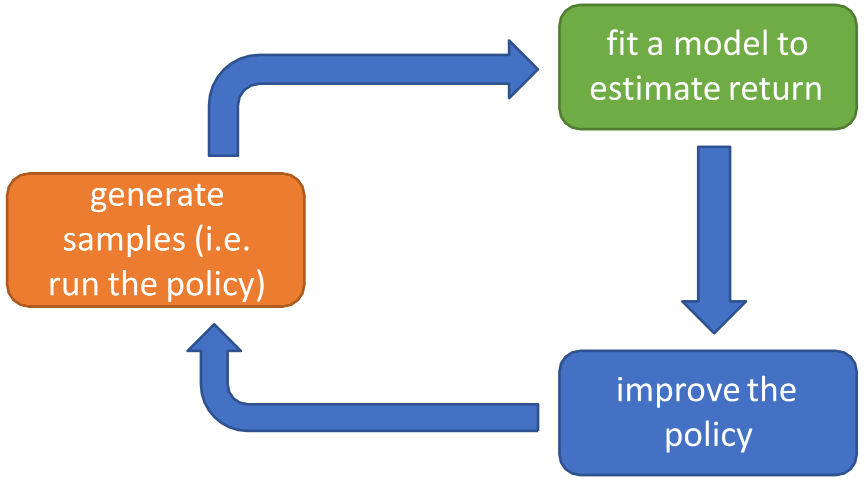
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

$K \times$

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

$Q_\phi(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')$


fit a model to estimate return
improve the policy
generate samples (i.e. run the policy)

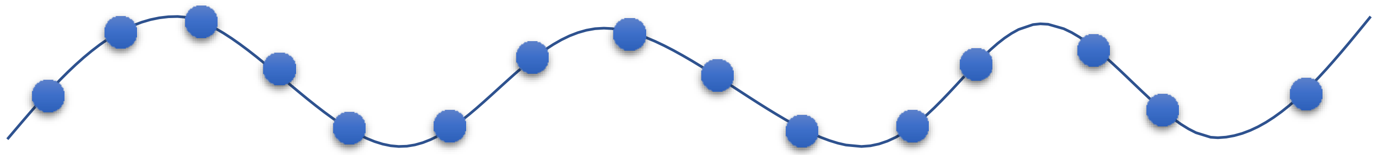
$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$

Problems in online Q-learning

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

- sequential states are strongly correlated
- target value is always changing



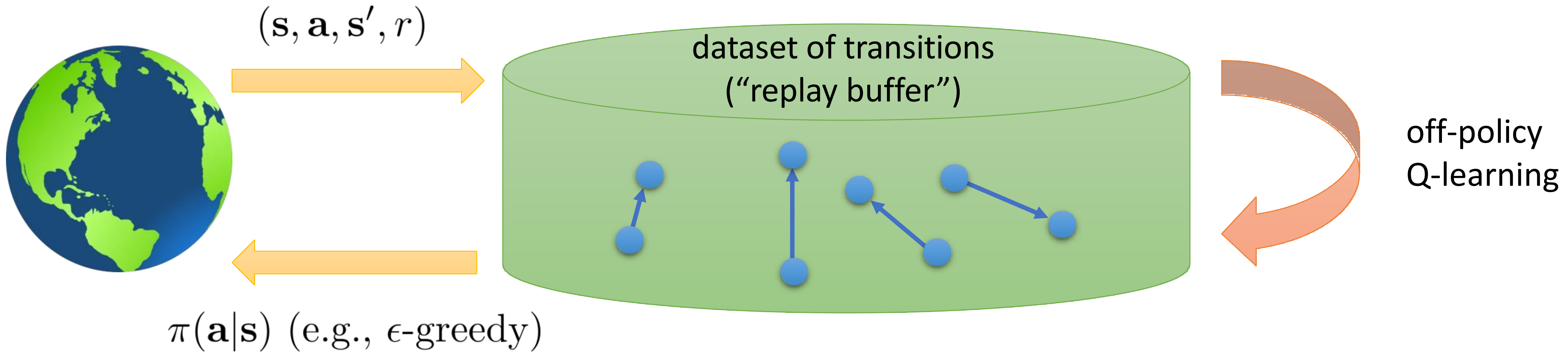
Solution: replay buffers

Q-learning with a replay buffer:

1. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
2. $\phi \leftarrow \phi - \alpha \underline{\sum_i} \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

+ samples are no longer correlated

+ multiple samples in the batch (low-variance gradient)


$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
dataset of transitions
("replay buffer")
off-policy
Q-learning
$\pi(\mathbf{a}|\mathbf{s})$ (e.g., ϵ-greedy)

Target Networks

What's wrong?

online Q iteration algorithm:

```
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$
```

~~these are correlated!~~

use replay buffer

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$$

no gradient through target value

This is still a problem!

Q-Learning and Regression

full Q-learning with replay buffer:

```
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
$K\times$
    2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
    3. $\phi \leftarrow \phi - \alpha \underline{\sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])}$
```

one gradient step, moving target

full fitted Q-iteration algorithm:

```
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
$K\times$
    2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
    3. set $\underline{\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2}$
```

perfectly well-defined, stable regression

Q-Learning with target networks

```
Q-learning with replay buffer and target network:
$N\times$  1. save target network parameters: $\phi' \leftarrow \phi$
               2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
    $K\times$      3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
                   4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \underline{[r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)]})$
```

targets don't change in inner loop!

supervised regression

“Classic” deep Q-learning algorithm (DQN)

Q-learning with replay buffer and target network:

```
1. save target network parameters: $\phi' \leftarrow \phi$
   $N\times$
   2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
      $K\times$
      3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
      4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$
```

“classic” deep Q-learning algorithm:

```
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update $\phi'$: copy $\phi$ every $N$ steps
```

$\}$ $K = 1$

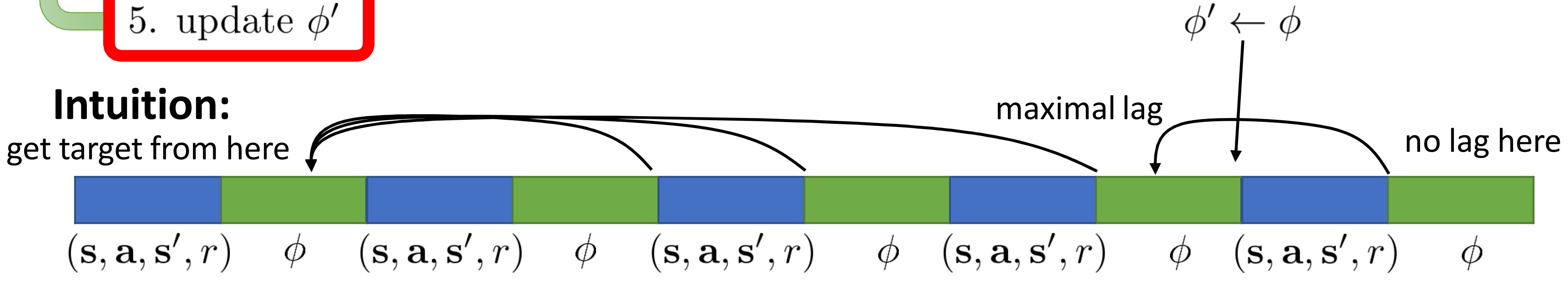
Mnih et al. ‘13

Alternative target network

"classic" deep Q-learning algorithm:

```
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update $\phi'$
```

Intuition:


get target from here
maximal lag
no lag here
$\phi' \leftarrow \phi$
$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
ϕ
$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
ϕ
$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
ϕ
$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
ϕ
$(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
ϕ

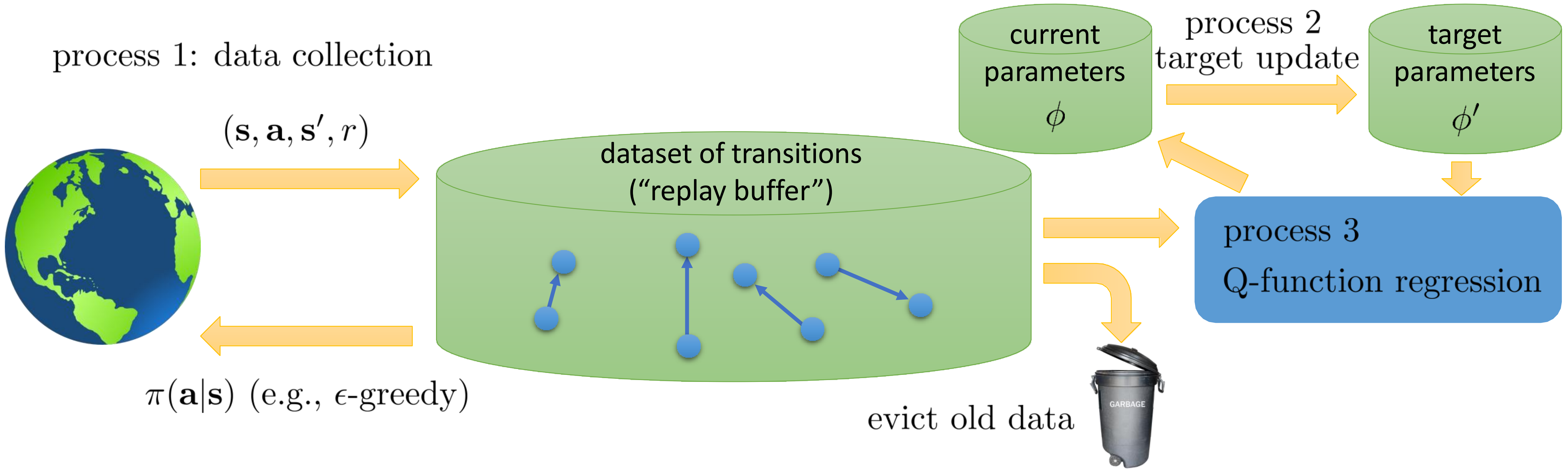
Feels weirdly uneven, can we always have the same lag?

Popular alternative (similar to Polyak averaging):

5. update ϕ': $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$ $\qquad$ $\tau = 0.999$ works well

A General View of Q-Learning

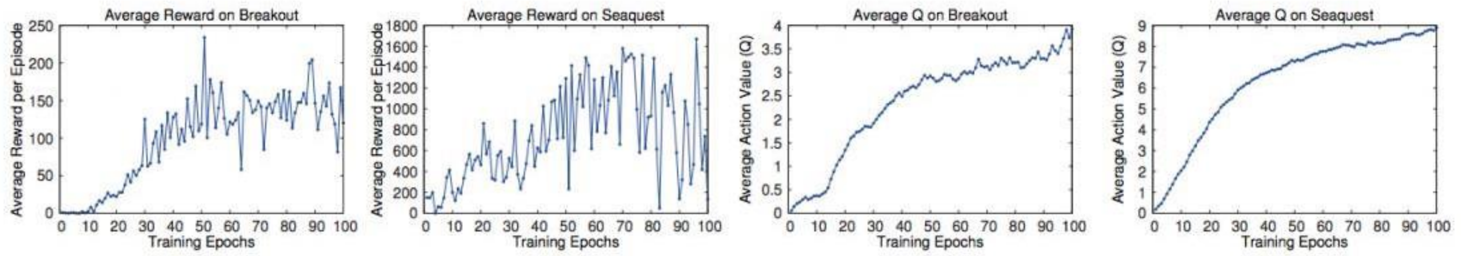
A more general view



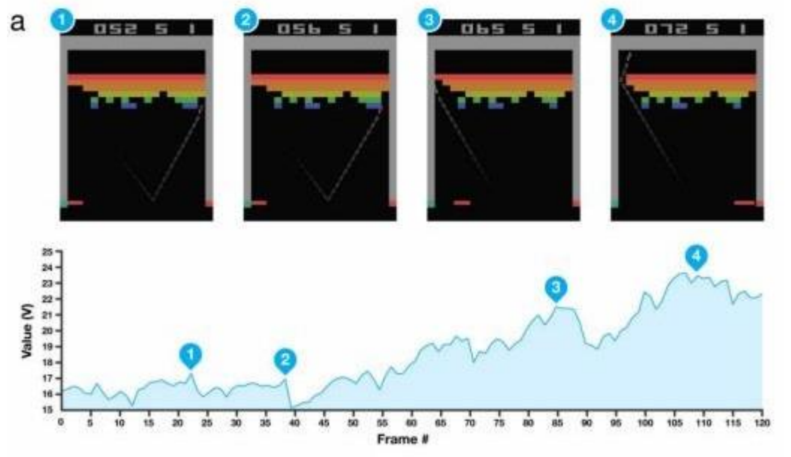
- Online Q-learning: evict immediately, process 1, process 2, and process 3 all run at the same speed
- DQN: process 1 and process 3 run at the same speed, process 2 is slow
- Fitted Q-iteration: process 3 in the inner loop of process 2, which is in the inner loop of process 1

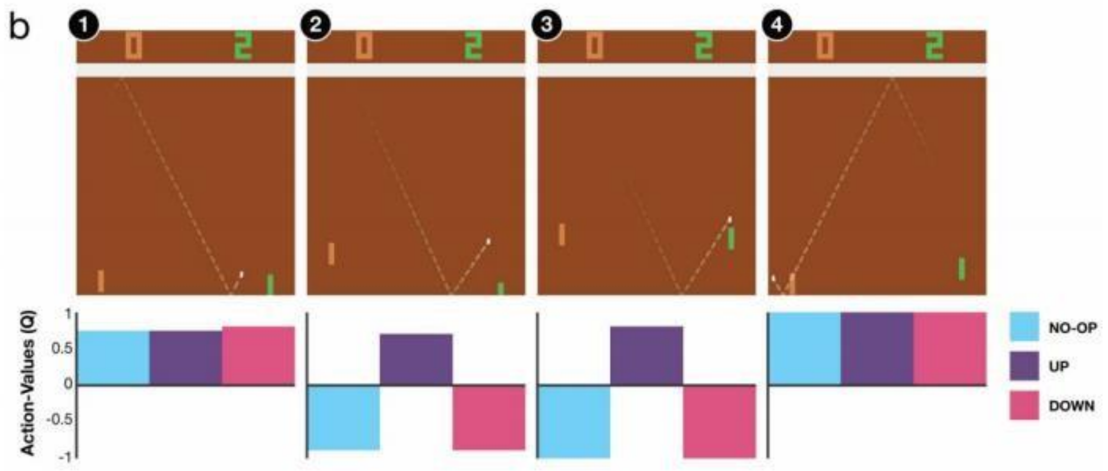
Improving Q-Learning

Are the Q-values accurate?

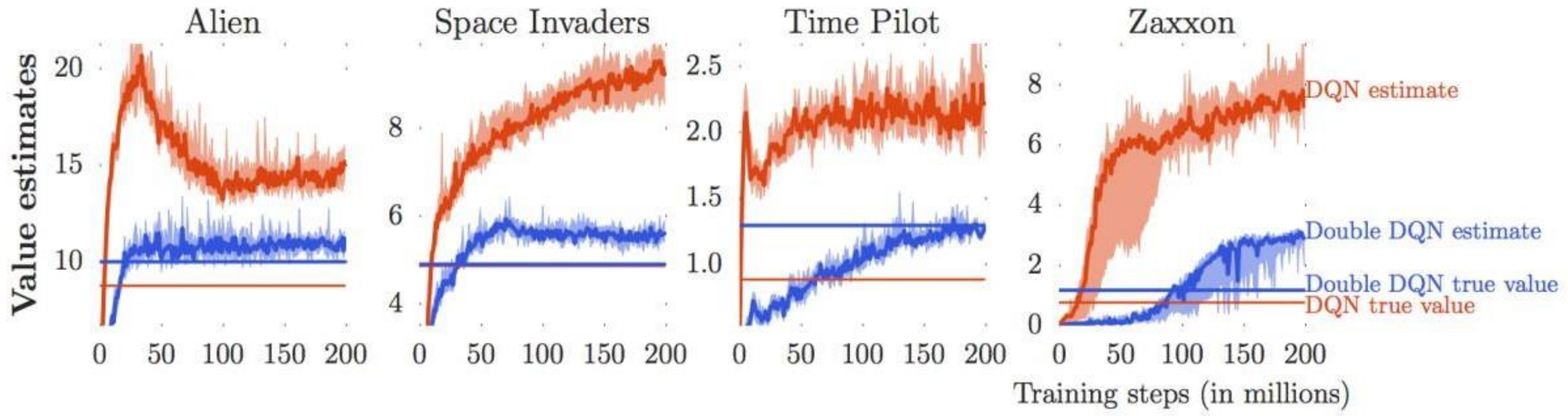


As predicted Q increases, so does the return





Are the Q-values accurate?



Overestimation in Q-learning

target value $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$

$\nwarrow$ this last term is the problem

imagine we have two random variables: X_1 and X_2

$E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$

$Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ is not perfect – it looks "noisy"

hence $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ *overestimates* the next value!

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underbrace{Q_{\phi'}}_{\text{value } also \text{ comes from } Q_{\phi'}}(\mathbf{s}', \underbrace{\arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')}_{\text{action selected according to } Q_{\phi'}})$

Double Q-learning

$E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underline{Q_{\phi'}}(\mathbf{s}', \underline{\arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')})$

value *also* comes from $Q_{\phi'}$ action selected according to $Q_{\phi'}$

if the noise in these is decorrelated, the problem goes away!

idea: don't use the same network to choose the action and evaluate value!

"double" Q-learning: use two networks:

$$Q_{\phi_A}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_B}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_A}(\mathbf{s}', \mathbf{a}'))$$

$$Q_{\phi_B}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_A}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_B}(\mathbf{s}', \mathbf{a}'))$$

if the two Q's are noisy in *different* ways, there is no problem

Double Q-learning in practice

where to get two Q-functions?

just use the current and target networks!

standard Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

double Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi}(\mathbf{s}', \mathbf{a}'))$

just use current network (not target network) to evaluate action

still use target network to evaluate value!

Multi-step returns

Q-learning target: $y_{j,t} = r_{j,t} + \gamma \max_{\mathbf{a}_{j,t+1}} Q_{\phi'}(\mathbf{s}_{j,t+1}, \mathbf{a}_{j,t+1})$

these are the only values that matter if $Q_{\phi'}$ is bad!

these values are important if $Q_{\phi'}$ is good

where does the signal come from?

Q-learning does this: max bias, min variance

remember this?

Actor-critic: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{i,t}) \right)$

+ lower variance (due to critic)

- not unbiased (if the critic is not perfect)

Policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$

+ no bias

- higher variance (because single-sample estimate)

can we construct multi-step targets, like in actor-critic?

$$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$$

N-step return estimator

Q-learning with N-step returns

$$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$$

this is supposed to estimate $Q^\pi(\mathbf{s}_{j,t}, \mathbf{a}_{j,t})$ for π

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

+ less biased target values when Q-values are inaccurate

+ typically faster learning, especially early on

- only actually correct when learning on-policy

why?

we need transitions $\mathbf{s}_{j,t'}, \mathbf{a}_{j,t'}, \mathbf{s}_{j,t'+1}$ to come from π for $t' - t < N - 1$

(not an issue when $N = 1$)

how to fix?

- ignore the problem
  - often works very well
- cut the trace – dynamically choose N to get only on-policy data
  - works well when data mostly on-policy, and action space is small
- importance sampling

For more details, see: “Safe and efficient off-policy reinforcement learning.” Munos et al. ‘16

Q-Learning with Continuous Actions

Q-learning with continuous actions

What's the problem with continuous actions?

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \boxed{\arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t)} \\ 0 \text{ otherwise} \end{cases}$$

this max

$$\text{target value } y_j = r_j + \gamma \boxed{\max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)}$$

this max
particularly problematic (inner loop of training)

How do we perform the max?

Option 1: optimization

- gradient based optimization (e.g., SGD) a bit slow in the inner loop
- action space typically low-dimensional – what about stochastic optimization?

Q-learning with stochastic optimization

Simple solution:

$$\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) \approx \max \{Q(\mathbf{s}, \mathbf{a}_1), \ldots, Q(\mathbf{s}, \mathbf{a}_N)\}$$

$(\mathbf{a}_1, \ldots, \mathbf{a}_N)$ sampled from some distribution (e.g., uniform)

+ dead simple

+ efficiently parallelizable

- not very accurate

but… do we care? How good does the target need to be anyway?

More accurate solution:

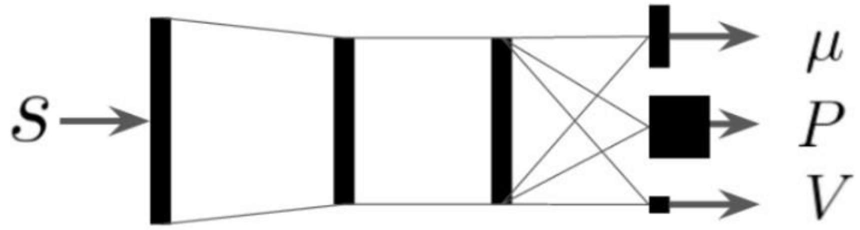
- cross-entropy method (CEM)
  - simple iterative stochastic optimization
- CMA-ES
  - substantially less simple iterative stochastic optimization

works OK, for up to about 40 dimensions

Easily maximizable Q-functions

Option 2: use function class that is easy to optimize

$$Q_\phi(\mathbf{s},\mathbf{a}) = -\frac{1}{2}(\mathbf{a}-\mu_\phi(\mathbf{s}))^T P_\phi(\mathbf{s})(\mathbf{a}-\mu_\phi(\mathbf{s})) + V_\phi(\mathbf{s})$$



NAF Architecture.

NAF: **N**ormalized **A**dvantage **F**unctions

$$\arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = \mu_\phi(\mathbf{s}) \qquad \max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = V_\phi(\mathbf{s})$$

+ no change to algorithm

+ just as efficient as Q-learning

- loses representational power

Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG (Lillicrap et al., ICLR 2016)

“deterministic” actor-critic
(really approximate Q-learning)

$\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = Q_\phi(\mathbf{s}, \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}))$

idea: train another network $\mu_\theta(\mathbf{s})$ such that $\mu_\theta(\mathbf{s}) \approx \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a})$

how? just solve $\theta \leftarrow \arg\max_\theta Q_\phi(\mathbf{s}, \mu_\theta(\mathbf{s}))$ $\qquad \dfrac{dQ_\phi}{d\theta} = \dfrac{d\mathbf{a}}{d\theta}\dfrac{dQ_\phi}{d\mathbf{a}}$

new target $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_\theta(\mathbf{s}'_j)) \approx r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j))$

Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG:

```
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$ using *target* nets $Q_{\phi'}$ and $\mu_{\theta'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j) \frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mu(\mathbf{s}_j))$
6. update $\phi'$ and $\theta'$ (e.g., Polyak averaging)
```

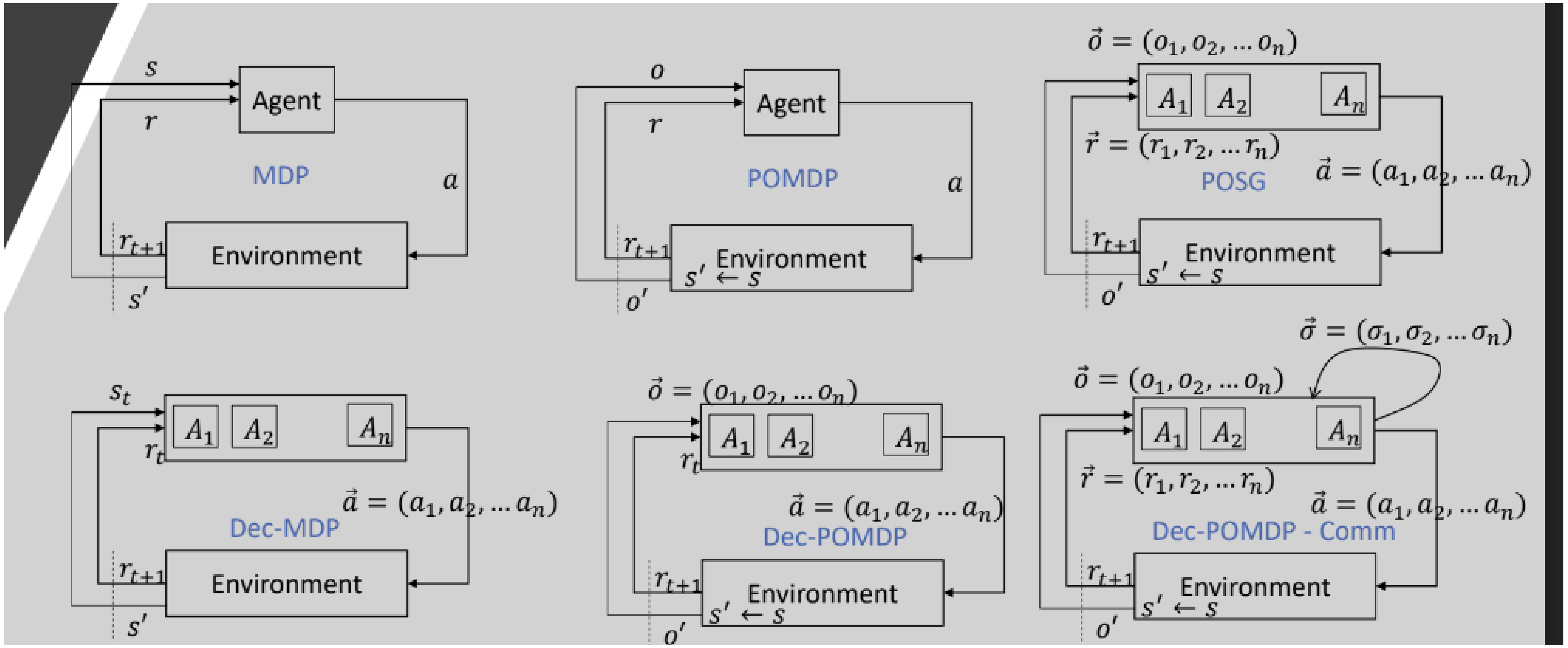
Multi-Agent RL

Policy Learning in a Multi-Agent Setting

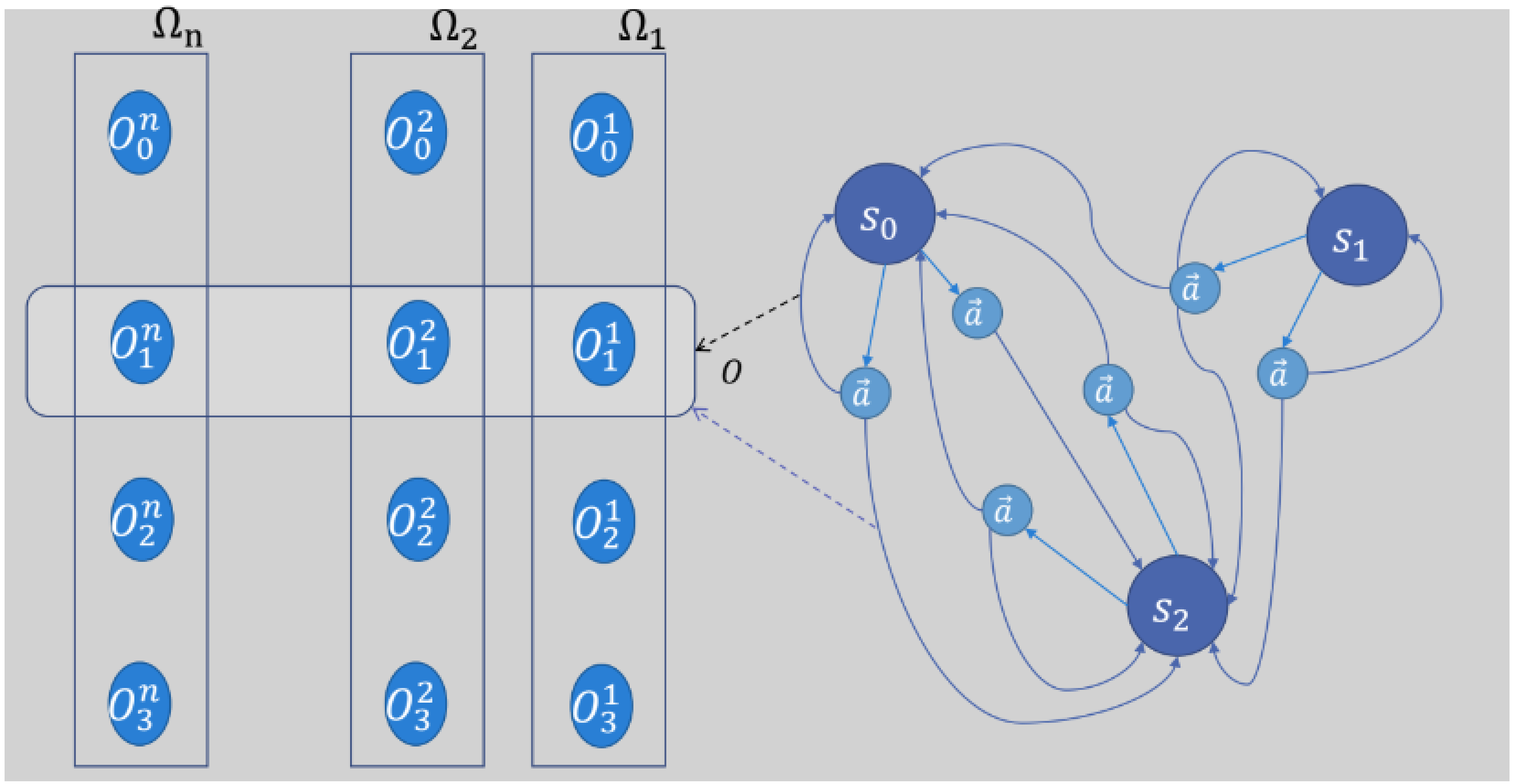
- This is the extension of the concept of learning to an environment where there are multiple agents
- Each agent is trying to learn, each agent is imperfect in the beginning

Swarm Robotics Example

Variations of Multi-Agent MDPs



Decentralized POMDPs-(Dec-POMDP)



Decentralized POMDPs- (Dec-POMDP)

- Now while we have n agents taking actions to receive a single reward, each agent receives a different partial observation
- Ω_i : finite set of observations available to agent i
- Ω : set of joint observations $\Omega = \Omega_1 \times \Omega_2 \cdots \times \Omega_n$
- The observation function O is now defined between state transitions and joint action
- $O: A \times S \to \Delta \ \Omega$
- Hence Dec-POMDP is defined by the tuple $S, \ A_i \ , P, R, O, \ \Omega \ , I$

Game Theoretical Formulations of MARL

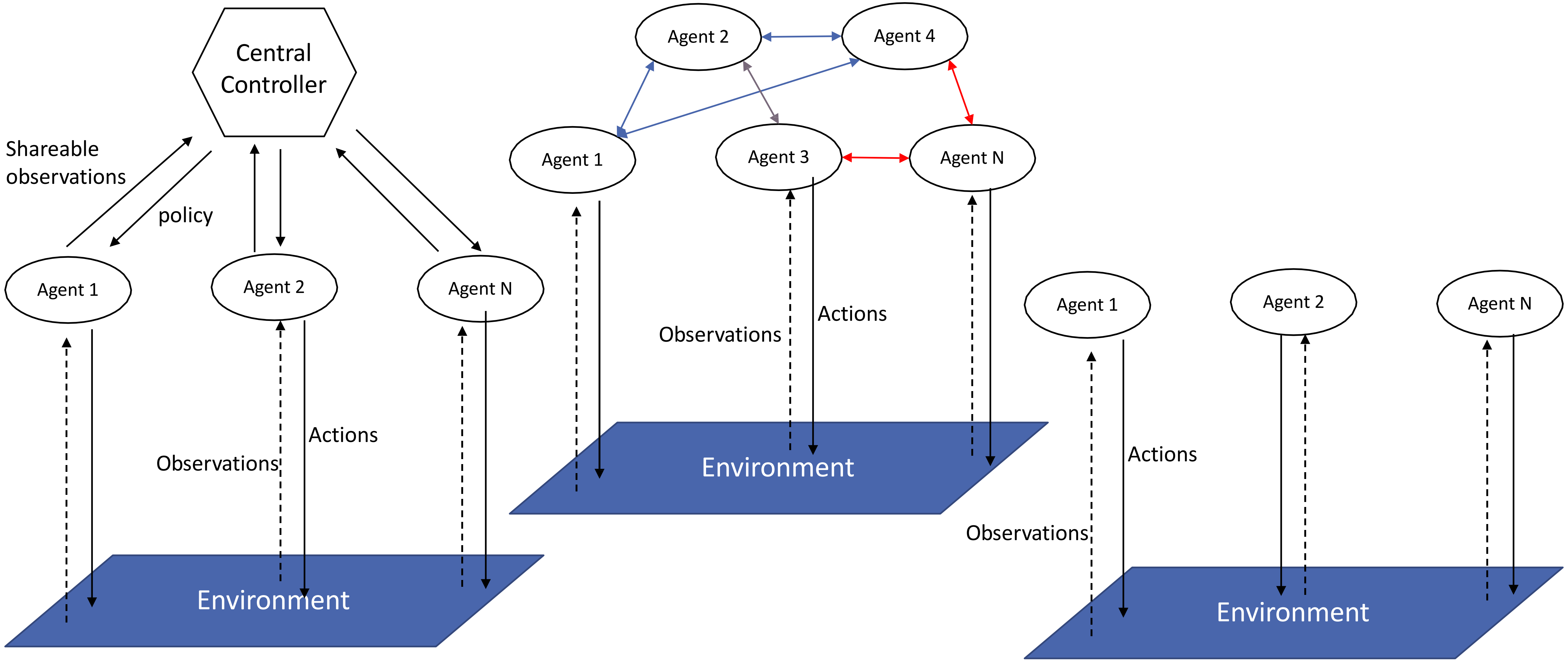
- There are two different but closely related theoretical frameworks for MARL
  - Markov Games – Also known as stochastic games
    - Cooperative Setting: All agents receive the same reward
    - Competitive setting: zero sum game, E.g. reward of one agent is the loss of other
    - Mixed setting: general sum game, each agent is self interested, and rewards may conflict with each other
  - Extensive form games
    - Better at handling imperfect information, such as partial observability

- Different solutions are proposed for these settings where the Nash Equilibrium is found
  - NE – joint policy learnt
  - By definition, NE characterizes the point that no agent will deviate from, if any algorithm finally converges.

Nash Equilibrium as a solution concept

- NE Is a reasonable solution concept in game theory, under the assumption that the agents are all rational, and are capable of perfectly reasoning and infinite mutual modelling of agents.
- However, with bounded rationality, the agents may only be able to perform finite mutual modelling
- As a result, the learning dynamics that are devised to converge to NE may not be justifiable for practical MARL agents.

Learning structures

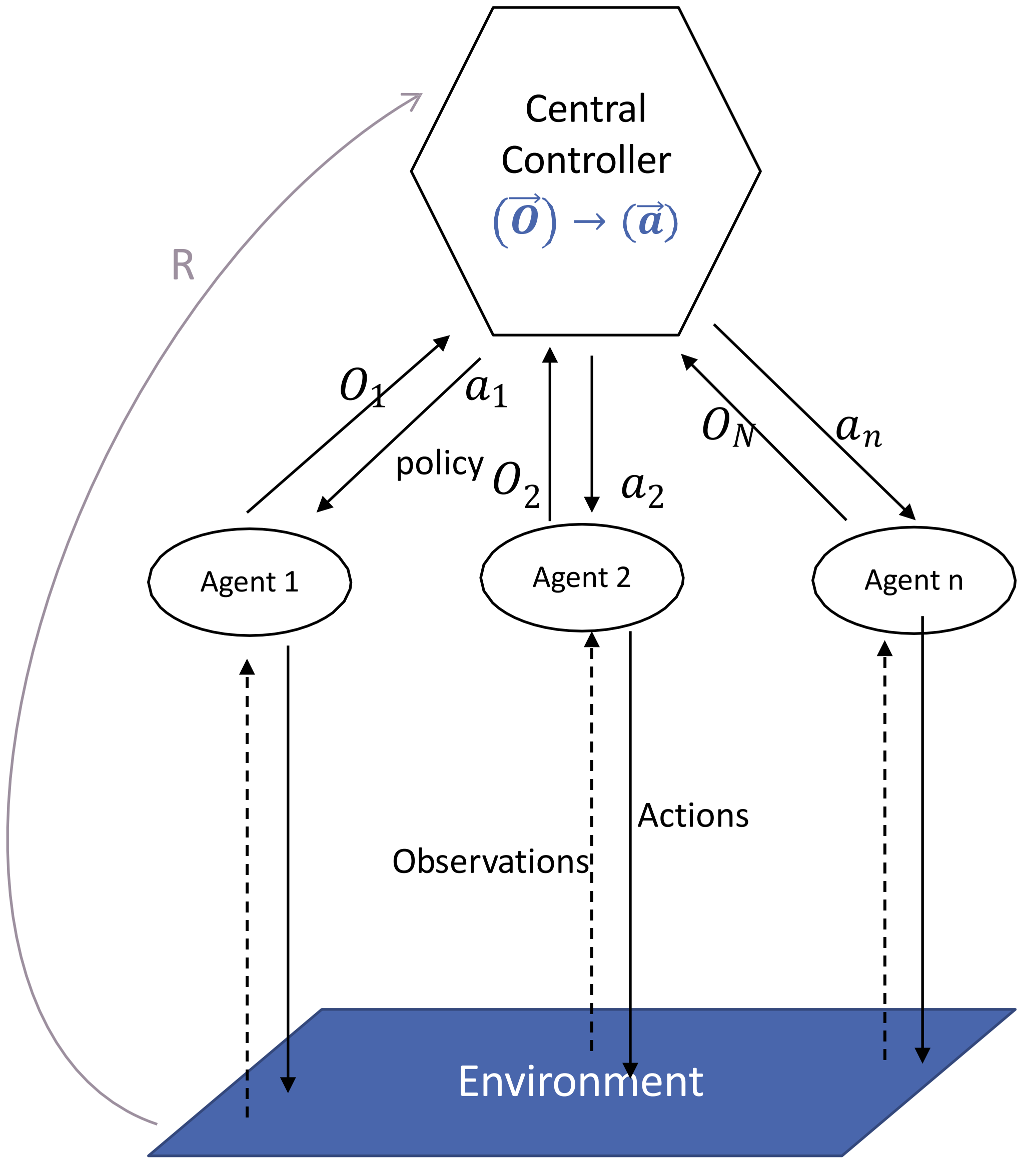


Zhang et al., Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms

Joint Action Learning

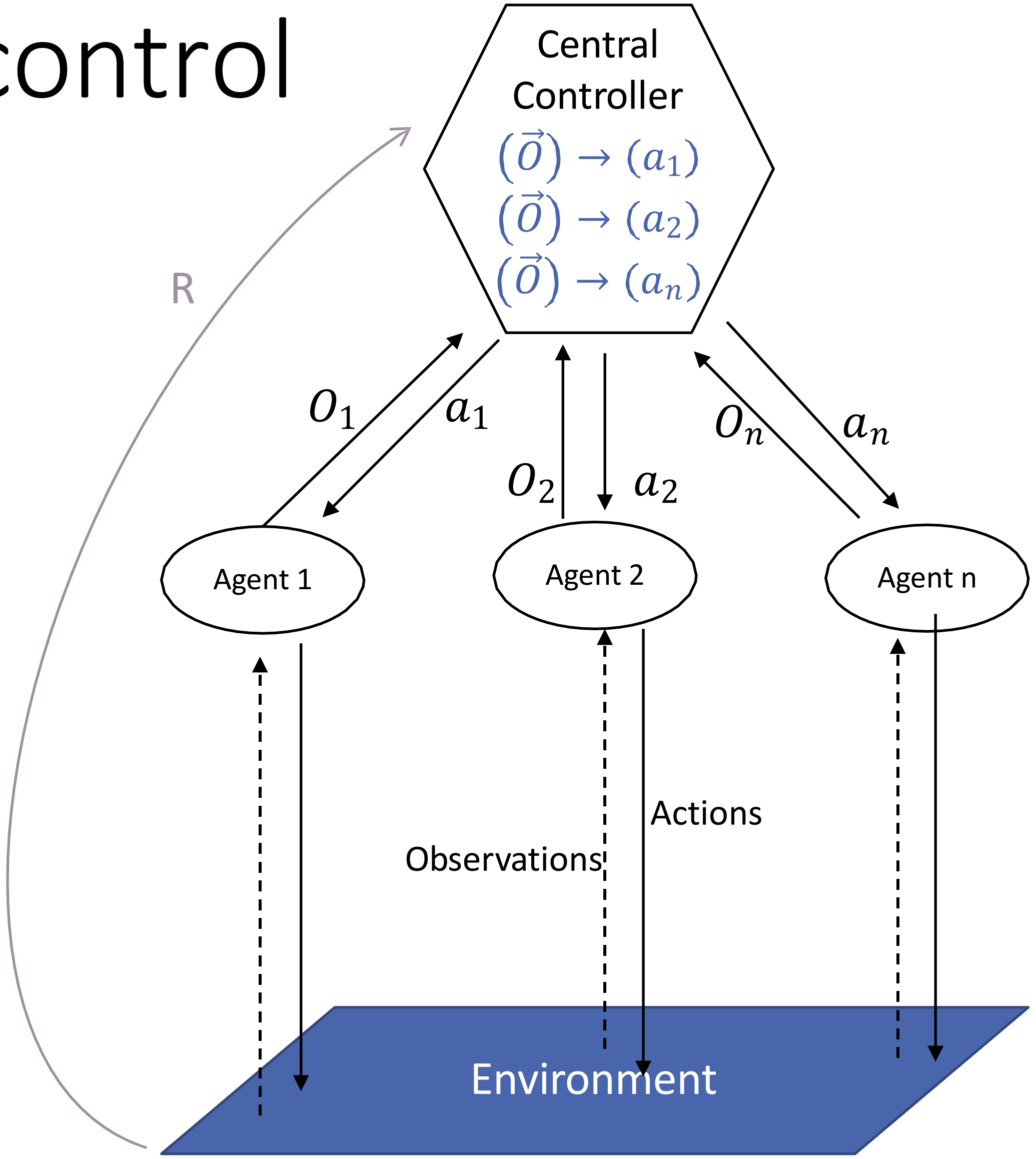
- Joint observation of all agents are mapped to a joint action of all agents
  - $(O_1, O_2 \dots, O_N) \rightarrow (A_1, A_2, \dots., A_N)$

- Centralized in both training and execution
  - leads to an exponential growth in the observation and actions spaces with the number of agents



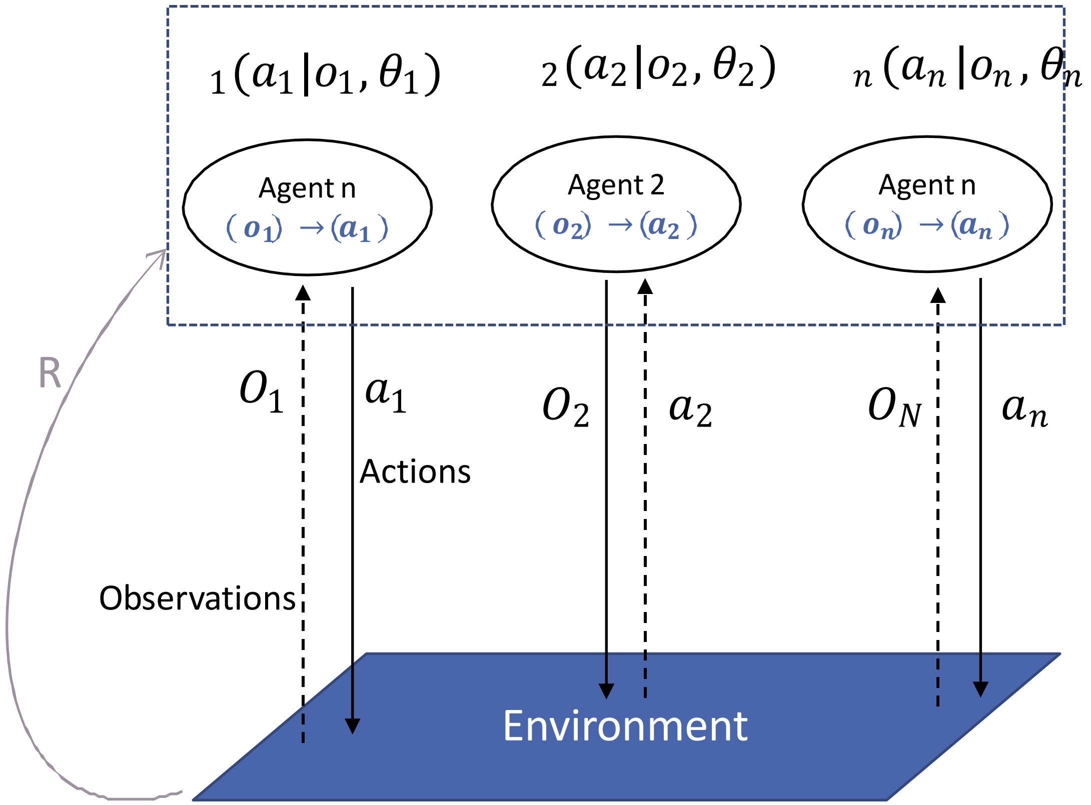
Fully factored centralized control

- Assume that the joint action (a) can be factored into individual components (a_i) for each agent
- Now the joint observation is mapped to the action of individual agents using a set of independent sub-policies
  - $P(a) = ς^n \; P(a_i)$
- This reduces the Action space from $|A|^n$ to $|A|n$
- Observation space still keeps growing and hence such approaches aren't suitable for complex environments nor large number of agents



Independent (Dec/Concurrent) Learning

- Each agent learns its own policy, without any knowledge about others, but with a joint reward function
  - $O_i \rightarrow A$
  - An agent considers other agents' and their actions as part of the environment dynamics
- This is helpful when agents learn heterogeneous policies
- Training does not scale as agents increase
  - Agents don't share experience, therefore sample complexity increases
  - High computational and memory requirements
- Learning in such an environment is non-stationary
  - As each agent evolves over time -> lack of convergence guarantees/ cant use experience replay



(Gupta et al., 2017): Paper

Questions?

Tomorrow:

1. Diving into differences in competition and collaboration.
2. Deep RL to solve multi-agent tasks.

Tutorial Outline

1. Introduction
   - Policy learning in Multi-Agent Settings
   - Potential Real World Applications
   - Basics of reinforcement learning

2. Recent Developments in Multi-Agent Reinforcement Learning
   - Centralized training and decentralized execution
   - Communication based MA learning
   - Scaling MA learning: Curriculum learning, Selective Attention, Evolutionary methods

3. **Challenges and Applications of Data driven Multi-Agent Learning**
   - **MA Imitation learning in sports analytics**
   - **Applications in driverless vehicle control**
   - **Using Agent based simulations to assist MA learning**

4. Experimental platforms and Conclusions

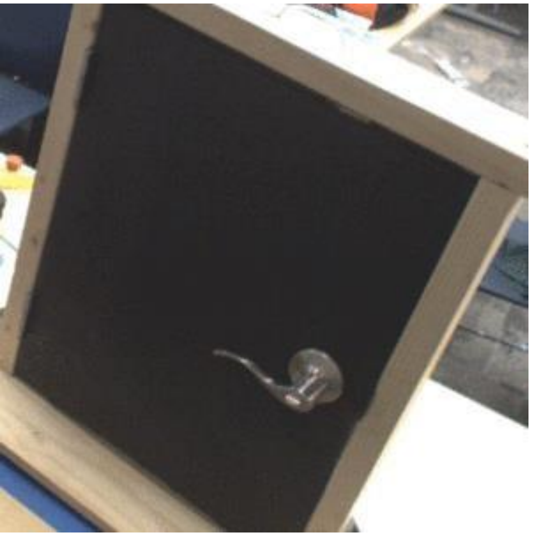
Imitation Learning

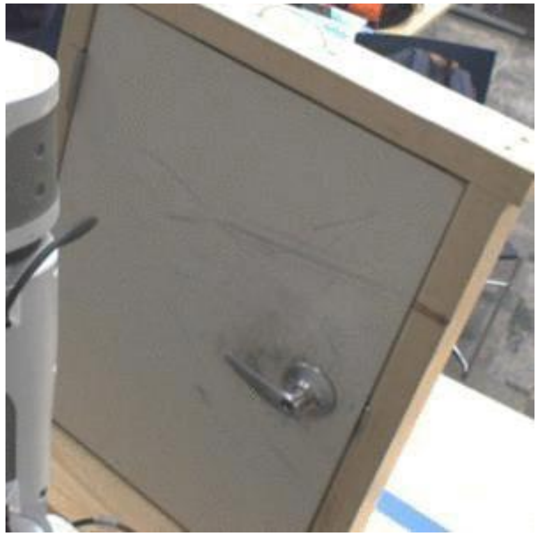
- Many real-world problems cannot be framed as reinforcement learning

- Imitation learning is an alternative concept to allow a robot (computer) to learn to perform a task by observing demonstrations from an expert

- Two types of imitation learning
  - Behavioural cloning
  - Inverse reinforcement learning

Imitation Learning





Multi-Agent Imitation Learning

- Growing availability of real world multi-agent data sources enables the possibility of learning MA decision making policies from data
  - E.g. Sports tracking: Learning team behaviour
  - E.g. Vehicle tracking: learning autonomous vehicle control policies

- Learning MA policies from expert demonstrations is known Multi-Agent Imitation Learning

Tian et al., Use of Machine Learning to Automate the Identification of Basketball Strategies Using Whole Team Player Tracking Data: Paper

MA Inverse Reinforcement Learning

- IRL assumes that the expert is optimizing an underlying reward function, and attempts to recover this reward function that rationalizes the demonstrations

Example: Artificial Intelligence for Player Behaviour Modelling

Coordinated Imitation Learning for building Artificial player models

- Many real-world multi-agent applications cannot be framed as reinforcement learning problem
  - The reward function used, determines the emergent agent behaviour

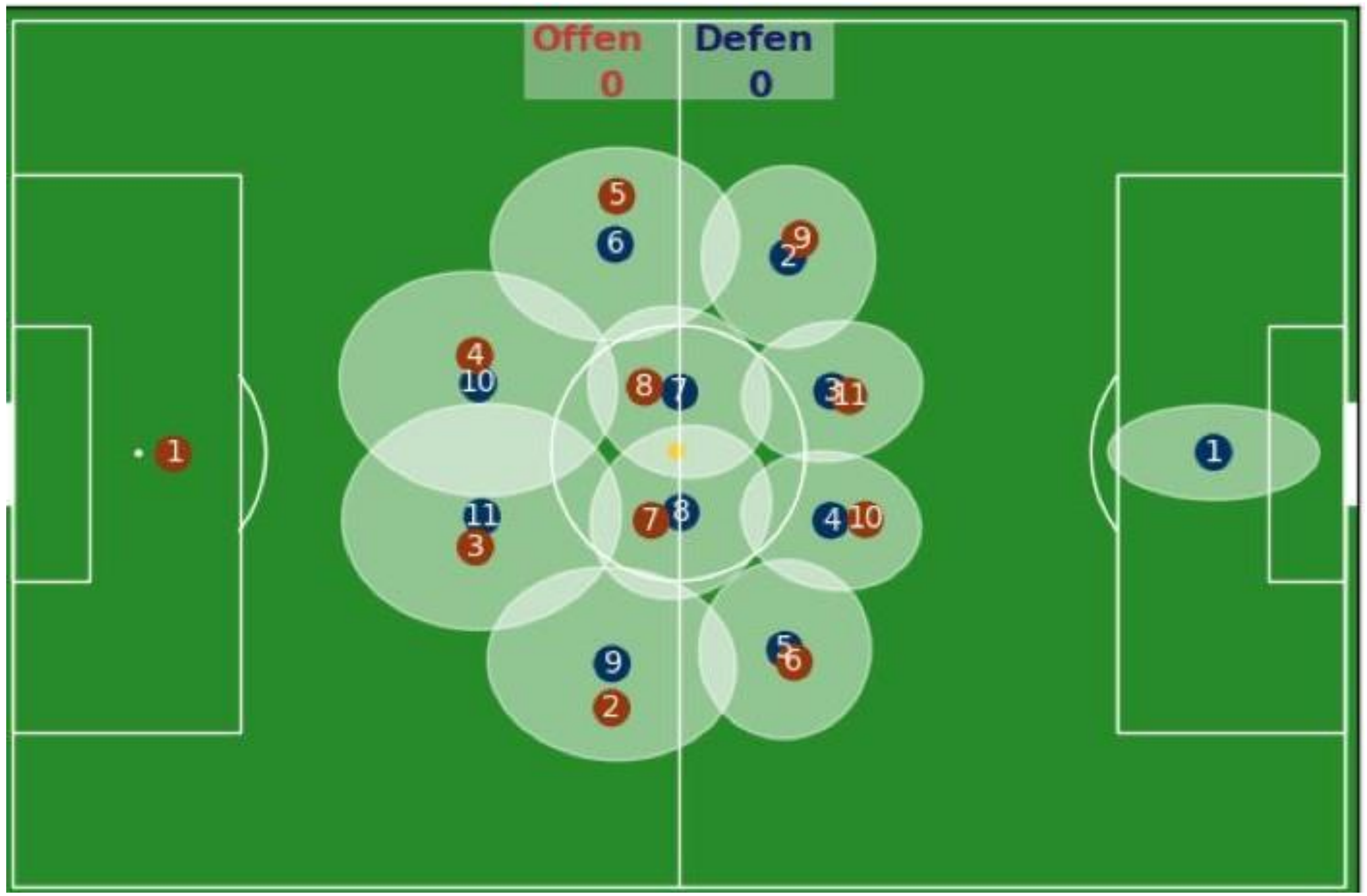
- In a scenario of learning from player tracking data in football there are many contextual challenges
  - It is very difficult to specify a suitable reward function: goals are a rarity
  - Goal probability model that captures various contexts is required for optimization
  - Different players must coordinate their behaviour according to the game situation
  - Player roles may change during the match, hence the policy assumed by an expert is not fixed
  - The actions of a player very high dimensional: especially considering the body movements associated with the actions

Le et al., Data-Driven Ghosting using Deep Imitation Learning: Paper

Data-Driven Ghosting

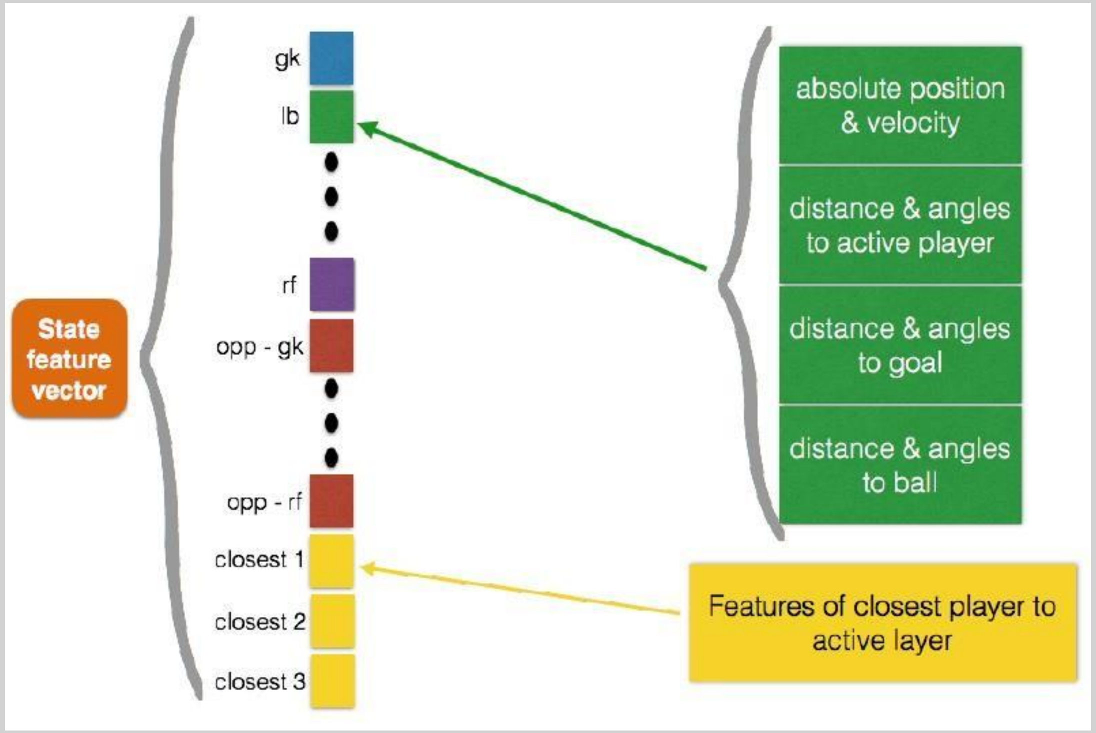
- Data Used and Pre-processing
  - 100 games of player tracking and event data from a recent professional soccer league
  - interested in modeling defensive situations, only focused on sequences of play where the opposition had control of the ball.
  - A defensive sequence is terminated when a goal is scored against the defending team, the ball gets out of the pitch, dead-ball events occur (e.g. foul, offside), or the defensive team regains possession of the ball.
  - In total, there were approximately 17400 sequences of attacking-defending situations
    - (~3 million frames at 10 frames per second).
    - The average length of all sequences is approximately 170 frames, or 17 seconds.

Pre-Processing: role alignment



Feature Engineering

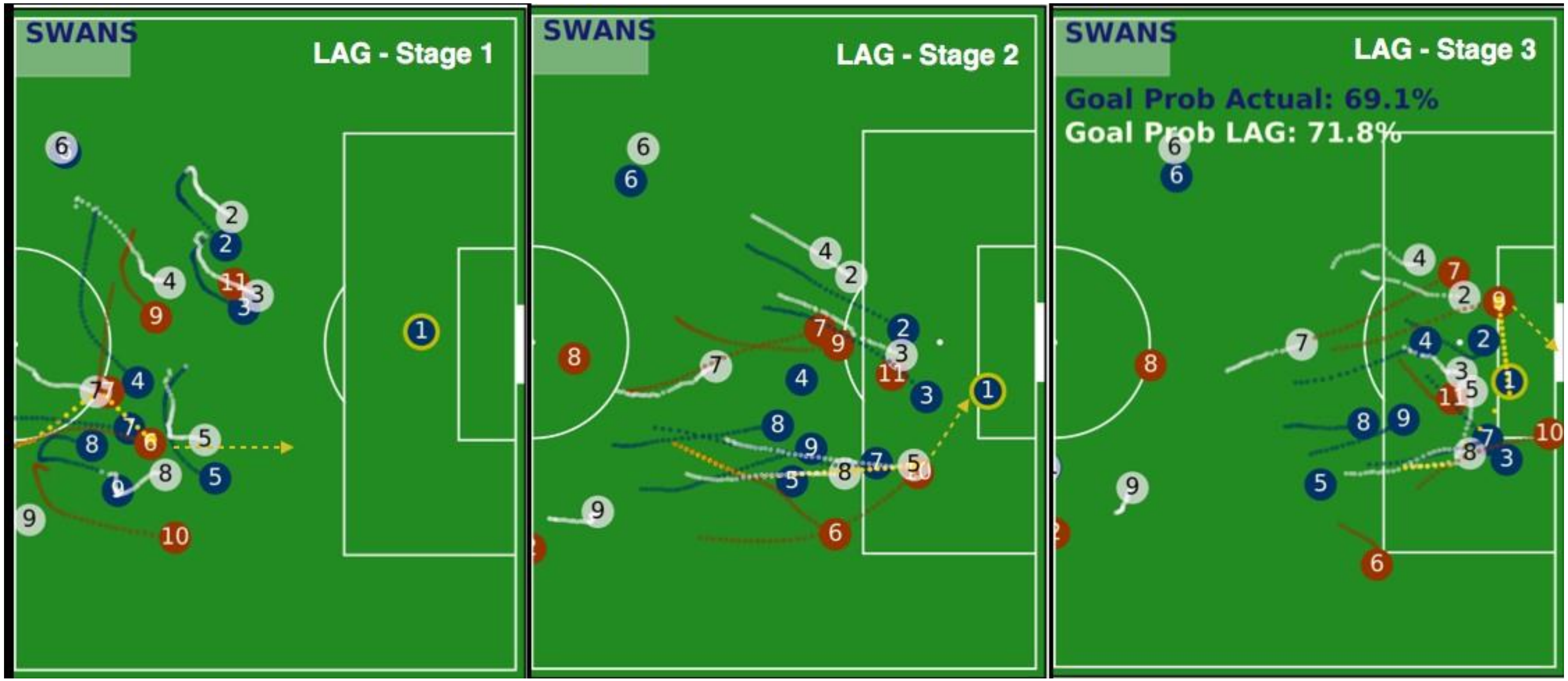
- Input feature contains three aspects
  - Absolute coordinates of players and ball
  - Relative polar coordinates of each player towards the ball and the goal
  - Role of the player


State feature vector
gk
lb
rf
opp - gk
opp - rf
closest 1
closest 2
closest 3
absolute position & velocity
distance & angles to active player
distance & angles to goal
distance & angles to ball
Features of closest player to active layer

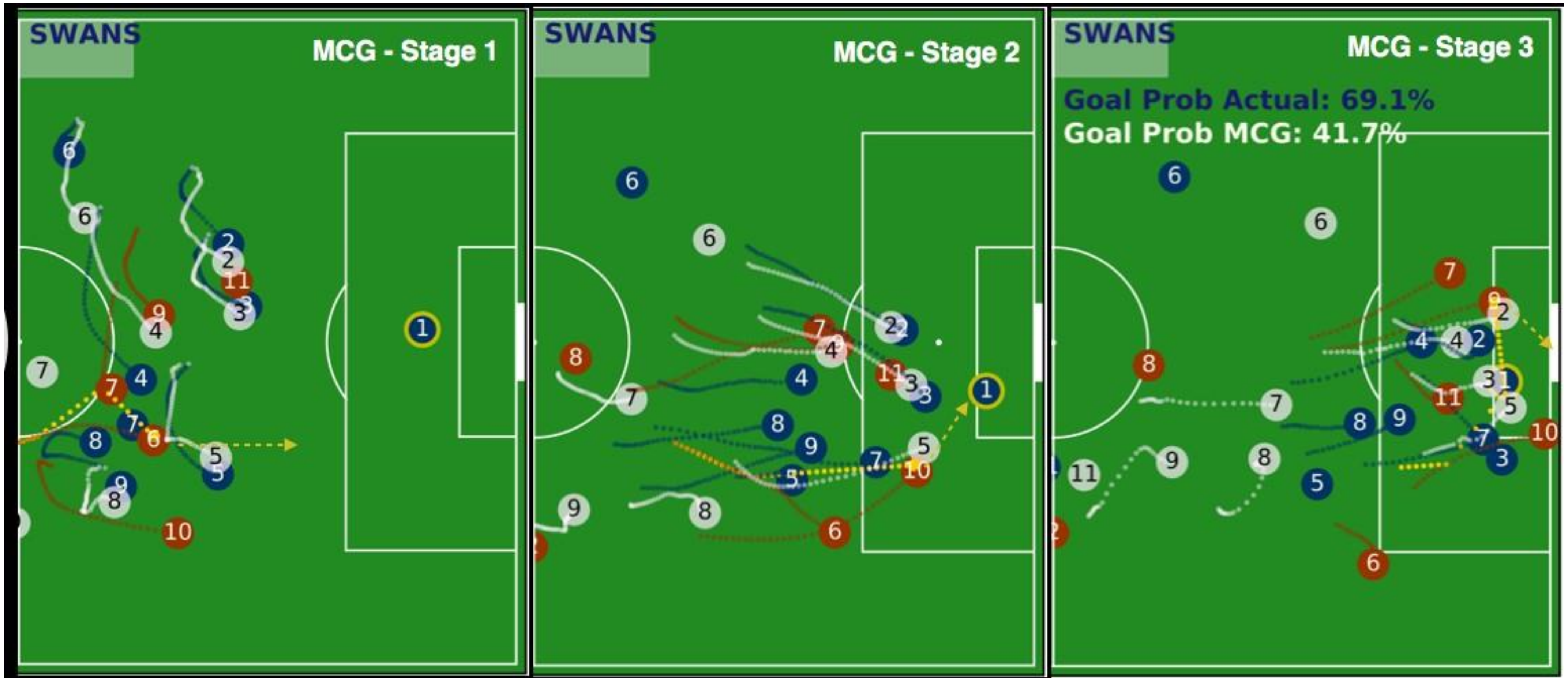
Model Building

- Objective: Predict the next position of the player (that is been modelled)
- A particular type of recurrent neural networks called Long Short-Term Memory (LSTM) was used due to its powerful ability to capture long-range dependencies in sequential data
- The model takes in a sequence of input feature vectors as described above and the corresponding sequence of each player’s positions as output labels.
-
- Each player is modelled by an LSTM, which consists of two hidden layers of networks with 512 hidden units in each layer.
- The role of these hidden units is to capture the information from the recent history of actions from all players and map the information to the position of the next time step

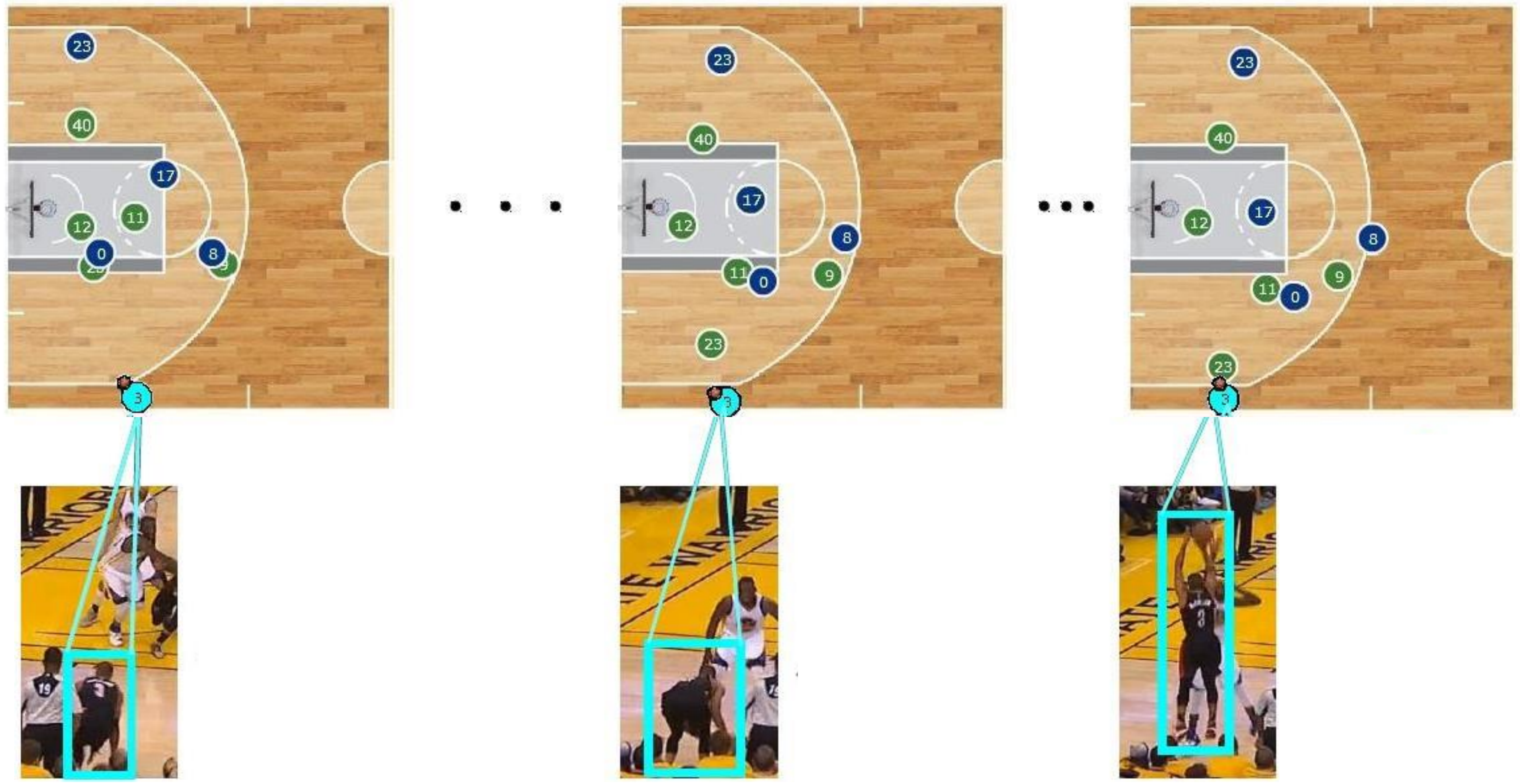
Comparing teams: league average



Comparing teams: Man City Ghosts



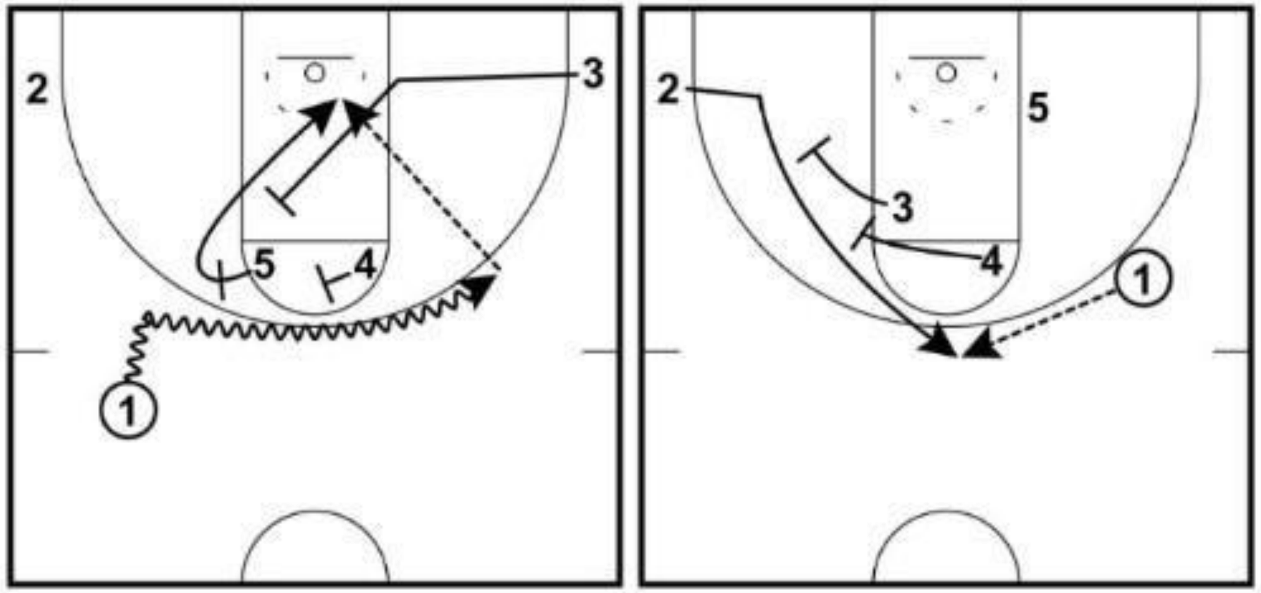
Analysing Shooting Styles in the NBA using Body Pose



http://www.sloansportsconference.com/wp-content/uploads/2017/02/1690.pdf

Play Sketching

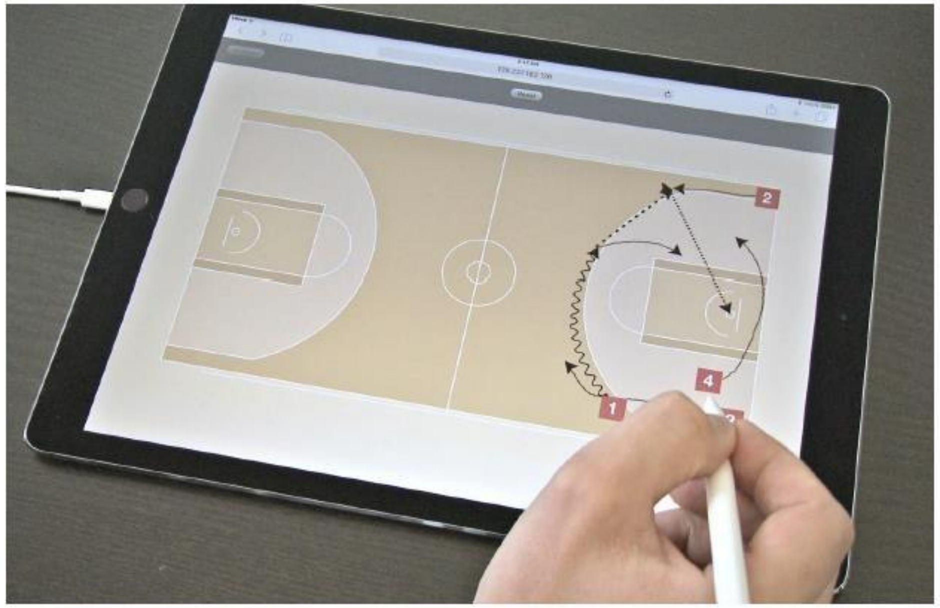
- Selecting and implementing effective basketball plays is one of the most important roles of a basketball coach.
- The ability to call the right play at the right time during a game can often be the difference between your team winning a losing



https://www.basketballforcoaches.com/basketball-plays/

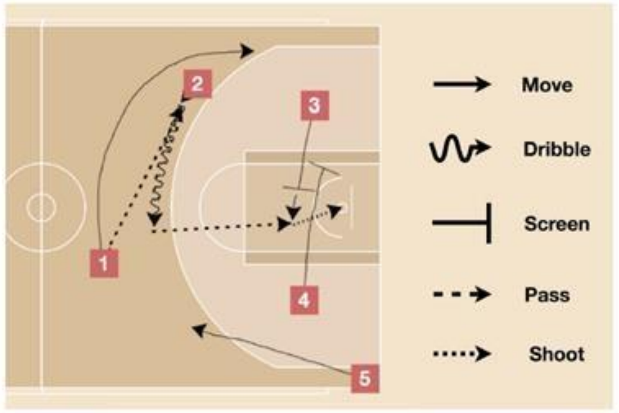
Interactive play sketching

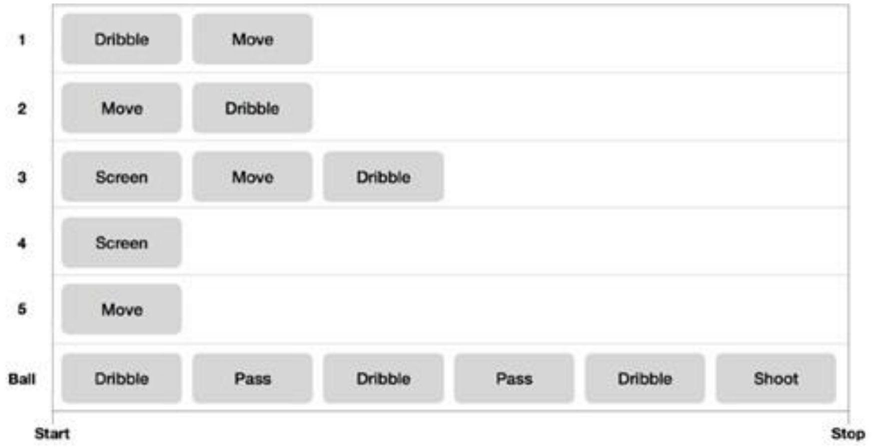
- Step 1 . Simulate the tracking data that would be recorded if players carried out the sketched play
- Step 2. Use data driven ghosting to see how a specific team will react in return towards the sketched play
    - Input to the data driven ghosting is now partly sketched data (e.g. attacking side)
    - Ghosts of other team are visualized
- Step 2: Run through an expected point model to predict the goodness of the sketch
- It is supposed to run in real time.



Seidl et al., Bhostgusters: Realtime Interactive Play Sketching with Synthesized NBA Defenses: Paper

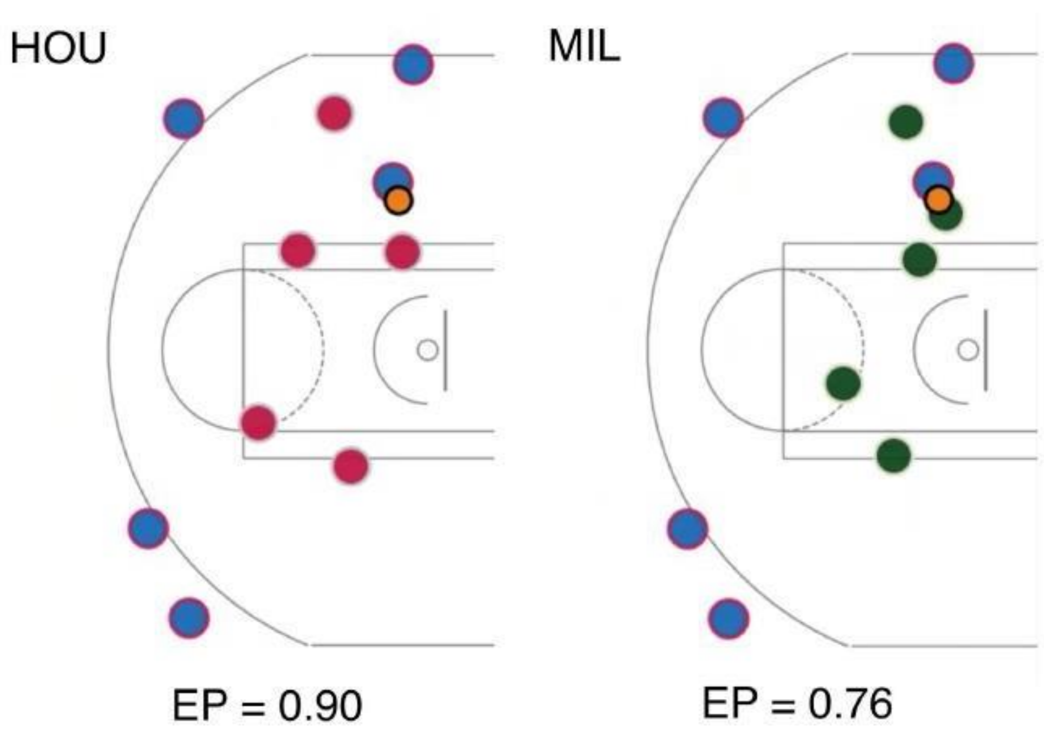
Play Sketching





1	Dribble	Move				
2	Move	Dribble				
3	Screen	Move	Dribble			
4	Screen					
5	Move					
Ball	Dribble	Pass	Dribble	Pass	Dribble	Shoot

Start Stop



Teaching Driverless Cars....

- Motorways / Country roads / or even urban roads with disciplined users are easy to deal with
- Real world environments could be really complex
- Intelligent mobility aspires beyond roads: pedestrianized areas / indoors, technology has got to be universal!

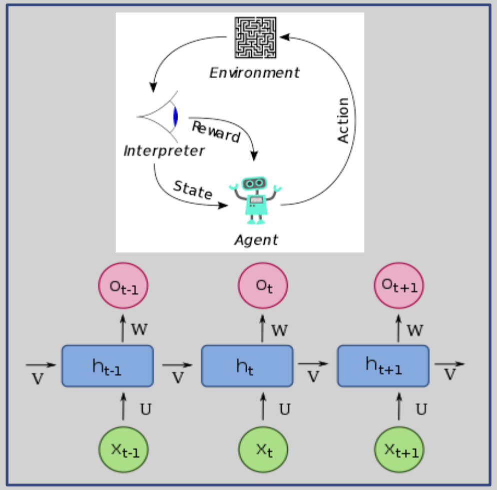
Infrastructure-led policy learning: Learning from Humans

- So how could connected autonomous vehicles make complex decisions in complicated environments?
- How could the vehicles be intelligent enough to make those decisions?
- Our benchmark of intelligence is “human intelligence”.
- Proposition: instead of collecting data from all possible scenarios that a machine will face, lets learn at key intersections, how humans negotiate.

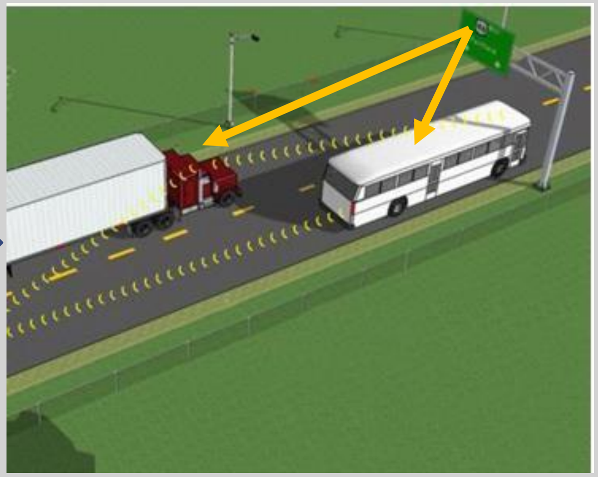
Infrastructure-led policy learning: Learning from Humans

Sensing Infrastructure

Scene Perception and data selection



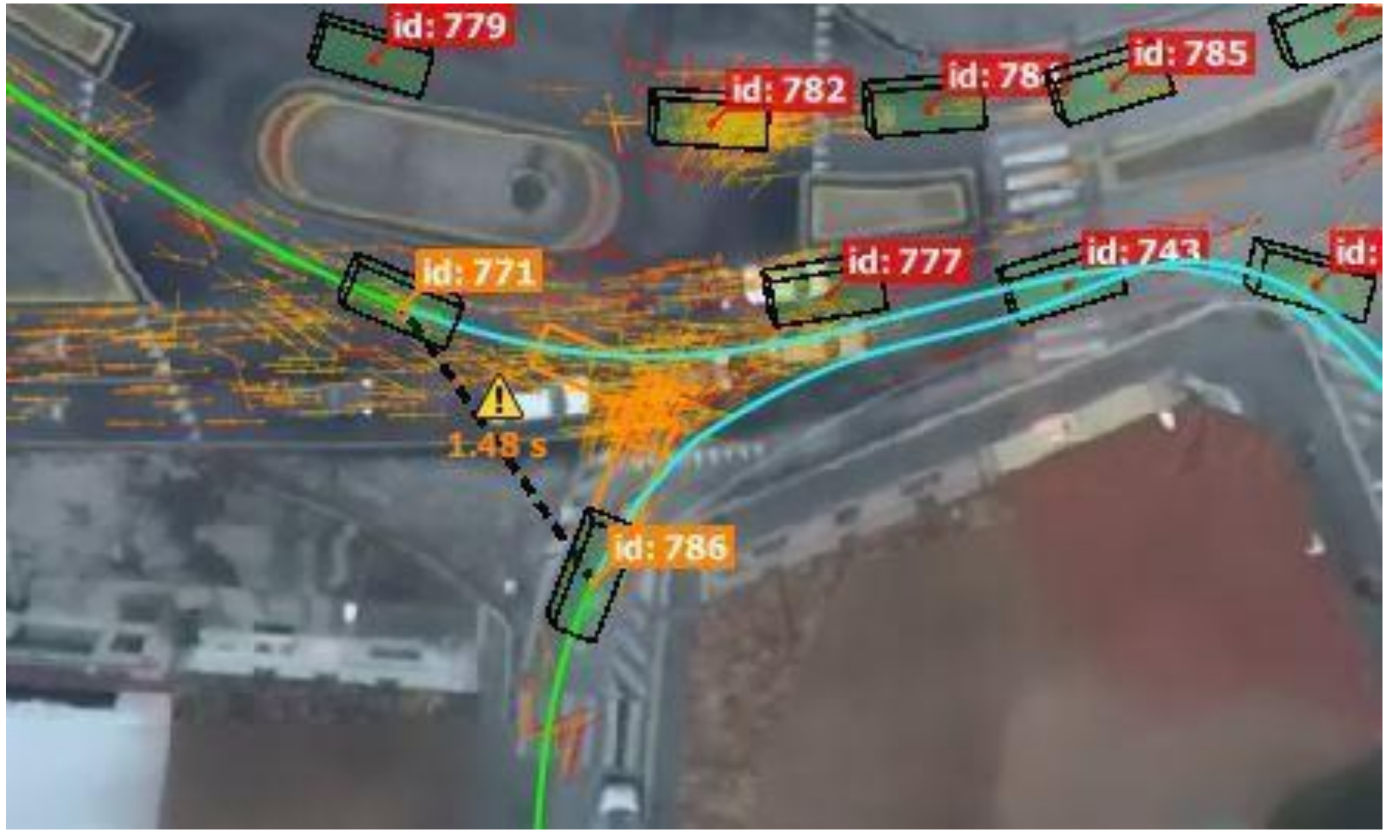
Data-driven driving policy learning



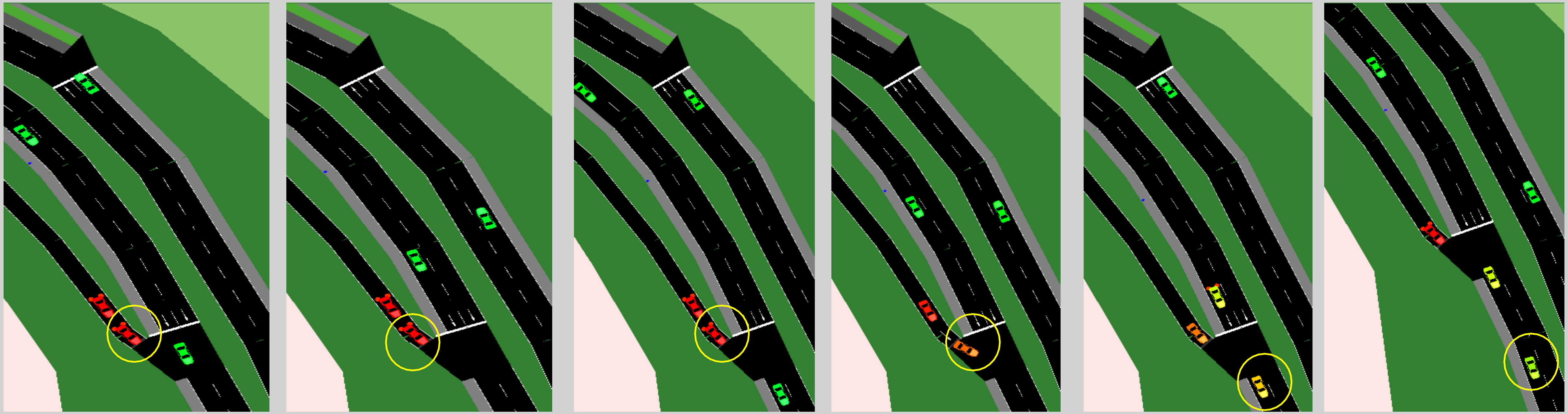
Application Layer

Inder et al., Learning Control Policies of Driverless Vehicles from UAV Video Streams in Complex Urban Environments: Paper

UAV Sensing to Learn Driving Policy Locally at a Junction



Example: A Vehicle Joining a Motorway



Learning from Experts to measure human driving performance

- Multi-agent imitation learning enables us to learn computational model of vehicle manoeuvring
- This model can be used to embed autonomous vehicles with a suitable driving policy
- Furthermore, this model can be utilized to measure how a given human driver handles a situation- for insurance purposes / urban modelling

Bridging the gap between gaming environments and real-world settings

Collision-damage can be mitigated by training vehicles in a mixed reality setup

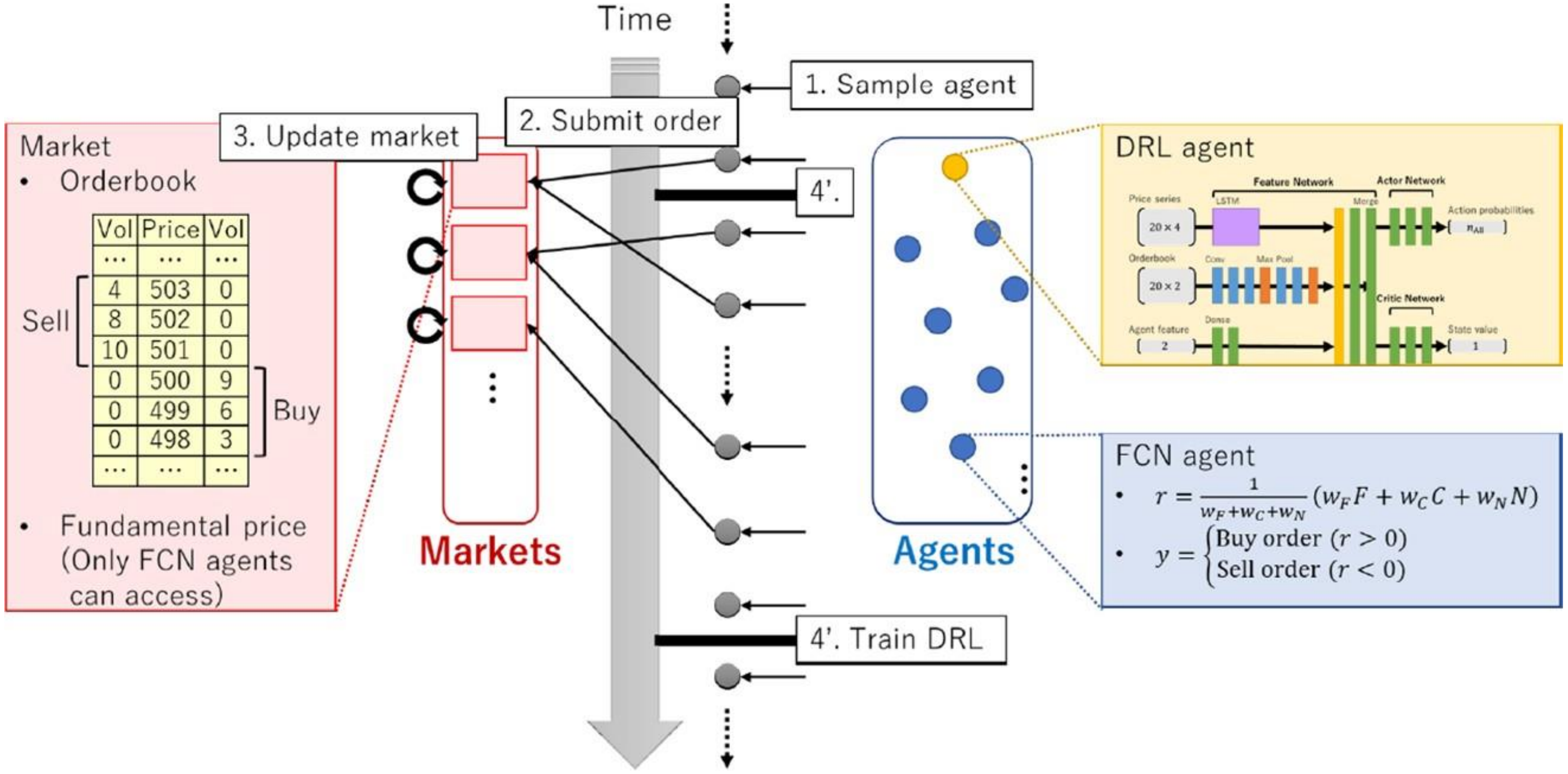
Using Agent Based Models for learning useful policies

- Sometime the past data can be a poor predicter of the future
  - Especially in the cases of financial markets

- Hence past data alone may not be adequate
  - COVID-19 related market crashes
  - Brexit day jump of the currency markets

- How could we improve predictive ability of RL/IL agents?
  - Agent based simulation is a well established field. E.g. Artificial markets
  - Simulations can be adapted to create realistic, but never realized scenarios
  - Simulations can also benefit from advanced stochastic models

Combination of RL and Agent Based Models: Example from Finance



Vol	Price	Vol
⋯	⋯	⋯
4	503	0
8	502	0
10	501	0
0	500	9
0	499	6
0	498	3
⋯	⋯	⋯

- $r = \frac{1}{w_F + w_C + w_N}(w_F F + w_C C + w_N N)$
- $y = \begin{cases} \text{Buy order } (r > 0) \\ \text{Sell order } (r < 0) \end{cases}$

Maeda et al., Deep Reinforcement Learning in Agent Based Financial Market Simulation: Paper

Challenges in using MARL for Game Intelligence

UK MULTI-AGENT SYSTEMS SYMPOSIUM

SAM DEVLIN

MICROSOFT RESEARCH

Tutorial Outline

Experimental Platforms

- OpenAI Gym
  - A good source to start experimenting with RL https://gym.openai.com/
  - There are different multi-agent extensions compatible with openAI Gym: https://github.com/koulanurag/ma-gym

- DeepMind OpenSpiel
  - Collection of environments and algorithms, strong on computational game theory
  - Paper: https://arxiv.org/pdf/1908.09453.pdf
  - https://github.com/deepmind/open_spiel

- Google Research Football Environment
  - The Environment offers a game engine, a set of research problems called Football Benchmarks and Football Academy
  - https://github.com/google-research/football

Experimental Platforms

- Multi-Agent Driving Simulator (Description)
  - Based on the popular TORCS platform, compatible wity OpenAI Gym
  - multiple cars running simultaneously on a track can be controlled by different control algorithms
  - https://github.com/abhisheknaik96/MultiAgentTORCS

- Competitive Multi-Agent Environments (Description)
  - From OpenAI: MuJoCo environments
  - Experimentations with self play, transfer learning
  - Paper: https://arxiv.org/pdf/1710.03748.pdf
  - https://github.com/openai/multiagent-competition

Multi-Agent Driving Simulator: Based on TORCS

Google Football Experimental Testbed

03:41
RBA 0
FRQ 0
Train from pixels or internal game state
github.com/google-research/football

Further Resources

- DL/RL summer School : The relationship to game theory
  - Michael Bowling: https://www.youtube.com/watch?v=p_n5fF8apiE&t=3947s
  - James Wright: https://www.youtube.com/watch?v=MEUdtwQev9A

- Dimitri Bertsekas: Distributed and Multiagent Reinforcement learning
  - https://www.youtube.com/watch?v=nTPuL6iVuwU

- Thore Graepel: Role of MA learning in AI @ DeepMind, the relationship to intelligence and social intelligence, link to game theory and social dilemmas
  - https://www.youtube.com/watch?v=CvL-KV3IBcM

Recent Survey/Tutorial Papers

- Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications [Link to Paper]
- Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms [Link to Paper]
- A Review of Cooperative Multi-Agent Deep Reinforcement Learning [Link to Paper]
- A Survey and Critique of Multiagent Deep Reinforcement Learning [Link to Paper]

Conclusions

- Intelligent agent design for multi-agent settings have many real world applications
- Reinforcement learning and imitation learning methods are two useful avenues that may allow us to design these agents by utilizing computational and big data resources
- Multi-agent policy learning is still in its adolescence, and scalability to meet real world application challenges is still being addressed
- It is a domain with true multi-disciplinary potential

Acknowledgements

- Engineering and Physical Sciences Research Council
  - EP/T000783/1: MIMIc: Multimodal Imitation Learning in Multi-Agent Environments
  - EP/S515140/1: Intuitive Learning: new paradigm in AI for decision making in intelligent mobility
- Industrial Sponsors
  - Chelsea Football Club Academy
  - Dallas Mavericks Basketball
  - Toyota Manufacturing UK
- My Team and Mentors
  - Rafael Moreira-Pina, Joosep Hook, Naomi White, Jamie Na, and Haylat Tibebu, Jamie Roche
  - Prof Ahmet Kondoz, Prof Mike Caine

Thank You!

Looking forward to discuss and collaborate with you all.

V.D.De-Silva@lboro.ac.uk

Implementation Tips and Examples

Simple practical tips for Q-learning

- Q-learning takes some care to stabilize
  - Test on easy, reliable tasks first, make sure your implementation is correct

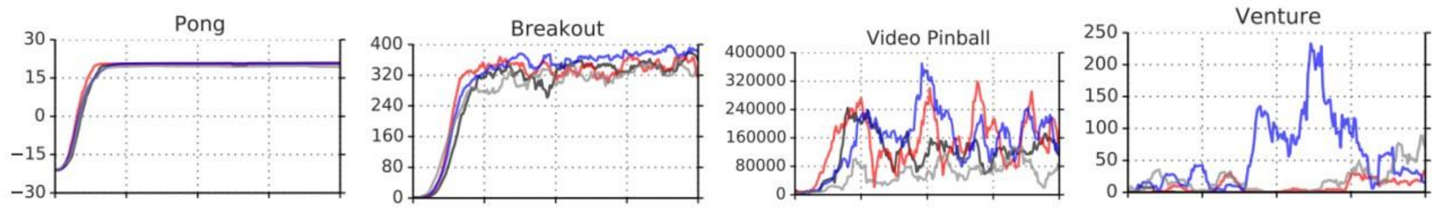


Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". *arXiv preprint arXiv:1511.05952* (2015), Figure 7

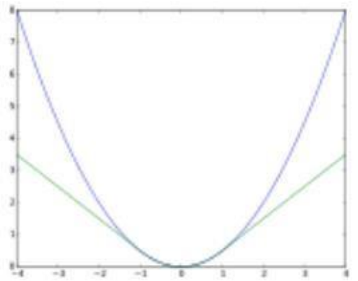
- Large replay buffers help improve stability
  - Looks more like fitted Q-iteration
- It takes time, be patient – might be no better than random for a while
- Start with high exploration (epsilon) and gradually reduce

Slide partly borrowed from J. Schulman

Advanced tips for Q-learning

- Bellman error gradients can be big; clip gradients or use Huber loss

$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \le \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$

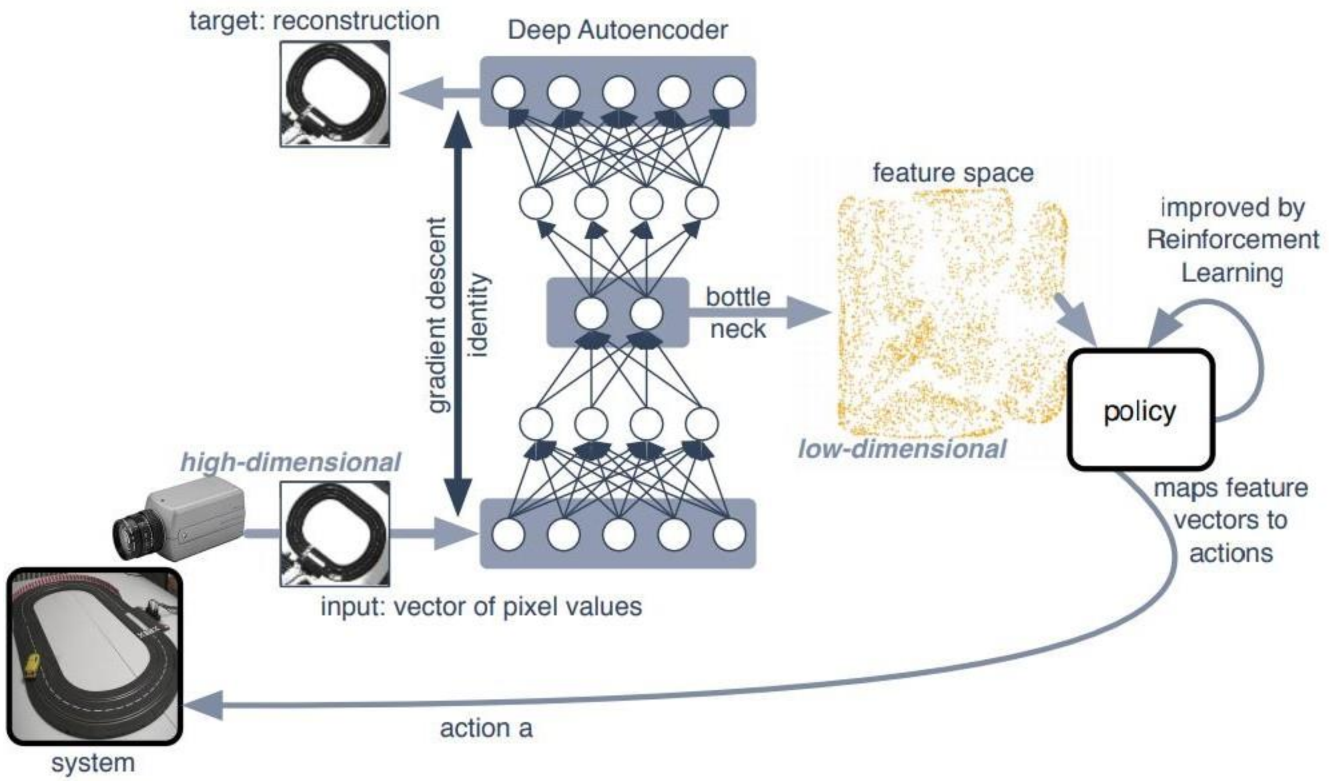


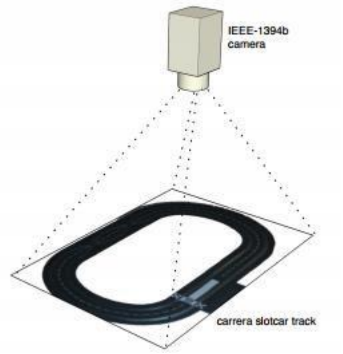
- Double Q-learning helps *a lot* in practice, simple and no downsides
- N-step returns also help a lot, but have some downsides
- Schedule exploration (high to low) and learning rates (high to low), Adam optimizer can help too
- Run multiple random seeds, it’s very inconsistent between runs

Slide partly borrowed from J. Schulman

Fitted Q-iteration in a latent space

- "Autonomous reinforcement learning from raw visual data," Lange & Riedmiller '12
- Q-learning on top of latent space learned with autoencoder
- Uses fitted Q-iteration
- Extra random trees for function approximation (but neural net for embedding)



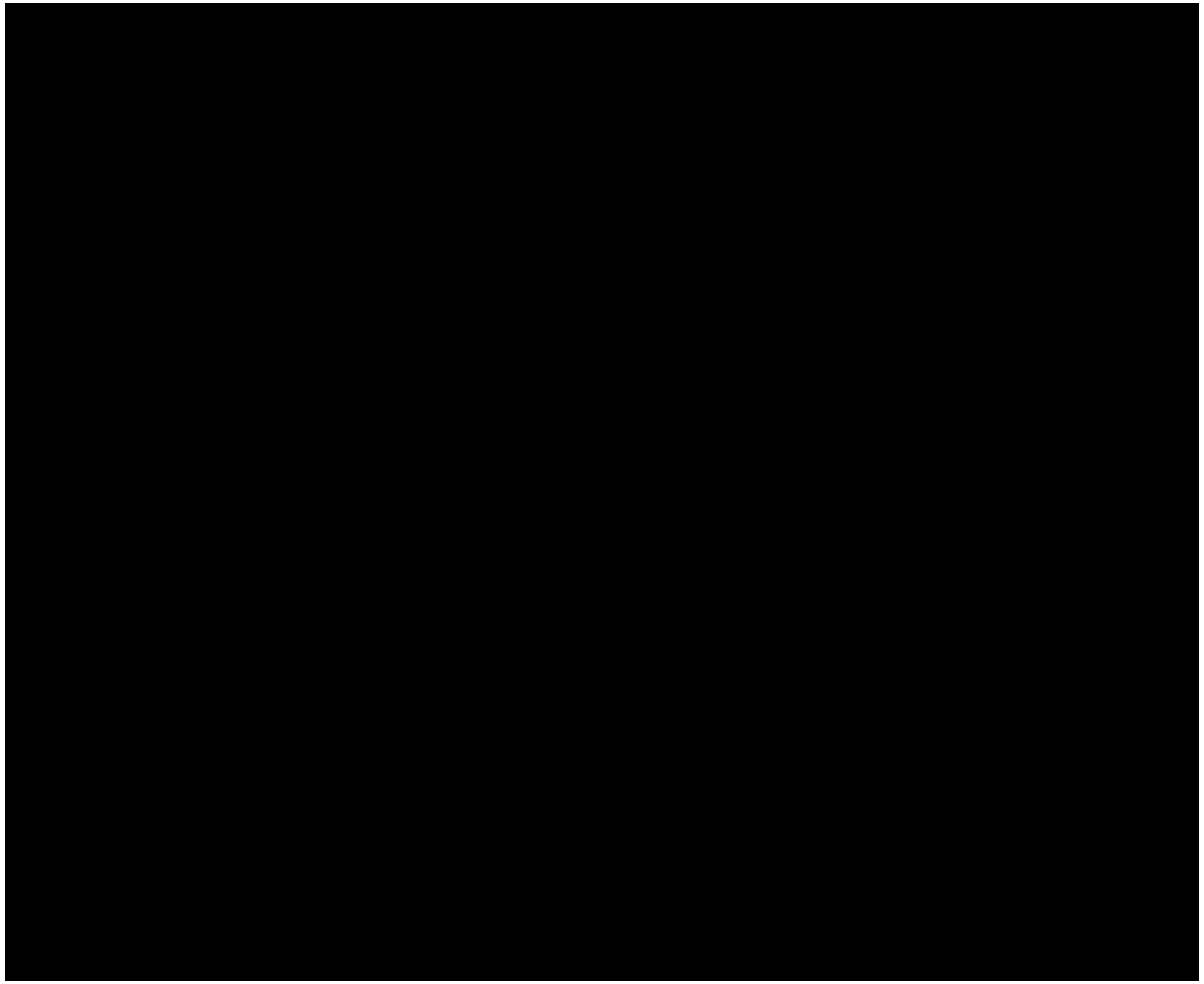


Q-learning with convolutional networks

- "Human-level control through deep reinforcement learning," Mnih et al. '13
- Q-learning with convolutional networks
- Uses replay buffer and target network
- One-step backup
- One gradient step
- Can be improved a lot with double Q-learning (and other tricks)

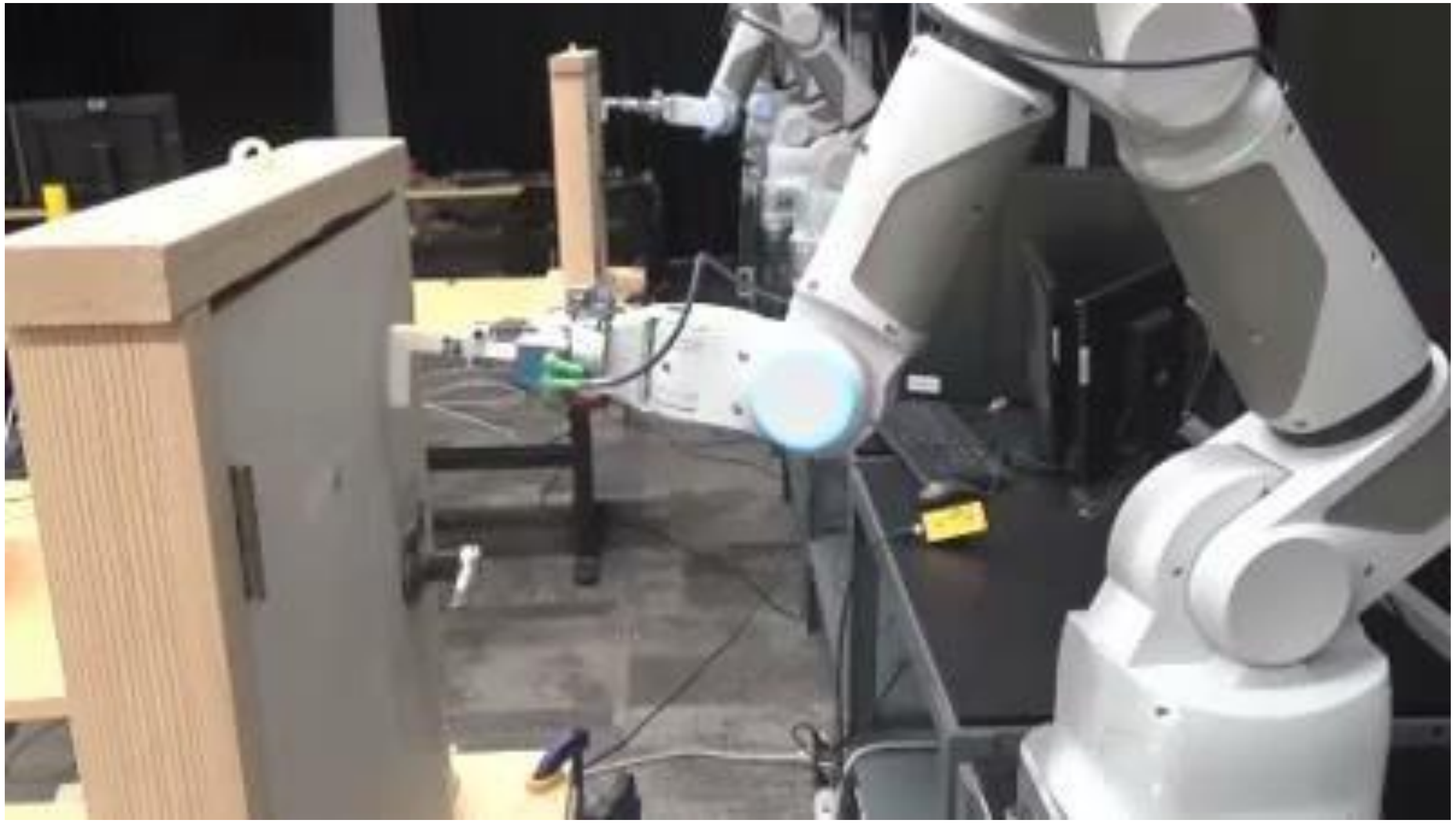
Q-learning with continuous actions

- "Continuous control with deep reinforcement learning," Lillicrap et al. '15
- Continuous actions with maximizer network
- Uses replay buffer and target network (with Polyak averaging)
- One-step backup
- One gradient step per simulator step

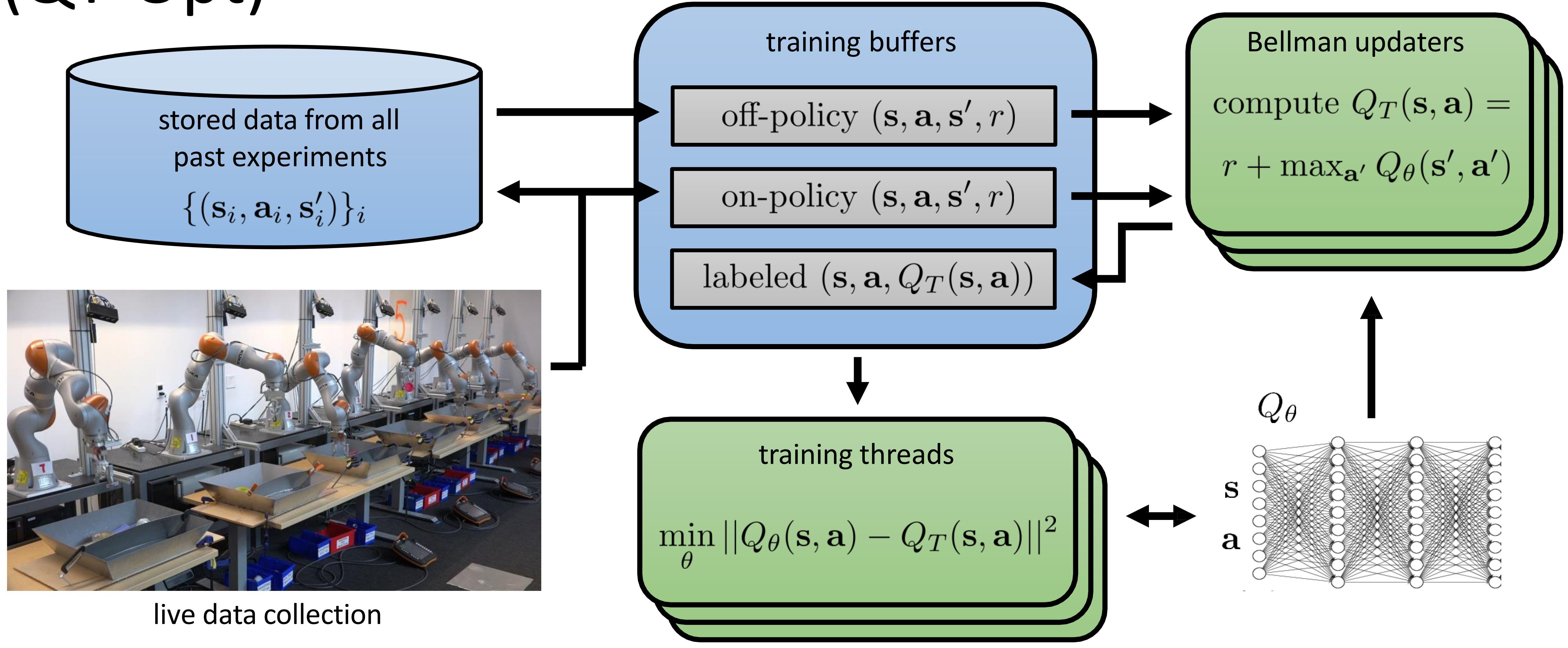


Q-learning on a real robot

- "Robotic manipulation with deep reinforcement learning and …," Gu*, Holly*, et al. '17
- Continuous actions with NAF (quadratic in actions)
- Uses replay buffer and target network
- One-step backup
- Four gradient steps per simulator step for efficiency
- Parallelized across multiple robots



Large-scale Q-learning with continuous actions (QT-Opt)


stored data from all past experiments
$\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)\}_i$
live data collection
training buffers
off-policy $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
on-policy $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
labeled $(\mathbf{s}, \mathbf{a}, Q_T(\mathbf{s}, \mathbf{a}))$
Bellman updaters
compute $Q_T(\mathbf{s}, \mathbf{a}) =$
$r + \max_{\mathbf{a}'} Q_\theta(\mathbf{s}', \mathbf{a}')$
training threads
$\min_\theta ||Q_\theta(\mathbf{s}, \mathbf{a}) - Q_T(\mathbf{s}, \mathbf{a})||^2$
Q_θ
$\mathbf{s}$
$\mathbf{a}$

Kalashnikov, Irpan, Pastor, Ibarz, Herzong, Jang, Quillen, Holly, Kalakrishnan, Vanhoucke, Levine. **QT-Opt: Scalable Deep Reinforcement Learning of Vision-Based Robotic Manipulation Skills**

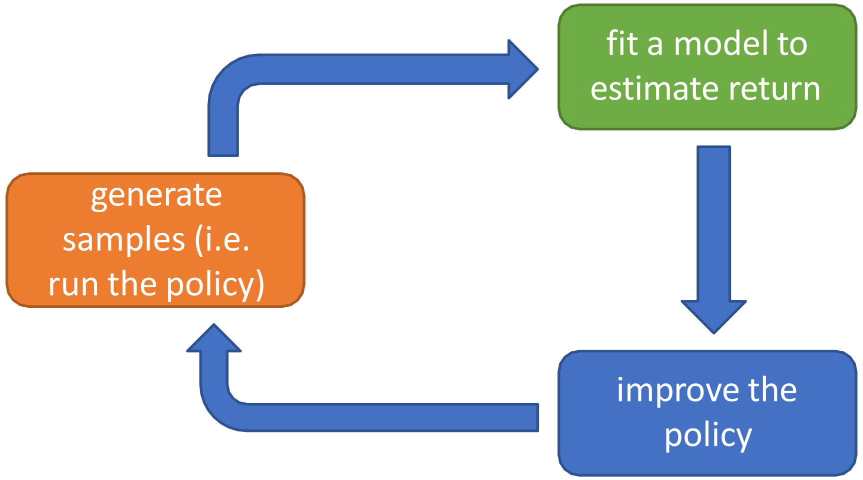
Q-learning suggested readings

- Classic papers
  - Watkins. (1989). Learning from delayed rewards: introduces Q-learning
  - Riedmiller. (2005). Neural fitted Q-iteration: batch-mode Q-learning with neural networks
- Deep reinforcement learning Q-learning papers
  - Lange, Riedmiller. (2010). Deep auto-encoder neural networks in reinforcement learning: early image-based Q-learning method using autoencoders to construct embeddings
  - Mnih et al. (2013). Human-level control through deep reinforcement learning: Q-learning with convolutional networks for playing Atari.
  - Van Hasselt, Guez, Silver. (2015). Deep reinforcement learning with double Q-learning: a very effective trick to improve performance of deep Q-learning.
  - Lillicrap et al. (2016). Continuous control with deep reinforcement learning: continuous Q-learning with actor network for approximate maximization.
  - Gu, Lillicrap, Stuskever, L. (2016). Continuous deep Q-learning with model-based acceleration: continuous Q-learning with action-quadratic value functions.
  - Wang, Schaul, Hessel, van Hasselt, Lanctot, de Freitas (2016). Dueling network architectures for deep reinforcement learning: separates value and advantage estimation in Q-function.

Review

- Q-learning in practice
  - Replay buffers
  - Target networks
- Generalized fitted Q-iteration
- Double Q-learning
- Multi-step Q-learning
- Q-learning with continuous actions
  - Random sampling
  - Analytic optimization
  - Second "actor" network

$$Q_\phi(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}',\mathbf{a}')$$


fit a model to estimate return
improve the policy
generate samples (i.e. run the policy)

$$\mathbf{a} = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a})$$