# Communicating Multi-agent Collision Avoidance with Deep Reinforcement Learning

By
Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P. How

Presenter: Jared Choi

# Motivation

- Finding a path
  - Computationally expensive due to
    - Collision checking
    - Feasibility checking
    - Efficiency checking

# Motivation

- Finding a path
  - Computationally expensive due to
    - Collision checking
    - Feasibility checking
    - Efficiency checking

- Offline Learning

# Background

- A sequential decision making problem can be formulated as a Markov Decision Process (MDP)
- $M = <S, A, P, R, \gamma>$

# Background

- A sequential decision making problem can be formulated as a Markov Decision Process (MDP)
- $M = <S, A, P, R, \gamma>$
  - S (state space)
  - A(action space)
  - P(state transition model)
  - R: reward function
  - $\gamma$: discount factor

# State Space (M = <**S**, γA, P, R, >)

- S(state space)
  - System's state is constructed by concatenating the two agents' individual states

$$s^o = [p_x, p_y, v_x, v_y, r] \in \mathbb{R}^5$$

**Observable State Vector**
**(position (x,y), velocity(x,y), radius)**

$$s^h = [p_{gx}, p_{gy}, v_{pref}, \psi] \in \mathbb{R}^4$$

**Unobservable State vector**
**(goal position (x,y), preferred speed, he**

# State Space (M = <**S**, A, P, R, γ>)

- S(state space)
  - System's state is constructed by concatenating the two agents' individual states

$$s^o = [p_x, p_y, v_x, v_y, r] \in \mathbb{R}^5$$

**Observable State Vector**
**(position (x,y), velocity(x,y), radius)**

$$s^h = [p_{gx}, p_{gy}, v_{pref}, \psi] \in \mathbb{R}^4$$

**Unobservable State vector**
**(goal position (x,y), preferred speed, hea**

$$\mathbf{s}^{jn} = [\mathbf{s},\ \tilde{\mathbf{s}}^o] \in \mathbb{R}^{14}$$

# Action Space (M = <S, **A**, P, R, $\gamma$ >)

- A(action space):
  - Set of permissible velocity vectors, a(s) = v

$$\mathbf{a}(\mathbf{s}) = \mathbf{v} \text{ for } ||\mathbf{v}||_2 < v_{pref}$$

# State Transition Model(M = <S, A, **P**, R, $\gamma$ >)

- P(state transition model)
  - A probabilistic state transition model
  - Determined by the agents' kinematics
  - Unknown to us $P(\mathbf{s}_{t+1}^{jn}, \mathbf{s}_t^{jn} | \mathbf{a}_t)$

# Reward Function (M = <S, A, P, **R**, $\gamma$>)

- R: reward function
  - Award the agent for reaching its goal
  - Penalize the agent for getting too close or colliding with other agent

$$R(\mathbf{s}^{jn}, \mathbf{a}) = \begin{cases} -0.25 & \text{if} \quad d_{min} < 0 \\ -0.1 - d_{min}/2 & \text{else if} \quad d_{min} < 0.2 \\ 1 & \text{else if} \quad \mathbf{p} = \mathbf{p}_g \\ 0 & \text{o.w.} \end{cases}$$

# Discount Factor(M = <S, A, P, R, $\gamma$ >)

- Discount factor

$$\gamma \in [0, 1)$$

# Value Function

- The value of a state
- Value depends on $\gamma$
  - $\gamma$ close to 1
    - We care about our long term reward
  - $\gamma$ close to 0
    - We care only about our immediate reward

$$V^*(\mathbf{s}_0^{jn}) = \sum_{t=0}^{T} \gamma^{t \cdot v_{pref}} R(\mathbf{s}_t^{jn}, \pi^*(\mathbf{s}_t^{jn}))$$

# Optimal Policy

- The best trajectory at given state

$$\pi^*(\mathbf{s}_0^{jn}) = \underset{\mathbf{a}}{\operatorname{argmax}}\, R(\mathbf{s}_0, \mathbf{a}) + \gamma^{\Delta t \cdot v_{pref}} \int_{\mathbf{s}_1^{jn}} P(\mathbf{s}_0^{jn}, \mathbf{s}_1^{jn} | \mathbf{a}) V^*(\mathbf{s}_1^{jn}) d\mathbf{s}_1^{jn}$$

# Value Function and Optimal Policy

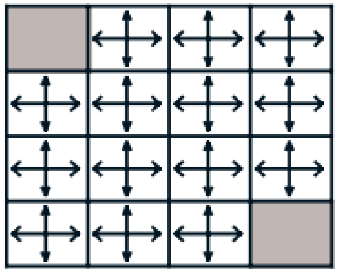
From David Silver’s slid

$v_k$ for the Random Policy

Greedy Policy w.r.t. $v_k$

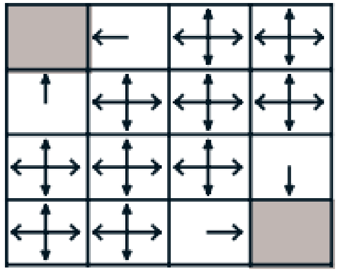
$k = 0$

| | | | |
|---|---|---|---|
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 0.0 |



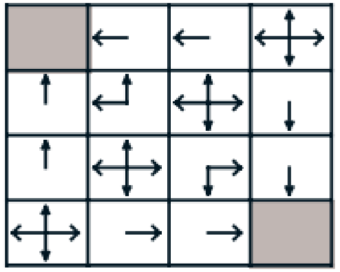
$k = 1$

| | | | |
|---|---|---|---|
| 0.0 | -1.0 | -1.0 | -1.0 |
| -1.0 | -1.0 | -1.0 | -1.0 |
| -1.0 | -1.0 | -1.0 | -1.0 |
| -1.0 | -1.0 | -1.0 | 0.0 |



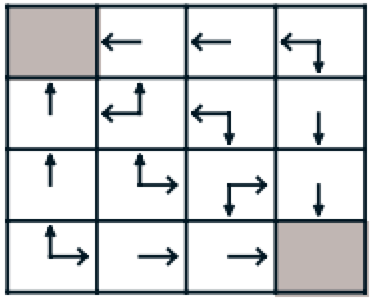
$k = 2$

| | | | |
|---|---|---|---|
| 0.0 | -1.7 | -2.0 | -2.0 |
| -1.7 | -2.0 | -2.0 | -2.0 |
| -2.0 | -2.0 | -2.0 | -1.7 |
| -2.0 | -2.0 | -1.7 | 0.0 |



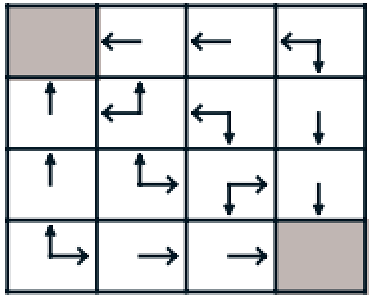
$k = 3$

| | | | |
|---|---|---|---|
| 0.0 | -2.4 | -2.9 | -3.0 |
| -2.4 | -2.9 | -3.0 | -2.9 |
| -2.9 | -3.0 | -2.9 | -2.4 |
| -3.0 | -2.9 | -2.4 | 0.0 |



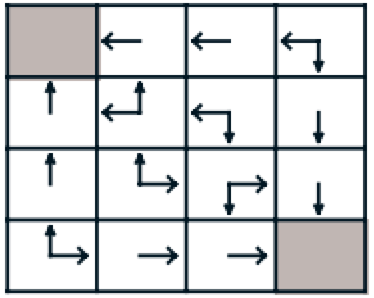
$k = 10$

| | | | |
|---|---|---|---|
| 0.0 | -6.1 | -8.4 | -9.0 |
| -6.1 | -7.7 | -8.4 | -8.4 |
| -8.4 | -8.4 | -7.7 | -6.1 |
| -9.0 | -8.4 | -6.1 | 0.0 |



$k = \infty$

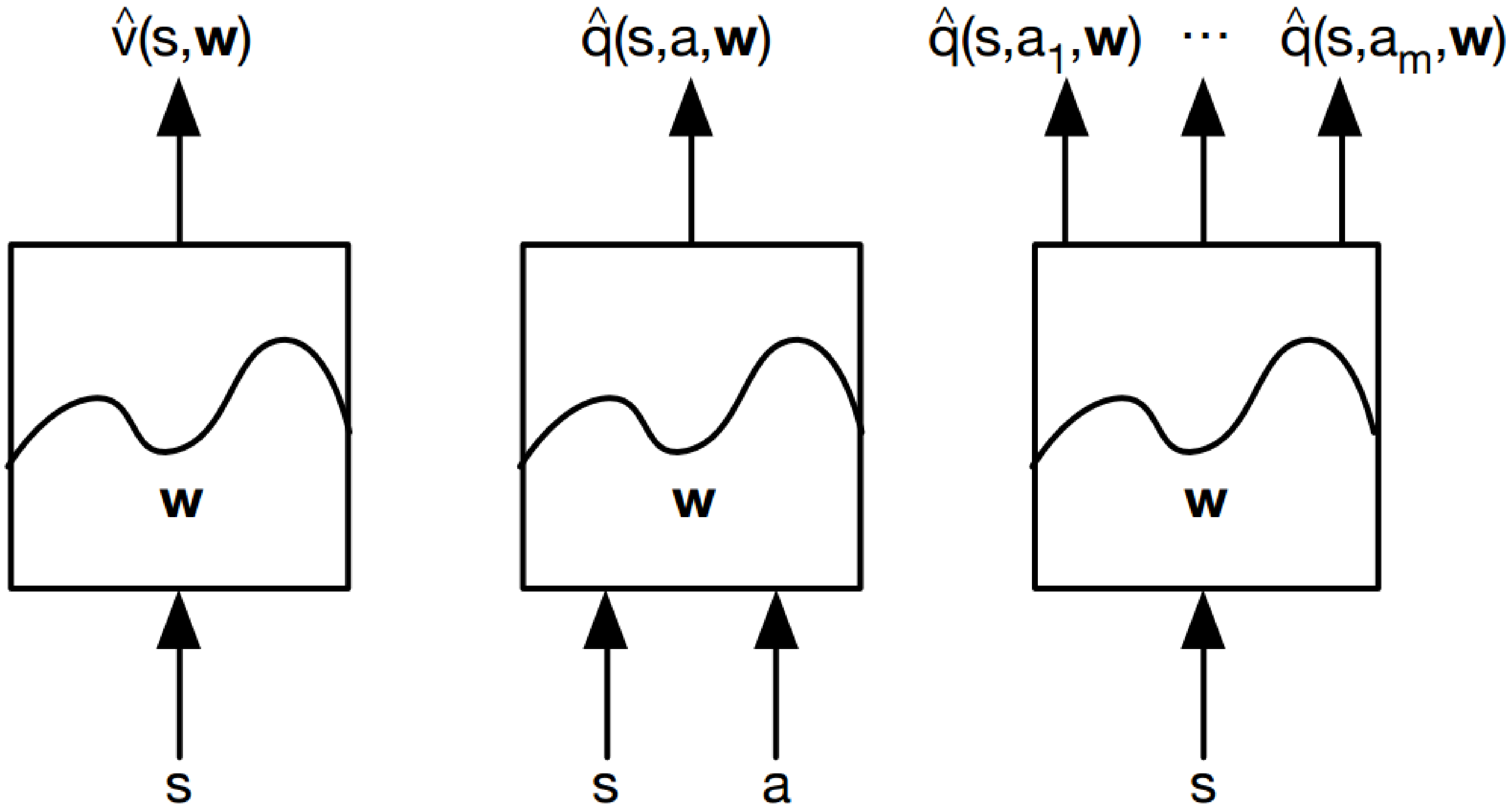
| | | | |
|---|---|---|---|
| 0.0 | -14. | -20. | -22. |
| -14. | -18. | -20. | -20. |
| -20. | -20. | -18. | -14. |
| -22. | -20. | -14. | 0.0 |



# Value Function and Optimal Policy

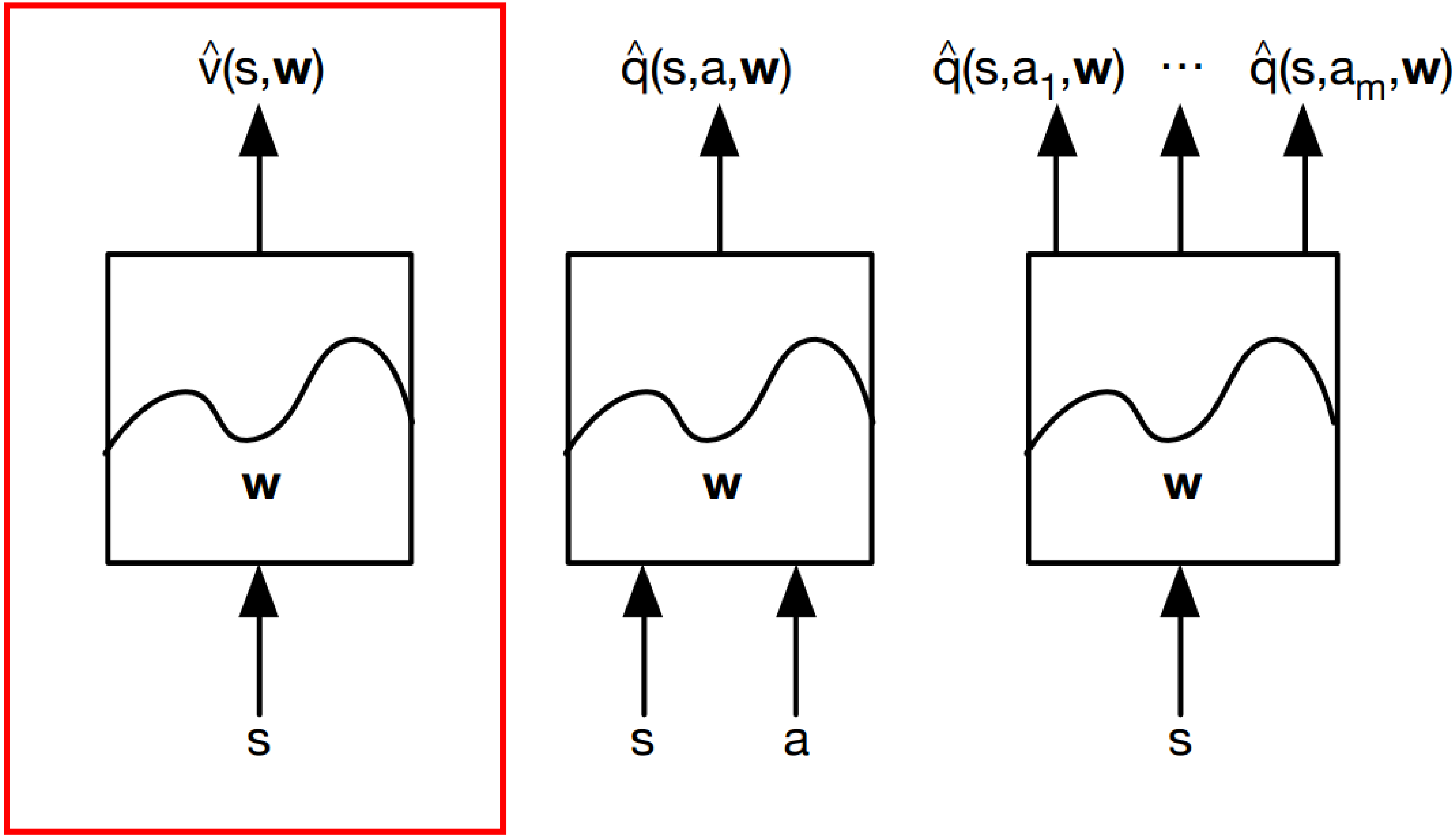
- Every state s has value V(s)
  - Store it in a lookup table
    - In a grid world : 16 values
    - In motion planning : Infinite values (b/c it's continuous state space)
- Solution:
  - Approximate value via neural network

# Value Function and Optimal Policy




From David Silver's slides

# Value Function and Optimal Policy



# Collision Avoidance Deep Reinforcement Learning

1. Train Value network using ORCA

2. Train again with Deep reinforcement Learning

# Collision Avoidance Deep Reinforcement Learning

1. Train Value network using ORCA
   - Why pre-train?

# Collision Avoidance Deep Reinforcement Learning

1. Train Value network using ORCA
   - Why pre-train?
     - Initializing the neural network is crucial to convergence
     - We want the network to output something reasonable

# Collision Avoidance Deep Reinforcement Learning

1. Train Value network using ORCA

- Why pre-train?
  - Initializing the neural network is crucial to convergence
  - We want the network to output something reasonable
- Generate 500 trajectories as a training set
  - Each trajectory contains 40 state-value pairs (total of 20,000 pairs)
  - Back-propagate to minimize our loss function:

$$\operatorname{argmin}_{\mathbf{w}} \sum_{k=1}^{N} \left( y_k - V(\mathbf{s}_k^{jn}; \mathbf{w}) \right)^2$$

# Collision Avoidance Deep Reinforcement Learning

1. Train Value network using ORCA

2. Train again with Deep reinforcement Learning

# Collision Avoidance Deep Reinforcement Learning

1. Train again with Deep reinforcement Learning

**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$
**Output:** value network $V(\cdot;\mathbf{w})$
$V(\cdot;\mathbf{w}) \leftarrow$ train_nn($D$) //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1,\ldots,N_{eps}$ **do**
  **for** *m times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL($V$), $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL($V$)
    $\mathbf{y}_{0:T},\ \tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues($V'$, $\mathbf{s}_{0:t_f}$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f}$)
    $E \leftarrow$ assimilate$\Big(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\Big)$
  $e \leftarrow$ randSubset($E$)
  $\mathbf{w} \leftarrow$ backprop($e$)
  **for** *every $C$ episodes* **do**
    Evaluate($V$), $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

1. Train again with Deep reinforcement Learning

**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$
**Output:** value network $V(\cdot;\mathbf{w})$
$V(\cdot;\mathbf{w}) \leftarrow$ train_nn($D$) //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1,\ldots,N_{eps}$ **do**
  **for** *m times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL($V$), $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL($V$)
    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues($V'$, $\mathbf{s}_{0:t_f}$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f}$)
    $E \leftarrow$ assimilate$\left(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\right)$
  $e \leftarrow$ randSubset($E$)
  $\mathbf{w} \leftarrow$ backprop($e$)
  **for** *every C episodes* **do**
    Evaluate($V$), $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

1. Train again with Deep reinforcement Learning

**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$
**Output:** value network $V(\cdot;\mathbf{w})$
$V(\cdot;\mathbf{w}) \leftarrow$ train_nn($D$) //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1,\ldots,N_{eps}$ **do**
  **for** $m$ *times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL($V$), $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL($V$)
    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues($V'$, $\mathbf{s}_{0:t_f}$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f}$)
    $E \leftarrow$ assimilate$\left(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\right)$
  $e \leftarrow$ randSubset($E$)
  $\mathbf{w} \leftarrow$ backprop($e$)
  **for** *every* $C$ *episodes* **do**
    Evaluate($V$), $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

## 1. Train again with Deep reinforcement Learning


**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$
**Output:** value network $V(\cdot; \mathbf{w})$
$V(\cdot; \mathbf{w}) \leftarrow$ train_nn($D$) //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1, \ldots, N_{eps}$ **do**
  **for** *$m$ times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL($V$), $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL($V$)
    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues($V'$, $\mathbf{s}_{0:t_f}$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f}$)
    $E \leftarrow$ assimilate$\Big(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\Big)$
  $e \leftarrow$ randSubset($E$)
  $\mathbf{w} \leftarrow$ backprop($e$)
  **for** *every $C$ episodes* **do**
    Evaluate($V$), $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

1. Train again with Deep reinforcement Learning


**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$

**Output:** value network $V(\cdot; \mathbf{w})$

$V(\cdot; \mathbf{w}) \leftarrow$ train_nn$(D)$ //step 1: initialization

duplicate value net $V' \leftarrow V$ //step 2: RL

initialize experience set $E \leftarrow D$

**for** *episode*$=1, \ldots, N_{eps}$ **do**

  **for** *m times* **do**

    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()

    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL$(V)$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL$(V)$

    $\mathbf{y}_{0:T}, \tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues$(V', \mathbf{s}_{0:t_f}, \tilde{\mathbf{s}}_{0:\tilde{t}_f})$

    $E \leftarrow$ assimilate$\Big(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\Big)$

  $e \leftarrow$ randSubset$(E)$

  $\mathbf{w} \leftarrow$ backprop$(e)$

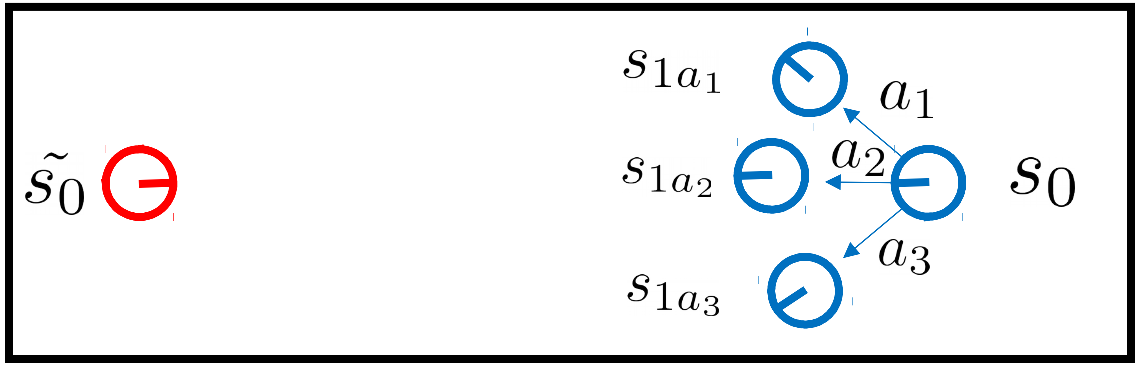
  **for** *every C episodes* **do**

    Evaluate$(V)$, $V' \leftarrow V$

**return** $V$

# Collision Avoidance Deep Reinforcement Learning

## 1.Train again with Deep reinforcement Learning

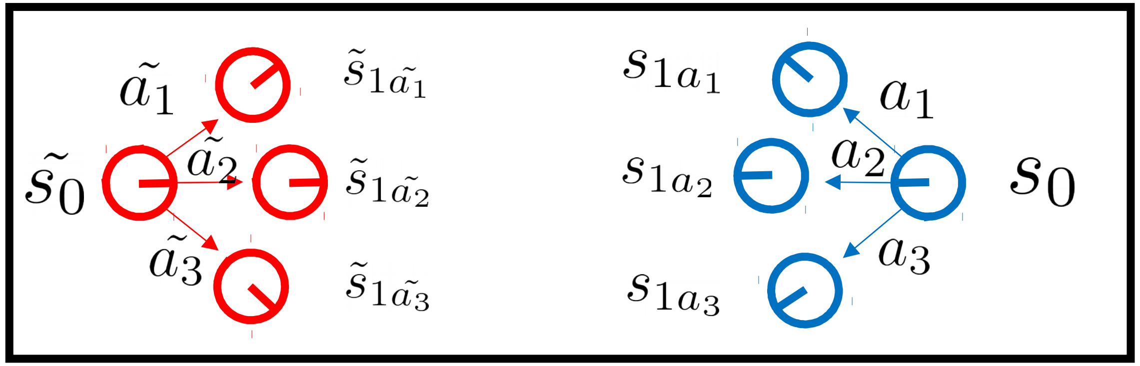



**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$
**Output:** value network $V(\cdot;\mathbf{w})$
$V(\cdot;\mathbf{w}) \leftarrow$ train_nn($D$) //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1,\ldots,N_{eps}$ **do**
  **for** *m times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL($V$), $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL($V$)
    $\mathbf{y}_{0:T},\ \tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues($V'$, $\mathbf{s}_{0:t_f}$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f}$)
    $E \leftarrow \text{assimilate}\Big(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\Big)$
  $e \leftarrow$ randSubset($E$)
  $\mathbf{w} \leftarrow$ backprop($e$)
  **for** *every C episodes* **do**
    Evaluate($V$), $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

1. Train again with Deep reinforcement Learning




**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$

**Output:** value network $V(\cdot; \mathbf{w})$

$V(\cdot; \mathbf{w}) \leftarrow$ train_nn$(D)$ //step 1: initialization

duplicate value net $V' \leftarrow V$ //step 2: RL

initialize experience set $E \leftarrow D$

**for** *episode*$=1, \ldots, N_{eps}$ **do**

  **for** *m times* **do**

    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()

    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL$(V)$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL$(V)$

    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues$(V', \mathbf{s}_{0:t_f}, \tilde{\mathbf{s}}_{0:\tilde{t}_f})$

    $E \leftarrow$ assimilate$\left(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\right)$

  $e \leftarrow$ randSubset$(E)$

  $\mathbf{w} \leftarrow$ backprop$(e)$

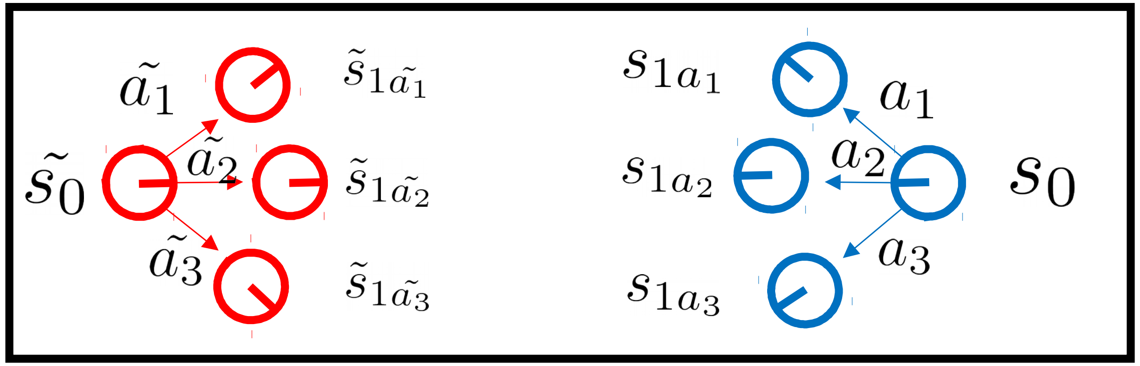
  **for** *every C episodes* **do**

    Evaluate$(V)$, $V' \leftarrow V$

**return** $V$

# Collision Avoidance Deep Reinforcement Learning

## 1. Train again with Deep reinforcement Learning



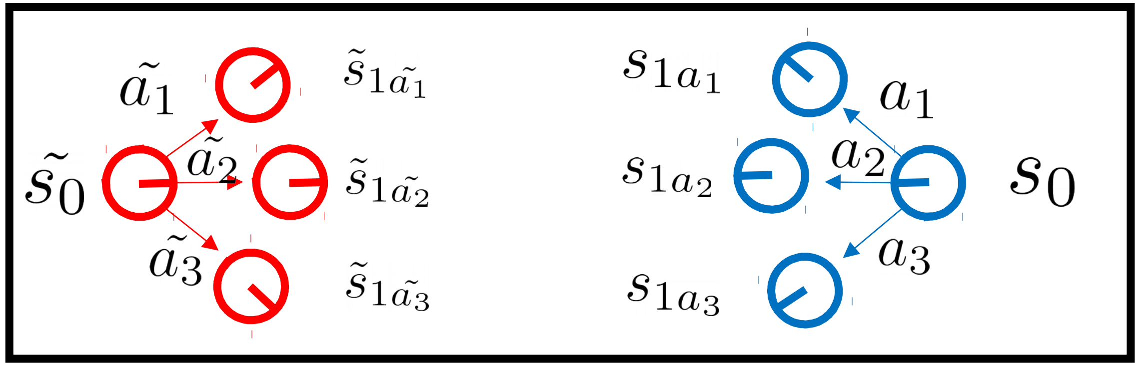

$\tilde{s_1} \approx (\tilde{s_0} + \Delta t \cdot \tilde{v_0})$ $\quad$ $s_1 = (s_0 + \Delta t \cdot a_n)$

**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$
**Output:** value network $V(\cdot; \mathbf{w})$
$V(\cdot; \mathbf{w}) \leftarrow$ train_nn($D$) //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1, \ldots, N_{eps}$ **do**
  **for** *m times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL($V$), $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL($V$)
    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues($V'$, $\mathbf{s}_{0:t_f}$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f}$)
    $E \leftarrow$ assimilate$\Big(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\Big)$
  $e \leftarrow$ randSubset($E$)
  $\mathbf{w} \leftarrow$ backprop($e$)
  **for** *every C episodes* **do**
    Evaluate($V$), $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

## 1.Train again with Deep reinforcement Learning




$\tilde{s_1} \approx (\tilde{s_0} + \Delta t \cdot \tilde{v_0})$ $\quad s_1 = (s_0 + \Delta t \cdot a_n)$

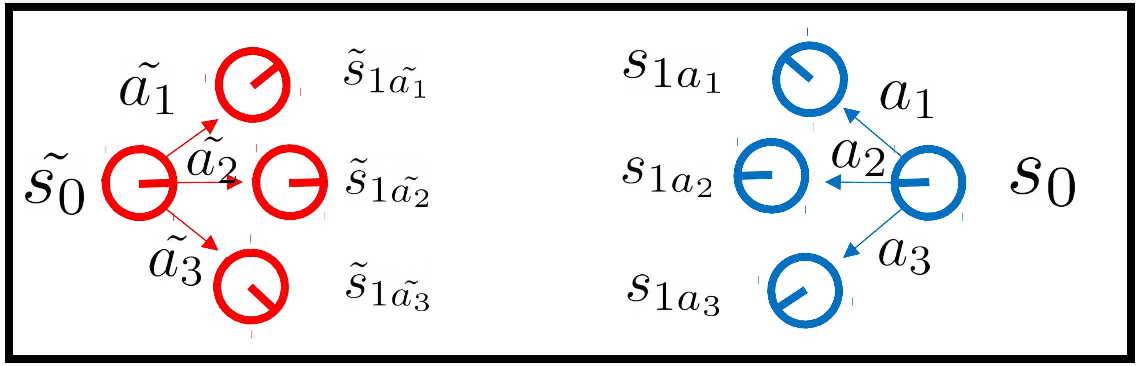
$[s_{1a_1}, \tilde{s}_{1\tilde{a}_1}]$

**Algorithm 2:** Deep V-learning

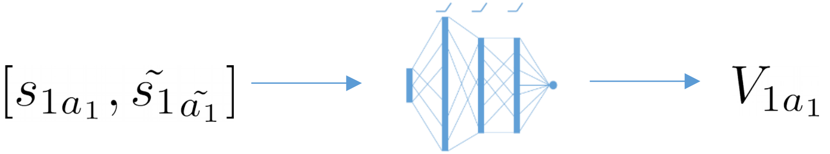
**Input:** trajectory training set $D$
**Output:** value network $V(\cdot; \mathbf{w})$
$V(\cdot; \mathbf{w}) \leftarrow$ train_nn($D$) //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1, \ldots, N_{eps}$ **do**
  **for** *m times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL($V$), $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL($V$)
    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues($V'$, $\mathbf{s}_{0:t_f}$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f}$)
    $E \leftarrow$ assimilate$\left(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\right)$
  $e \leftarrow$ randSubset($E$)
  $\mathbf{w} \leftarrow$ backprop($e$)
  **for** *every C episodes* **do**
    Evaluate($V$), $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

## 1. Train again with Deep reinforcement Learning




$\tilde{s_1} \approx (\tilde{s_0} + \Delta t \cdot \tilde{v_0})$ $\quad s_1 = (s_0 + \Delta t \cdot a_n)$

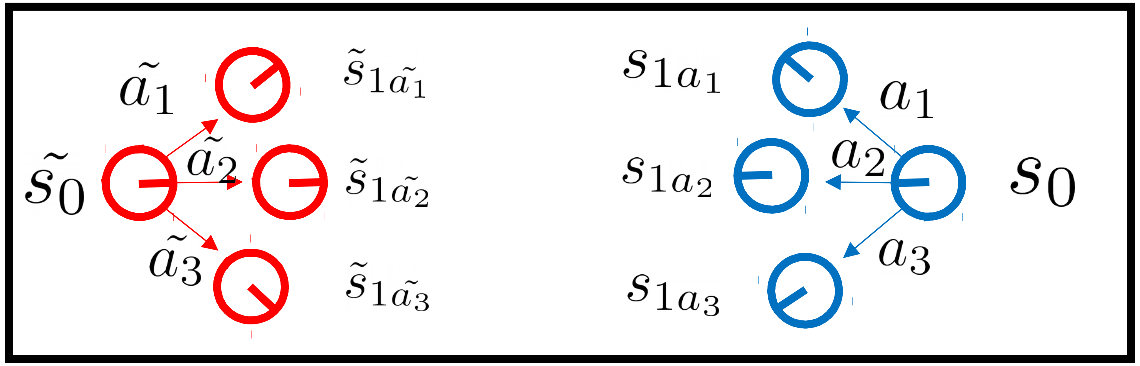



**Algorithm 2:** Deep V-learning

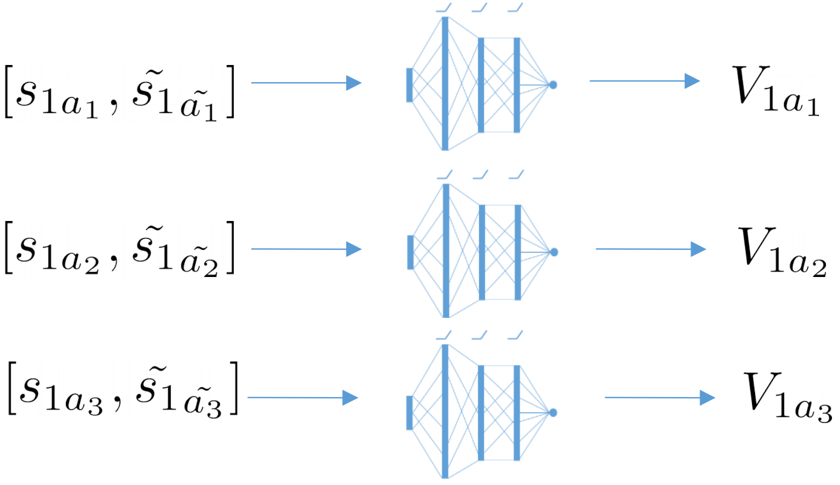
```
Input: trajectory training set D
Output: value network V(·; w)
V(·; w) ← train_nn(D)                         //step 1: initialization
duplicate value net V′ ← V                    //step 2: RL
initialize experience set E ← D
for episode=1, ..., N_eps do
    for m times do
        s_0, s̃_0 ← randomTestcase()
        s_{0:t_f} ← CADRL(V), s̃_{0:t̃_f} ← CADRL(V)
        y_{0:T}, ỹ_{0:t̃_f} ← findValues(V′, s_{0:t_f}, s̃_{0:t̃_f})
        E ← assimilate(E, (y, s^{jn})_{0:t_f}, (ỹ, s̃^{jn})_{0:t̃_f})
    e ← randSubset(E)
    w ← backprop(e)
    for every C episodes do
        Evaluate(V), V′ ← V
return V
```

# Collision Avoidance Deep Reinforcement Learning

## 1. Train again with Deep reinforcement Learning




$\tilde{s}_1 \approx (\tilde{s}_0 + \Delta t \cdot \tilde{v}_0)$ $\quad$ $s_1 = (s_0 + \Delta t \cdot a_n)$

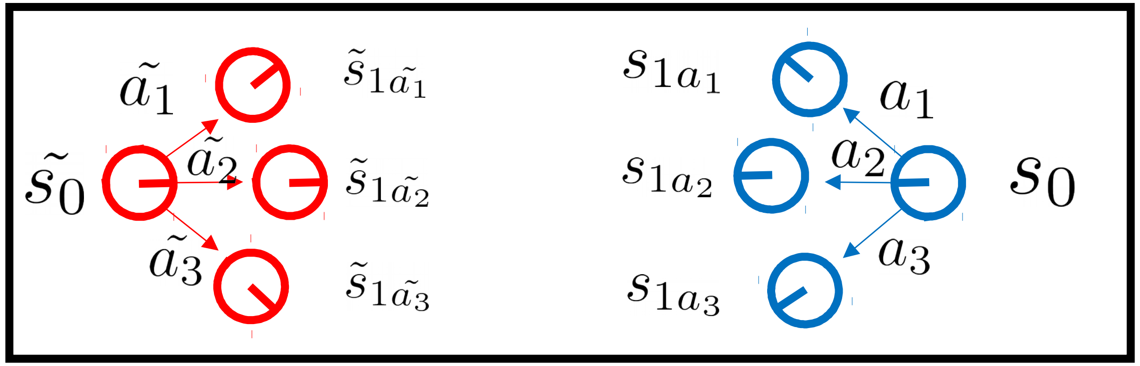



**Algorithm 2:** Deep V-learning

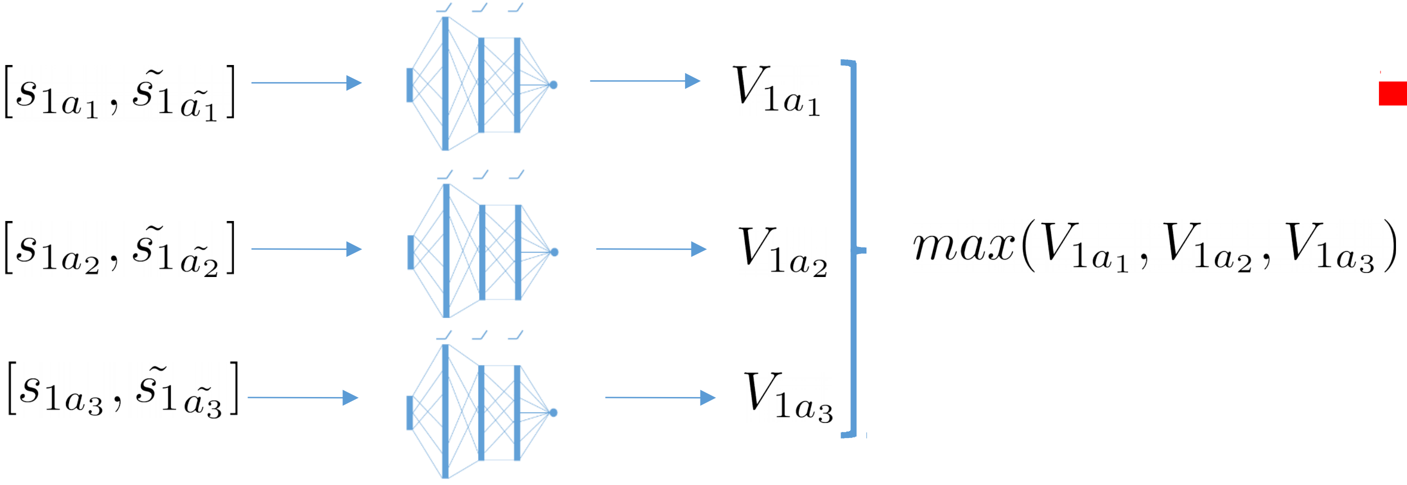
**Input:** trajectory training set $D$
**Output:** value network $V(\cdot; \mathbf{w})$
$V(\cdot; \mathbf{w}) \leftarrow$ train_nn$(D)$ //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1, \ldots, N_{eps}$ **do**
  **for** *m times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL$(V)$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL$(V)$
    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues$(V', \mathbf{s}_{0:t_f}, \tilde{\mathbf{s}}_{0:\tilde{t}_f})$
    $E \leftarrow$ assimilate$\left(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\right)$
  $e \leftarrow$ randSubset$(E)$
  $\mathbf{w} \leftarrow$ backprop$(e)$
  **for** *every C episodes* **do**
    Evaluate$(V)$, $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

## 1. Train again with Deep reinforcement Learning




$\tilde{s_1} \approx (\tilde{s_0} + \Delta t \cdot \tilde{v_0})$  $s_1 = (s_0 + \Delta t \cdot a_n)$

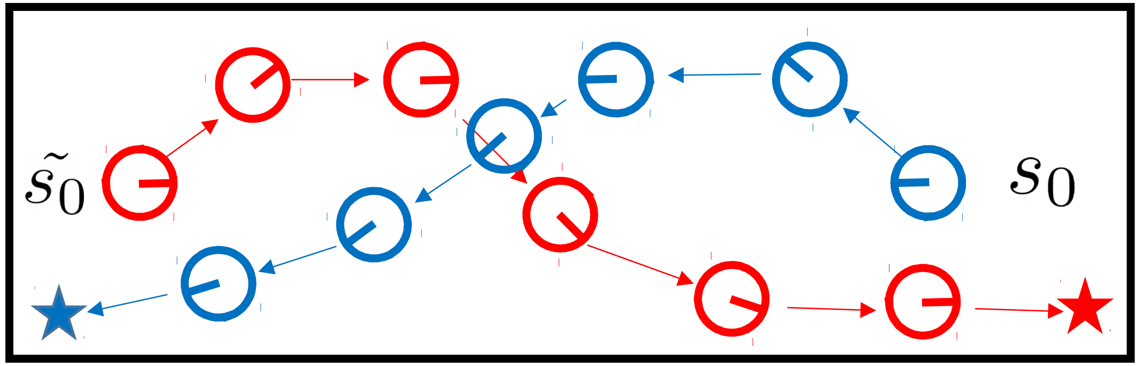



**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$
**Output:** value network $V(\cdot; \mathbf{w})$
$V(\cdot; \mathbf{w}) \leftarrow$ train_nn$(D)$ //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1, \ldots, N_{eps}$ **do**
  **for** *m times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL$(V)$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL$(V)$
    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues$(V', \mathbf{s}_{0:t_f}, \tilde{\mathbf{s}}_{0:\tilde{t}_f})$
    $E \leftarrow$ assimilate$\left(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\right)$
  $e \leftarrow$ randSubset$(E)$
  $\mathbf{w} \leftarrow$ backprop$(e)$
  **for** *every $C$ episodes* **do**
    Evaluate$(V)$, $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

1. Train again with Deep reinforcement Learning

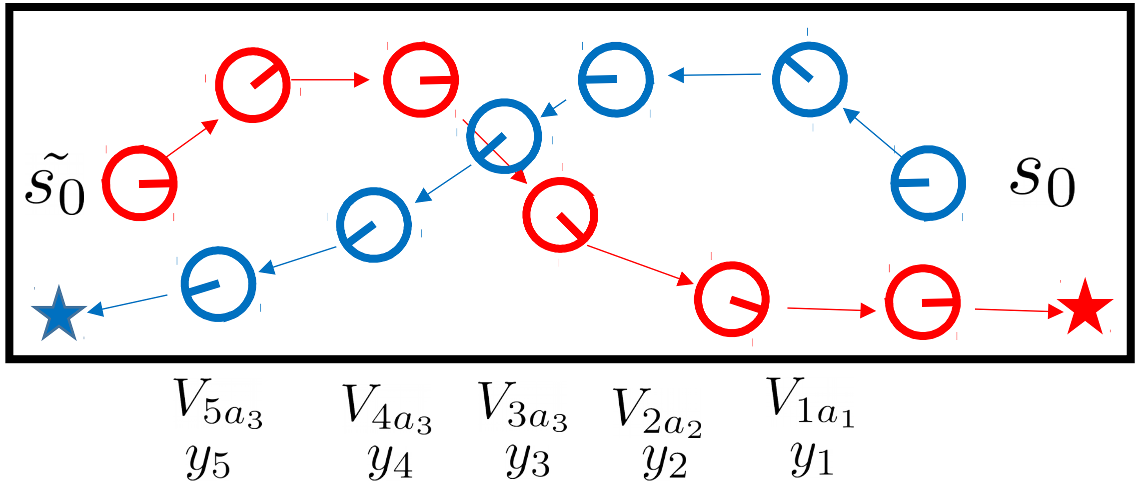



**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$
**Output:** value network $V(\cdot;\mathbf{w})$
$V(\cdot;\mathbf{w}) \leftarrow$ train_nn$(D)$ //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1,\ldots,N_{eps}$ **do**
  **for** $m$ *times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL$(V)$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL$(V)$
    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues$(V', \mathbf{s}_{0:t_f}, \tilde{\mathbf{s}}_{0:\tilde{t}_f})$
    $E \leftarrow$ assimilate$\Big(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\Big)$
  $e \leftarrow$ randSubset$(E)$
  $\mathbf{w} \leftarrow$ backprop$(e)$
  **for** *every* $C$ *episodes* **do**
    Evaluate$(V)$, $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

## 1. Train again with Deep reinforcement Learning

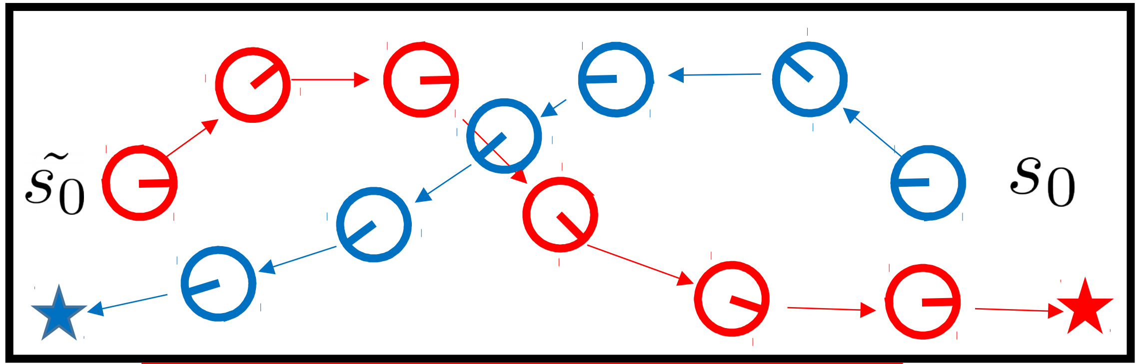



**Algorithm 2:** Deep V-learning

**Input:** trajectory training set $D$
**Output:** value network $V(\cdot;\mathbf{w})$
$V(\cdot;\mathbf{w}) \leftarrow$ train_nn$(D)$ //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1,\ldots,N_{eps}$ **do**
  **for** *$m$ times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow$ randomTestcase()
    $\mathbf{s}_{0:t_f} \leftarrow$ CADRL$(V)$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow$ CADRL$(V)$
    $\mathbf{y}_{0:T},\ \tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow$ findValues$(V',\ \mathbf{s}_{0:t_f},\ \tilde{\mathbf{s}}_{0:\tilde{t}_f})$
    $E \leftarrow$ assimilate$\left(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\right)$
  $e \leftarrow$ randSubset$(E)$
  $\mathbf{w} \leftarrow$ backprop$(e)$
  **for** *every $C$ episodes* **do**
    Evaluate$(V)$, $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

## 1. Train again with Deep reinforcement Learning




$$\begin{array}{ccccc} V_{5a_3} & V_{4a_3} & V_{3a_3} & V_{2a_2} & V_{1a_1} \\ y_5 & y_4 & y_3 & y_2 & y_1 \end{array}$$

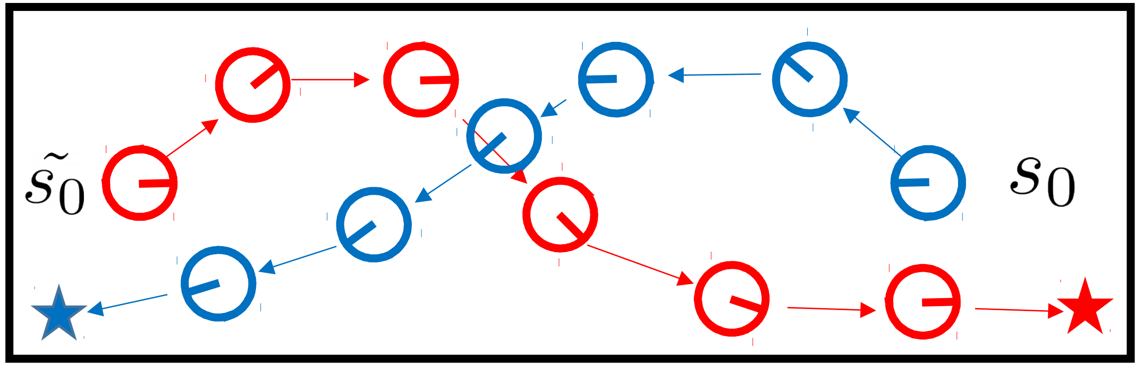
$$E = \begin{bmatrix} \begin{array}{ccccc} V_{5a_3} & V_{4a_3} & V_{3a_3} & V_{2a_2} & V_{1a_1} \\ y_5 & y_4 & y_3 & y_2 & y_1 \end{array} \\ \bullet \quad \bullet \quad \bullet \end{bmatrix}$$

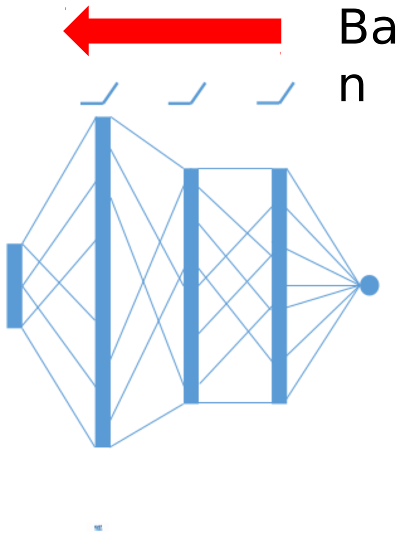
**Algorithm 2:** Deep V-learning

```
Input: trajectory training set D
Output: value network V(·; w)
V(·; w) ← train_nn(D)                          //step 1: initialization
duplicate value net V′ ← V                      //step 2: RL
initialize experience set E ← D
for episode=1, ..., N_eps do
    for m times do
        s_0, s̃_0 ← randomTestcase()
        s_{0:t_f} ← CADRL(V), s̃_{0:t̃_f} ← CADRL(V)
        y_{0:T}, ỹ_{0:t̃_f} ← findValues(V′, s_{0:t_f}, s̃_{0:t̃_f})
        E ← assimilate(E, (y, s^{jn})_{0:t_f}, (ỹ, s̃^{jn})_{0:t̃_f})
    e ← randSubset(E)
    w ← backprop(e)
    for every C episodes do
        Evaluate(V), V′ ← V
return V
```

# Collision Avoidance Deep Reinforcement Learning

## 1.Train again with Deep reinforcement Learning



Backpropagatio
n

$$\operatorname{argmin}_{\mathbf{w}} \sum_{k=1}^{N} \left( y_k - V(\mathbf{s}_k^{jn}; \mathbf{w}) \right)^2$$

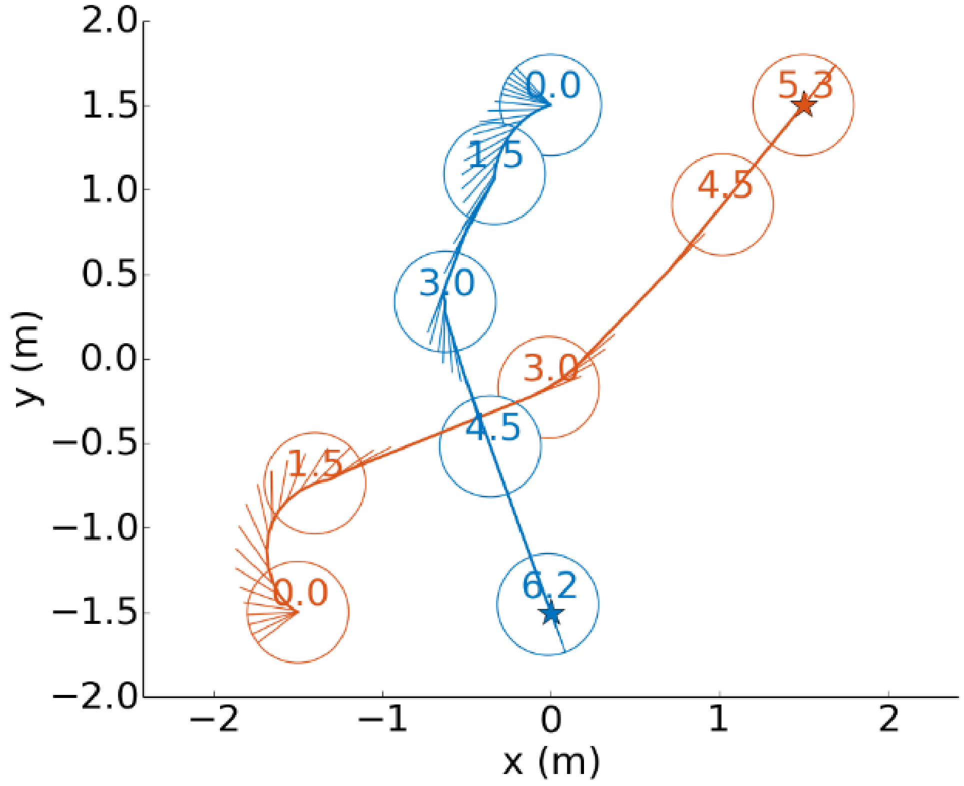
$$E = \begin{bmatrix} \boxed{\begin{matrix} V_{5a_3} & V_{4a_3} & V_{3a_3} & V_{2a_2} & V_{1a_1} \\ y_5 & y_4 & y_3 & y_2 & y_1 \end{matrix}} \\ \bullet \quad \bullet \quad \bullet \end{bmatrix}$$

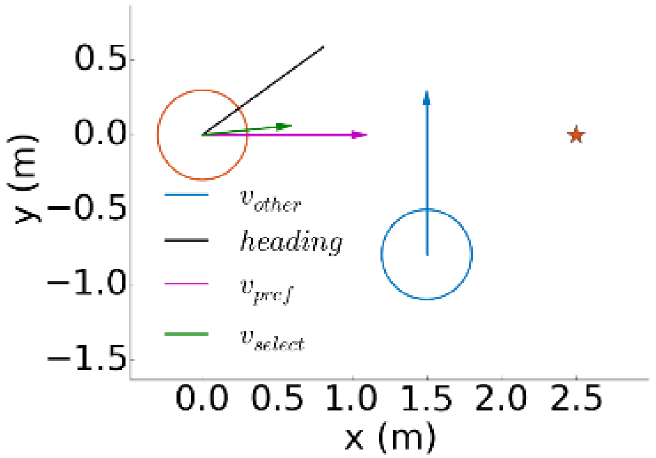
**Algorithm 2:** Deep V-learning

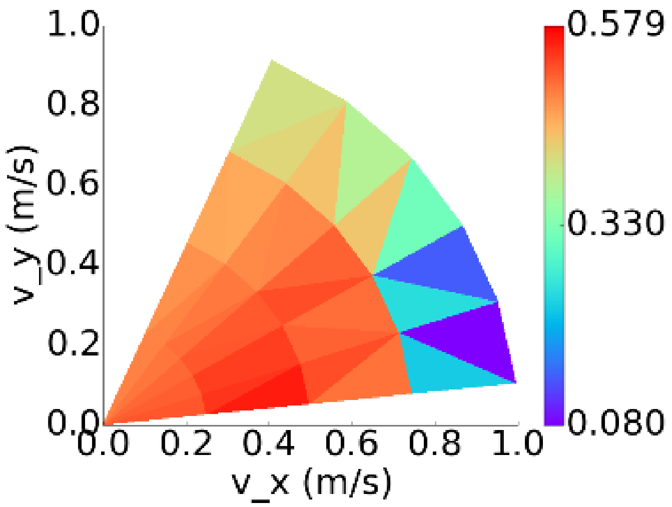
**Input:** trajectory training set $D$
**Output:** value network $V(\cdot; \mathbf{w})$
$V(\cdot; \mathbf{w}) \leftarrow \text{train\_nn}(D)$ //step 1: initialization
duplicate value net $V' \leftarrow V$ //step 2: RL
initialize experience set $E \leftarrow D$
**for** *episode*$=1, \ldots, N_{eps}$ **do**
  **for** *m times* **do**
    $\mathbf{s}_0, \tilde{\mathbf{s}}_0 \leftarrow \text{randomTestcase}()$
    $\mathbf{s}_{0:t_f} \leftarrow \text{CADRL}(V)$, $\tilde{\mathbf{s}}_{0:\tilde{t}_f} \leftarrow \text{CADRL}(V)$
    $\mathbf{y}_{0:T}$, $\tilde{\mathbf{y}}_{0:\tilde{t}_f} \leftarrow \text{findValues}(V', \mathbf{s}_{0:t_f}, \tilde{\mathbf{s}}_{0:\tilde{t}_f})$
    $E \leftarrow \text{assimilate}\left(E, (\mathbf{y}, \mathbf{s}^{jn})_{0:t_f}, (\tilde{\mathbf{y}}, \tilde{\mathbf{s}}^{jn})_{0:\tilde{t}_f}\right)$
  $e \leftarrow \text{randSubset}(E)$
  $\mathbf{w} \leftarrow \text{backprop}(e)$
  **for** *every C episodes* **do**
    $\text{Evaluate}(V)$, $V' \leftarrow V$
**return** $V$

# Collision Avoidance Deep Reinforcement Learning

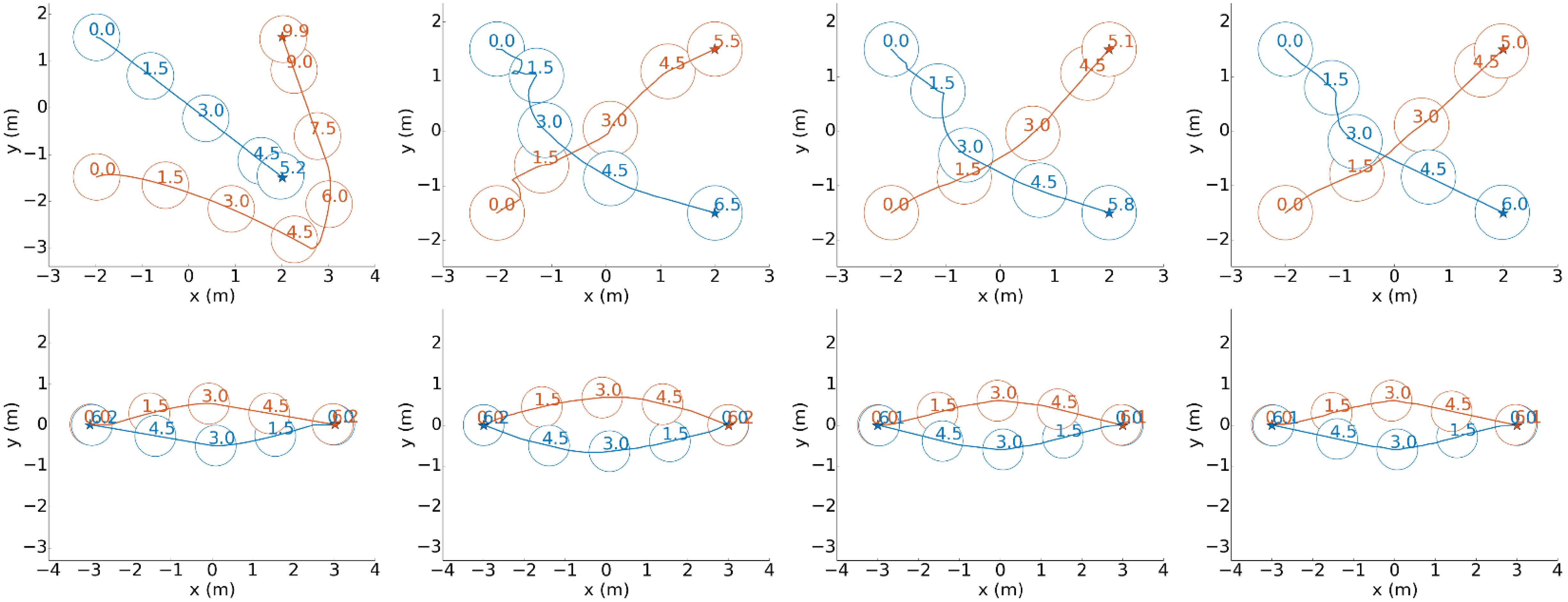
1. Train again with Deep reinforcement Learning



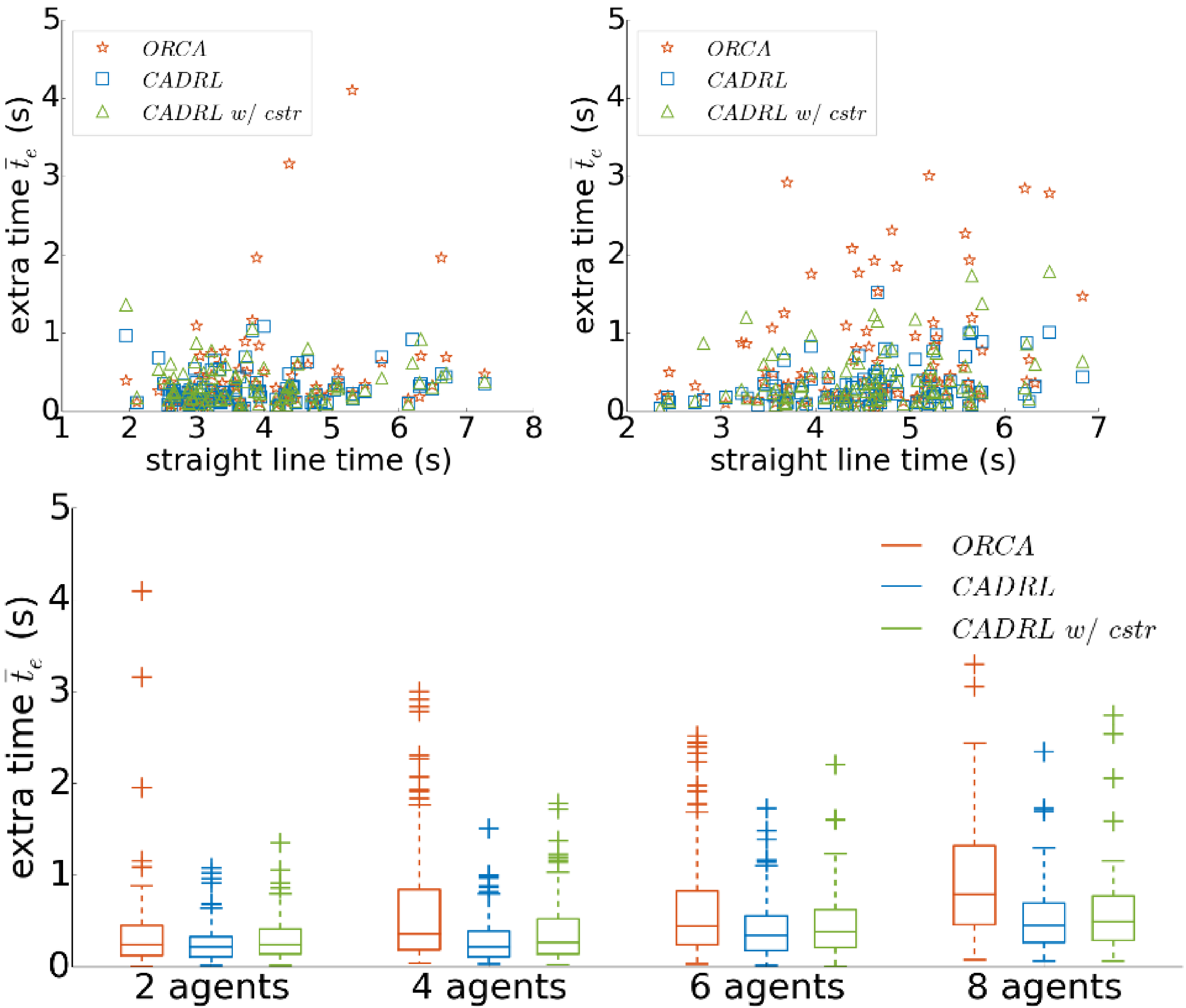



# Result



# Result


ORCA
CADRL
CADRL w/ cstr
extra time $\bar{t}_e$ (s)
straight line time (s)
ORCA
CADRL
CADRL w/ cstr
extra time $\bar{t}_e$ (s)
straight line time (s)
ORCA
CADRL
CADRL w/ cstr
extra time $\bar{t}_e$ (s)
2 agents
4 agents
6 agents
8 agents

# Result

| Test case configuration | | Extra time to goal $\bar{t}_e$ (s) [Avg / 75th / 90th percentile] | | | Average min separation dist. (m) | | |
|---|---|---|---|---|---|---|---|
| num agents | domain size (m) | ORCA | CADRL | CADRL w/ cstr | OCRA | CADRL | CADRL w/ cstr |
| 2 | $4.0 \times 4.0$ | 0.46 / 0.45 / 0.73 | 0.27 / 0.33 / 0.56 | 0.31 / 0.42 / 0.60 | 0.122 | 0.199 | 0.198 |
| 4 | $5.0 \times 5.0$ | 0.69 / 0.85 / 1.85 | 0.31 / 0.40 / 0.76 | 0.39 / 0.53 / 0.86 | 0.120 | 0.192 | 0.191 |
| 6 | $6.0 \times 6.0$ | 0.65 / 0.83 / 1.50 | 0.44 / 0.56 / 0.87 | 0.48 / 0.63 / 1.02 | 0.118 | 0.117 | 0.180 |
| 8 | $7.0 \times 7.0$ | 0.96 / 1.33 / 1.84 | 0.54 / 0.70 / 1.01 | 0.59 / 0.77 / 1.09 | 0.110 | 0.171 | 0.170 |

# Q&A

# Quiz

- Values are update after each episode (T/F)
- Value function needs to be trained with ORCA (T/F)
- ORCA path does not need to be optimal (T/F)