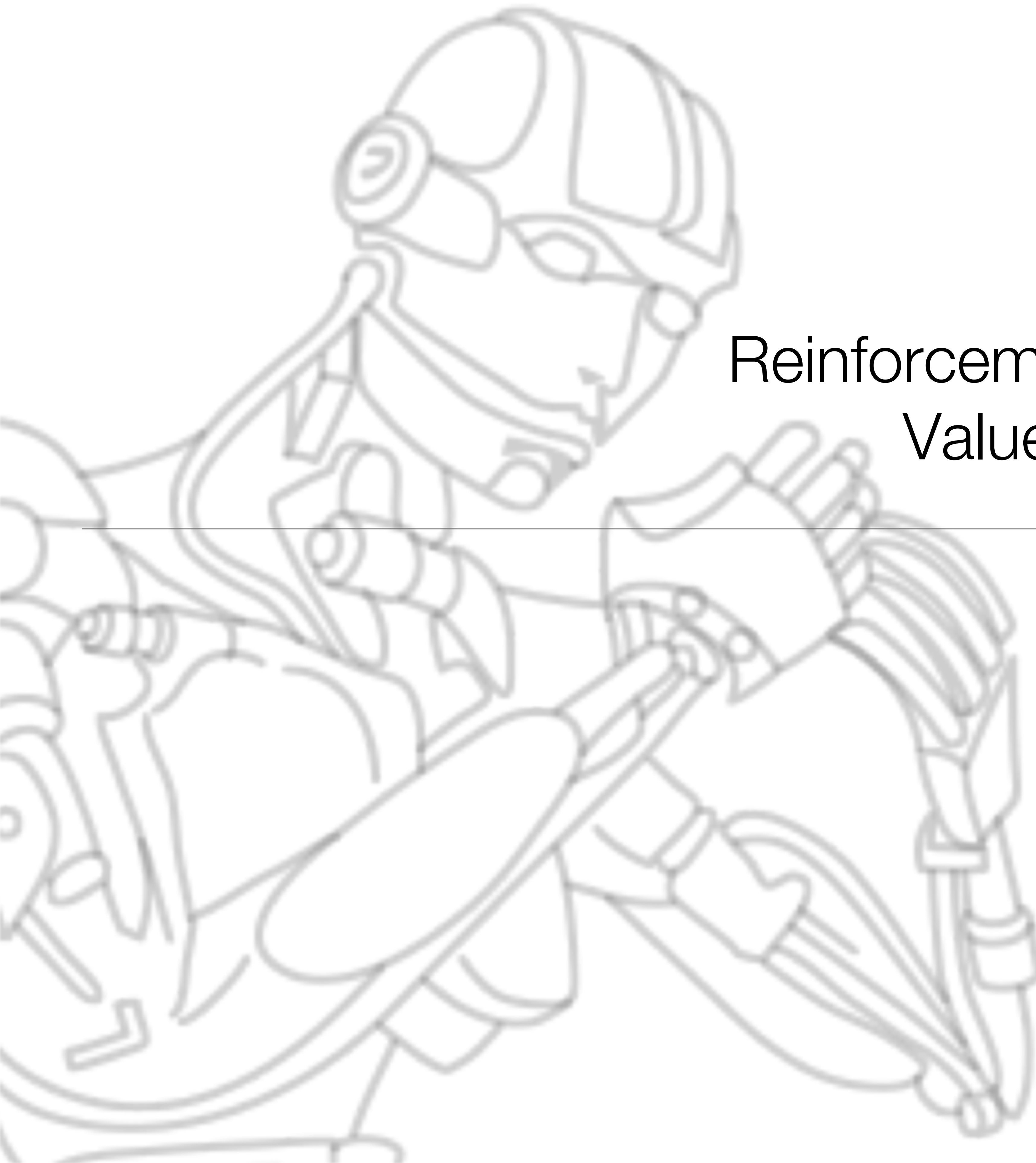
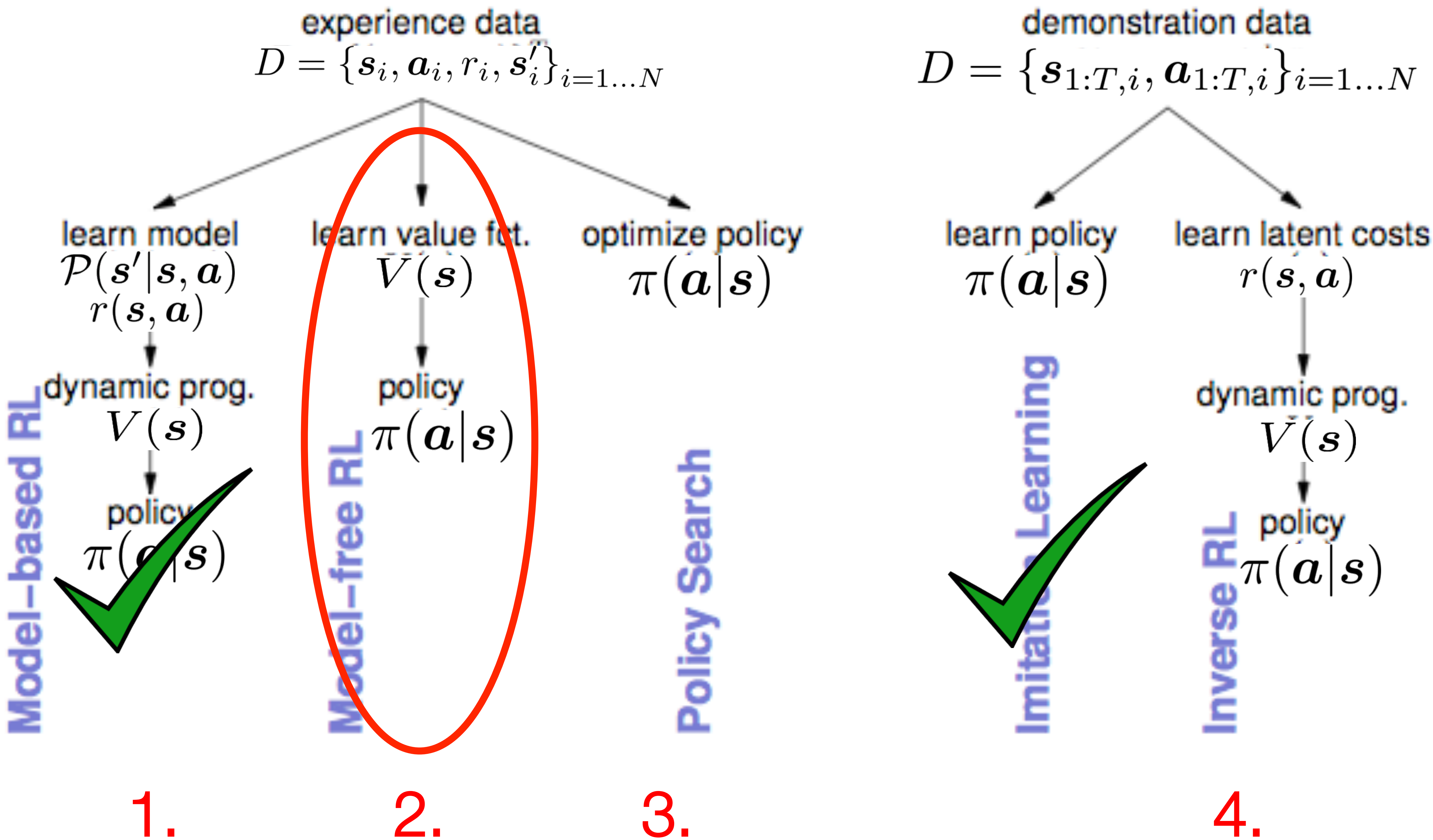


Reinforcement Learning Part 2
Value Function Methods

Jan Peters
Gerhard Neumann

The Bigger Picture: How to learn policies



Purpose of this Lecture

Often, learning a good model is too hard

- ➡The optimization inherent in optimal control is prone to model errors, as the controller may achieve the objective only because model errors get exploited
- ➡Optimal control methods based on linearization of the dynamics work only for moderately non-linear tasks
- ➡Model-free approaches are needed that do not make any assumption on the structure of the model

Classical Reinforcement Learning:

- ➡Solve the optimal control problem by learning the value function, not the model!

Outline of the Lecture

1. **Quick recap of dynamic programming**
2. **Reinforcement Learning with Temporal Differences**
3. **Value Function Approximation**
4. **Batch Reinforcement Learning Methods**
   Least-Squares Temporal Difference Learning
   Fitted Q-Iteration
5. **Robot Application:** Robot Soccer

Final Remarks

Markov Decision Processes (MDP)

Classical reinforcement learning is typically formulated for the infinite horizon objective

Infinite Horizon: maximize **discounted accumulated reward**

$$J_{\boldsymbol{\pi}} = \mathbb{E}_{\mu_0, \mathcal{P}, \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]$$

$0 \leq \gamma < 1$ … discount factor

Trades-off long term vs. immediate reward

Value functions and State-Action Value Functions

Refresher**:** Value function and state-action value function can be computed iteratively

$$V^\pi(\boldsymbol{s}) = \mathbb{E}_\pi\Big[r(\boldsymbol{s},\boldsymbol{a}) + \gamma \mathbb{E}_{\mathcal{P}}\left[V^\pi(\boldsymbol{s}')\right] \Big| \boldsymbol{s}\Big]$$

$$= \int \pi(\boldsymbol{a}|\boldsymbol{s})\Big(r(\boldsymbol{s},\boldsymbol{a}) + \gamma \int \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) V^\pi(\boldsymbol{s}') d\boldsymbol{s}'\Big) d\boldsymbol{a}$$

$$Q^\pi(\boldsymbol{s},\boldsymbol{a}) = r(\boldsymbol{s},\boldsymbol{a}) + \gamma \mathbb{E}_{\mathcal{P},\pi}\Big[Q^\pi(\boldsymbol{s}',\boldsymbol{a}') \Big| \boldsymbol{s},\boldsymbol{a}\Big]$$

$$= r(\boldsymbol{s},\boldsymbol{a}) + \gamma \int \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) \int \pi(\boldsymbol{a}'|\boldsymbol{s}') Q^\pi(\boldsymbol{s}',\boldsymbol{a}') d\boldsymbol{a}' d\boldsymbol{s}'$$

Finding an optimal value function

Bellman Equation of optimality

$$V^*(\boldsymbol{s}) = \max_{\boldsymbol{a}} \left(r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\mathcal{P}} \left[V^*(\boldsymbol{s}') \middle| \boldsymbol{s}, \boldsymbol{a} \right] \right)$$

⇨ Iterating the Bellman Equation converges to the optimal value function V^* and is called **value iteration**

Alternatively we can also **iterate Q-functions…**

$$Q^*(\boldsymbol{s}, \boldsymbol{a}) = r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\mathcal{P}} \left[\max_{\boldsymbol{a}'} Q^*(\boldsymbol{s}', \boldsymbol{a}') \middle| \boldsymbol{s}, \boldsymbol{a} \right]$$

Outline of the Lecture

1. Quick recap of dynamic programming

2. Reinforcement Learning with Temporal Differences

3. Value Function Approximation

4. Batch Reinforcement Learning Methods

Least-Squares Temporal Difference Learning

Fitted Q-Iteration

5. Robot Application: Robot Soccer

Final Remarks

Value-based Reinforcement Learning

Classical Reinforcement Learning

Updates the value function based on samples

$$\mathcal{D} = \{\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i\}_{i=1\ldots N}$$

We do not have a model and we do not want to learn it

Use the samples to update Q-function (or V-function)

Lets start simple:

Discrete states/actions ➡ Tabular Q-function

Temporal difference learning

Given a transition (s_t, a_t, r_t, s_{t+1}), we want to update the V-function

- Estimate of the current value: $V(s_t)$
- 1-step prediction of the current value: $\hat{V}(s_t) = r_t + \gamma V(s_{t+1})$
- 1-step prediction error (called temporal difference (TD) error)

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Update current value with the temporal difference error

$$V_{\text{new}}(s_t) = V(s_t) + \alpha \delta_t = (1 - \alpha) V(s_t) + \alpha (r_t + \gamma V(s_{t+1}))$$

Temporal difference learning

The **TD error**

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

compares the **one-time step lookahead prediction**

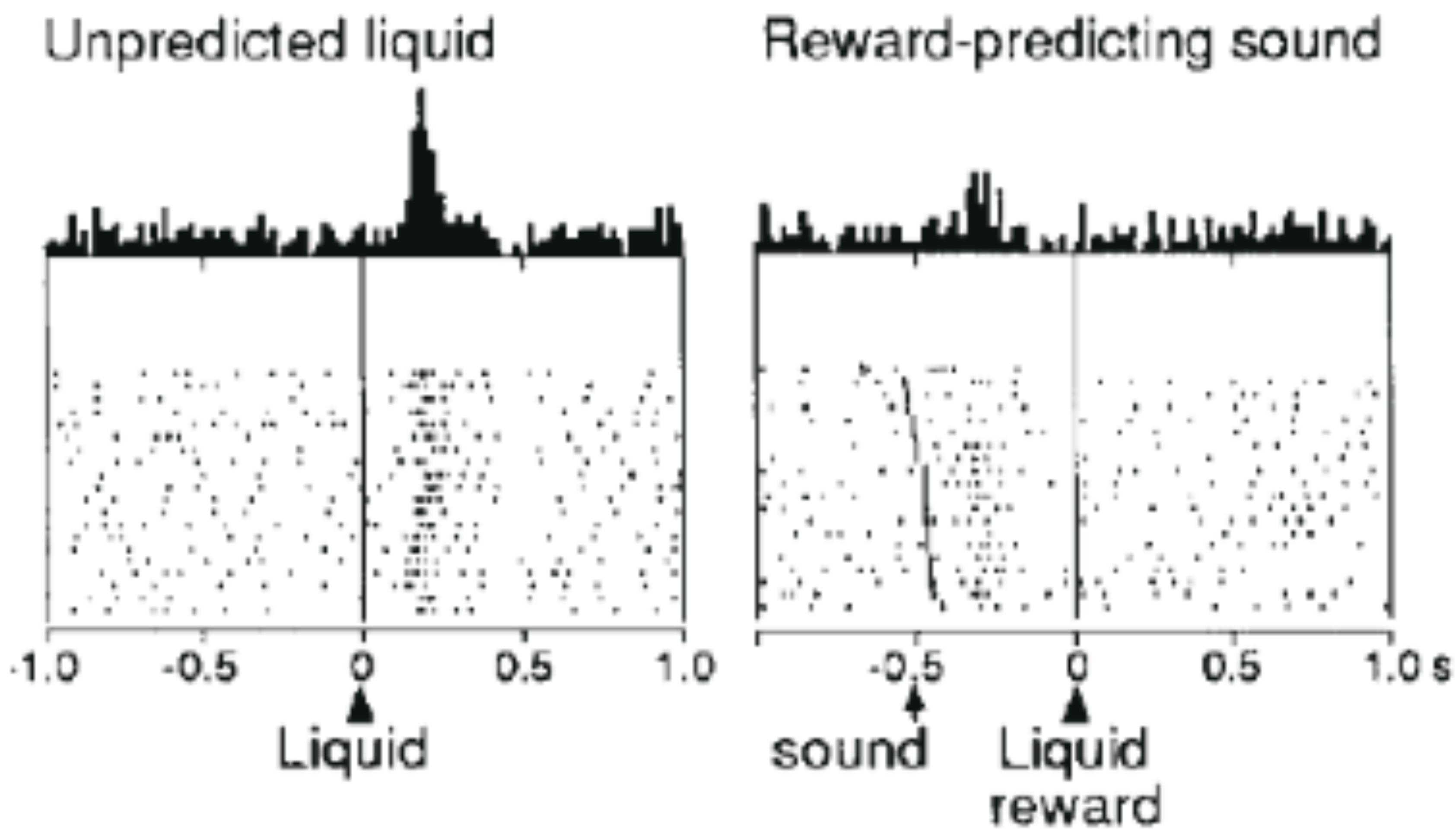
$$\hat{V}(s_t) = r_t + \gamma V(s_{t+1})$$

with the **current estimate** of the value function $V(s_t)$

- ⇨ if $\hat{V}(s_t) > V(s_t)$ than $V(s_t)$ is increased
- ⇨ if $\hat{V}(s_t) < V(s_t)$ than $V(s_t)$ is decreased

Dopamine as TD-error?

Temporal difference error signals can be measured in the brain of monkeys



Monkey brains seem to have it...

Algorithmic Description of TD Learning

Init: $V_0^*(s) \leftarrow 0$

Repeat $t = t + 1$

Observe transition (s_t, a_t, r_t, s_{t+1})

Compute TD error $\delta_t = r_t + \gamma V_t(s_{t+1}) - V_t(s_t)$

Update V-Function $V_{t+1}(s_t) = V_t(s_t) + \alpha\delta_t$

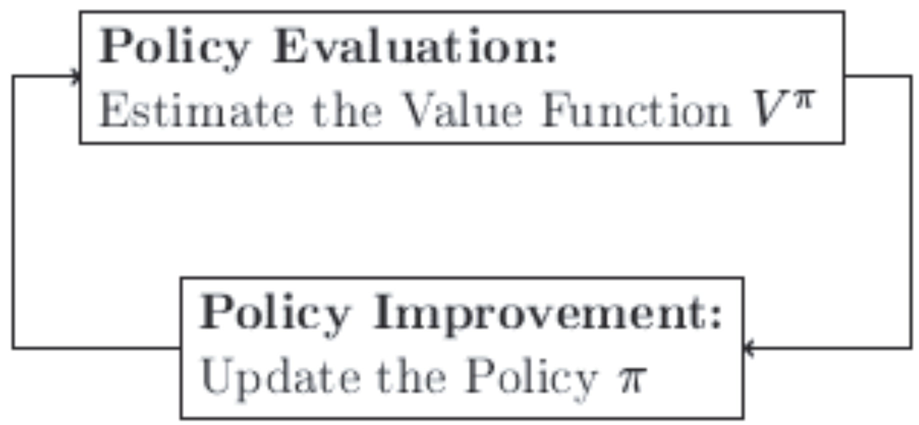
until convergence of V

➡ Used to compute Value function of behavior policy

➡ Sample-based version of policy evaluation

Temporal difference learning for control

So far: Policy evaluation with TD methods



Can **we also do the policy improvement step with samples**?

Yes, but we need to enforce exploration!

Epsilon-Greedy Policy: $$\pi(\boldsymbol{a}|\boldsymbol{s}) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}|, & \text{if } \boldsymbol{a} = \text{argmax}_{\boldsymbol{a}'} Q^\pi(\boldsymbol{s}, \boldsymbol{a}') \\ \epsilon/|\mathcal{A}, & \text{otherwise} \end{cases}$$

Soft-Max Policy: $$\pi(\boldsymbol{a}|\boldsymbol{s}) = \frac{\exp(\beta Q(\boldsymbol{s}, \boldsymbol{a}))}{\sum_{\boldsymbol{a}'} \exp(\beta Q(\boldsymbol{s}, \boldsymbol{a}'))}$$

➡ **Do not always take greedy action**

Temporal difference learning for control

Update equations for learning the Q-function $Q(s, a)$

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\delta_t, \quad \delta_t = r_t + \gamma Q_t(s_{t+1}, a_?) - Q_t(s_t, a_t)$$

Two different methods to estimate $a_?$

Q-learning: $a_? = \mathrm{argmax}_a Q_t(s_{t+1}, a)$

Estimates Q-function of optimal policy

Off-policy samples: $a_? \neq a_{t+1}$

SARSA: $a_? = a_{t+1}$ **,** where $a_{t+1} \sim \pi(a|s_{t+1})$

Estimates Q-function of exploration policy

On-policy samples

Note: The policy for generating the actions depends on the Q-function ➡ non-stationary policy

Outline of the Lecture

1. Quick recap of dynamic programming

2. Reinforcement Learning with Temporal Differences

3. Value Function Approximation

4. Batch Reinforcement Learning Methods

Least-Squares Temporal Difference Learning

Fitted Q-Iteration

5. Robot Application: Robot Soccer

Final Remarks

Approximating the Value Function

In the continuous case, we need to approximate the V-function (except for LQR)

Lets keep it simple, we use **a linear model** to represent the V-function

$$V^{\pi}(\boldsymbol{s}) \approx V_{\boldsymbol{\omega}}(\boldsymbol{s}) = \boldsymbol{\phi}^T(\boldsymbol{s})\boldsymbol{\omega}$$

How can we find the parameters ω ?

➡ Again with Temporal Difference Learning

TD-learning with Function Approximation

Derivation:

Use the **recursive definition of V-function**:

$$\mathrm{MSE}(\boldsymbol{\omega}) \approx \mathrm{MSE}_{\mathrm{BS}}(\boldsymbol{\omega}) = 1/N \sum_{i=1}^{N} \left(\hat{V}^{\pi}(\boldsymbol{s}_i) - V_{\boldsymbol{\omega}}(\boldsymbol{s}_i) \right)^2$$

with $\hat{V}^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi}\left[r(\boldsymbol{s}, \boldsymbol{a}) + \mathbb{E}_{\mathcal{P}}\left[V_{\boldsymbol{\omega}_{\mathrm{old}}}(\boldsymbol{s}') | \boldsymbol{s}, \boldsymbol{a} \right] \right]$

⇨ **Bootstrapping (BS):** Use the old approximation to get the target values for a new approximation

How can we **minimize** this function ?

Lets use **stochastic gradient descent**

Refresher: Stochastic Gradient Descent

Consider an **expected error function,**

$$E_{\boldsymbol{\omega}} = \mathbb{E}_p[e_{\boldsymbol{\omega}}(x)] \approx 1/N \sum_{i=1}^{N} e_{\boldsymbol{\omega}}(x_i), \quad x_i \sim p(x)$$

We can find a local minimum of E by **Gradient descent:**

$$\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k - \alpha_k \frac{dE_{\boldsymbol{\omega}}}{d\boldsymbol{\omega}} = \boldsymbol{\omega}_k - \alpha_k \sum_{i=1}^{N} \frac{de_{\boldsymbol{\omega}}(x_i)}{d\boldsymbol{\omega}}$$

Stochastic Gradient Descent does the gradient update already after a **single sample**

$$\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k - \alpha_k \frac{de_{\boldsymbol{\omega}}(x_k)}{d\boldsymbol{\omega}}$$

Converges under the stochastic approximation conditions

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

Temporal difference learning

Stochastic gradient descent on our error function MSE_{BS}

$$\mathrm{MSE}_{\mathrm{BS,t}}(\boldsymbol{\omega}) = 1/N \sum_{i=1}^{N} \left(\hat{V}(\boldsymbol{s}_t) - V_{\boldsymbol{\omega}}(\boldsymbol{s}_i)\right)^2$$

$$= 1/N \sum_{i=1}^{N} \left(r_i + \gamma V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_i') - V_{\boldsymbol{\omega}}(\boldsymbol{s}_i)\right)^2$$

Update rule (for current time step t, $V_{\boldsymbol{\omega}}(\boldsymbol{s}) = \boldsymbol{\phi}^T(\boldsymbol{s})\boldsymbol{\omega}$)

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_t \left. \frac{d\mathrm{MSE}_{\mathrm{BS}}}{d\boldsymbol{\omega}} \right|_{\boldsymbol{\omega}=\boldsymbol{\omega}_t}$$

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha \Big(r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_{t+1}) - V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_t) \Big) \boldsymbol{\phi}^T(\boldsymbol{s}_t)$$

$$= \boldsymbol{\omega}_t + \alpha \delta_t \boldsymbol{\phi}^T(\boldsymbol{s}_t)$$

with $\delta_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_{t+1}) - V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_t)$

Temporal difference learning

TD with function approximation

$$\boldsymbol{\omega}_t = \boldsymbol{\omega}_t + \alpha \delta_t \boldsymbol{\phi}^T(\boldsymbol{s}_t)$$

Difference to discrete algorithm:

- TD-error is correlated with the feature vector
- Equivalent if tabular feature coding is used, i.e., $\phi(s_i) = \boldsymbol{e}_i$

Similar update rules can be obtained for SARSA and Q-learning

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha \Big(r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma Q_{\boldsymbol{\omega}_t}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_?) - Q_{\boldsymbol{\omega}_t}(\boldsymbol{s}_t, \boldsymbol{a}_t) \Big) \boldsymbol{\phi}^T(\boldsymbol{s}_t, \boldsymbol{a}_t)$$

where $Q_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}) \approx \boldsymbol{\phi}^T(\boldsymbol{s}, \boldsymbol{a}) \boldsymbol{\omega}$

Temporal difference learning

Some remarks on temporal difference learning:

- Its **not a proper** stochastic gradient descent!!
- **Why?** Target values $\hat{V}^{\pi}(\boldsymbol{s})$ **change after each parameter update!**

  We ignore the fact that $\hat{V}^{\pi}(\boldsymbol{s})$ also depends on $\boldsymbol{\omega}$
- **Side note:** This „ignorance“ actually introduces a bias in our optimization, such that we are optimizing a different objective than the *MSE*
- In certain cases, we also get **divergence** (e.g. off-policy samples)
- TD-learning is very fast in terms of computation time O(#features), but not data-efficient ⇨ **each sample is just used once!**

Dann, Neumann, Peters: *Policy Evaluation with Temporal Differences: A survey and comparison*, JMLR, in press

Sucessful examples

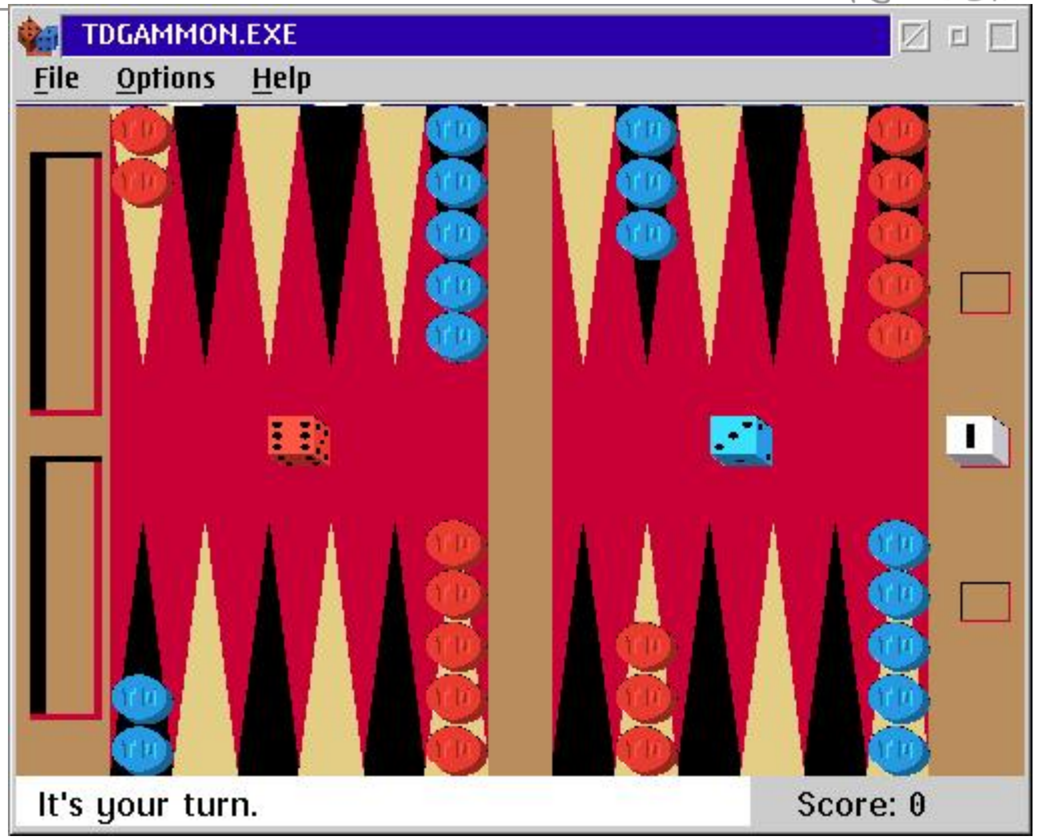
Linear function approximation

Tetris, Go

Non-linear function approximation

TD Gammon (Worldchampion level)

Atari Games (learning from raw pixel input)



Outline of the Lecture

1. Quick recap of dynamic programming

2. Value function approximation

3. Reinforcement Learning with Temporal Differences

4. Batch Reinforcement Learning Methods

Least-Squares Temporal Difference Learning

Fitted Q-Iteration

5. Robot Application: Robot Soccer

Final Remarks

Batch-Mode Reinforcement Learning

Online methods are typically **data-inefficient** as they use each data point only once

$$D = \left\{ \boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i \right\}_{i=1\ldots N}$$

Can we **re-use the whole „batch“** of data to **increase data-efficiency?**

- **Least-Squares Temporal Difference (LSTD) Learning**
- **Fitted Q-Iteration**

⇨ Computationally **much more expensive** then TD-learning!

Least-Squares Temporal Difference (LSTD)

Lets **minimize the bootstrapped *MSE*** objective (*MSE*$_{BS}$)

$$\mathrm{MSE_{BS}} = 1/N \sum_{i=1}^{N} \left(r(\boldsymbol{s}_i, \boldsymbol{a}_i) + \gamma V_{\boldsymbol{\omega}_{\mathrm{old}}}(\boldsymbol{s}_i') - V_{\boldsymbol{\omega}}(\boldsymbol{s}_i) \right)^2$$

$$= 1/N \sum_{i=1}^{N} \left(r(\boldsymbol{s}_i, \boldsymbol{a}_i) + \gamma \boldsymbol{\phi}^T(\boldsymbol{s}_i') \boldsymbol{\omega}_{\mathrm{old}} - \boldsymbol{\phi}^T(\boldsymbol{s}_i) \boldsymbol{\omega} \right)^2$$

Least-Squares Solution:

$$\boldsymbol{\omega} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T (\boldsymbol{R} + \gamma \boldsymbol{\Phi}' \boldsymbol{\omega}_{\mathrm{old}})$$

with $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\boldsymbol{s}_1), \boldsymbol{\phi}(\boldsymbol{s}_2), \dots, \boldsymbol{\phi}(\boldsymbol{s}_N)]^T$

$$\boldsymbol{\Phi}' = [\boldsymbol{\phi}(\boldsymbol{s}_1'), \boldsymbol{\phi}(\boldsymbol{s}_2'), \dots, \boldsymbol{\phi}(\boldsymbol{s}_N')]^T$$

Least-Squares Temporal Difference (LSTD)

Least-Squares Solution:

$$\boldsymbol{\omega} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T(\boldsymbol{R} + \gamma\boldsymbol{\Phi}'\boldsymbol{\omega}_{\text{old}})$$

Fixed Point: In case of convergence, we want to have $\boldsymbol{\omega}_{\text{old}} = \boldsymbol{\omega}$

$$\begin{aligned}
\boldsymbol{\omega} &= (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T(\boldsymbol{R} + \gamma\boldsymbol{\Phi}'\boldsymbol{\omega}) \\
\left(\boldsymbol{I} - \gamma(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{\Phi}'\right)\boldsymbol{\omega} &= (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{R} \\
(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\left(\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}'\right)\boldsymbol{\omega} &= (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{R} \\
\boldsymbol{\Phi}^T\left(\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}'\right)\boldsymbol{\omega} &= \boldsymbol{\Phi}^T\boldsymbol{R}
\end{aligned}$$

$$\boldsymbol{\omega} = \left(\boldsymbol{\Phi}^T(\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}')\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{R}$$

Least-Squares Temporal Difference (LSTD)

LSTD solution:

$$\boldsymbol{\omega} = \left(\boldsymbol{\Phi}^T(\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}')\right)^{-1}\boldsymbol{\Phi}^T\boldsymbol{R}$$

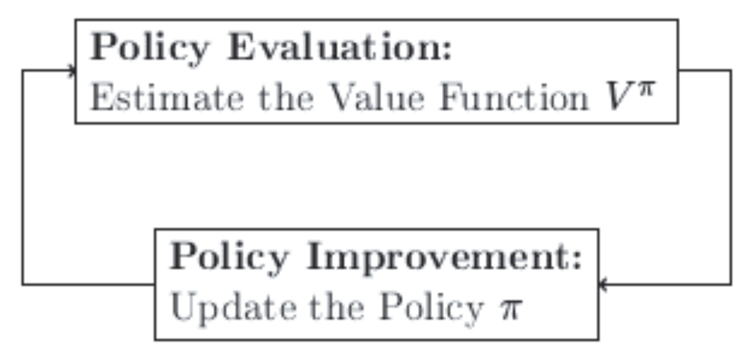
Same solution as convergence point of **TD-learning**

One shot! **No iterations** necessary for policy evaluation

LSQ: Adaptation for learning the Q-function

$$\boldsymbol{\Phi} = [\phi(\boldsymbol{s}_1, \boldsymbol{a}_1), \phi(\boldsymbol{s}_2, \boldsymbol{a}_2), \ldots, \phi(\boldsymbol{s}_N, \boldsymbol{a}_N)]^T$$

$$\boldsymbol{\Phi}' = [\phi(\boldsymbol{s}_2, \boldsymbol{a}_2), \phi(\boldsymbol{s}_3, \boldsymbol{a}_3), \ldots, \phi(\boldsymbol{s}_{N+1}, \boldsymbol{a}_{N+1})]^T$$



⇨ Used for **Least-Squares Policy Iteration (LSPI)**

Lagoudakis and Parr, ***Least-Squares Policy Iteration,*** *JMLR*

Learning to Ride a Bicycle

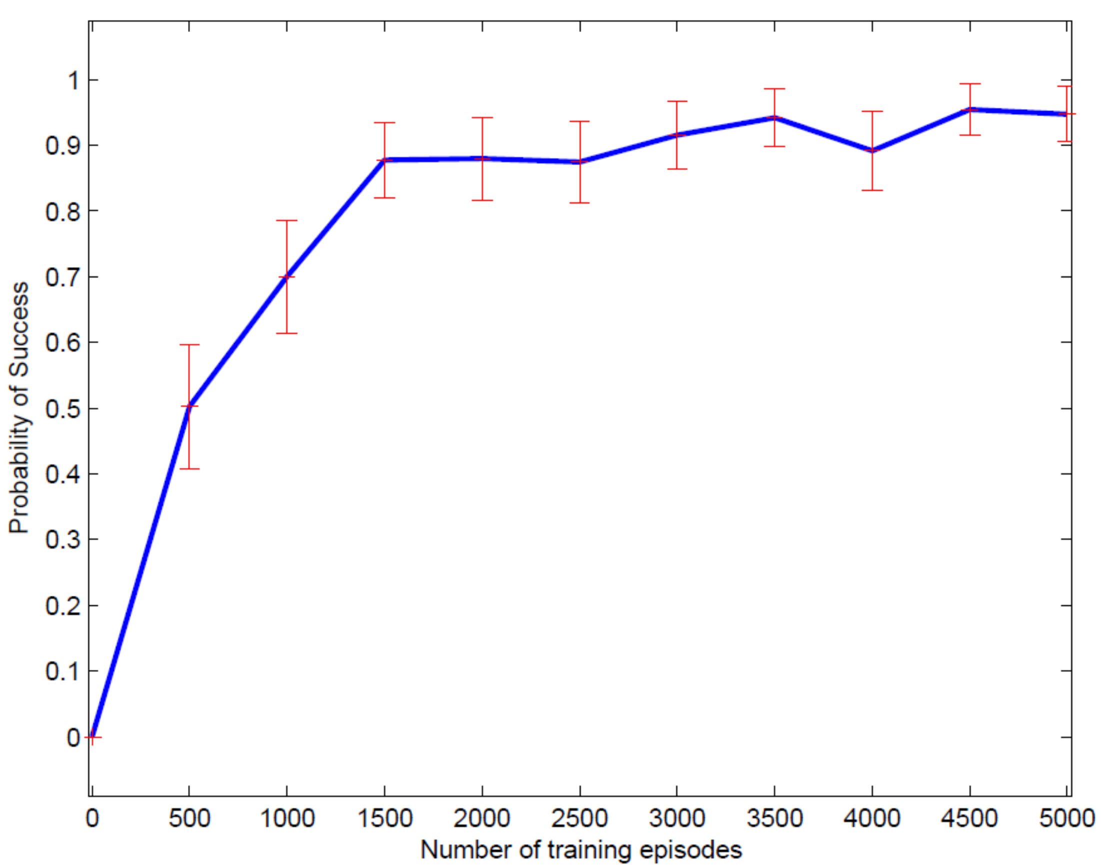
State space: $s = [\theta, \dot{\theta}, \omega, \dot{\omega}, \ddot{\omega}, \psi]$

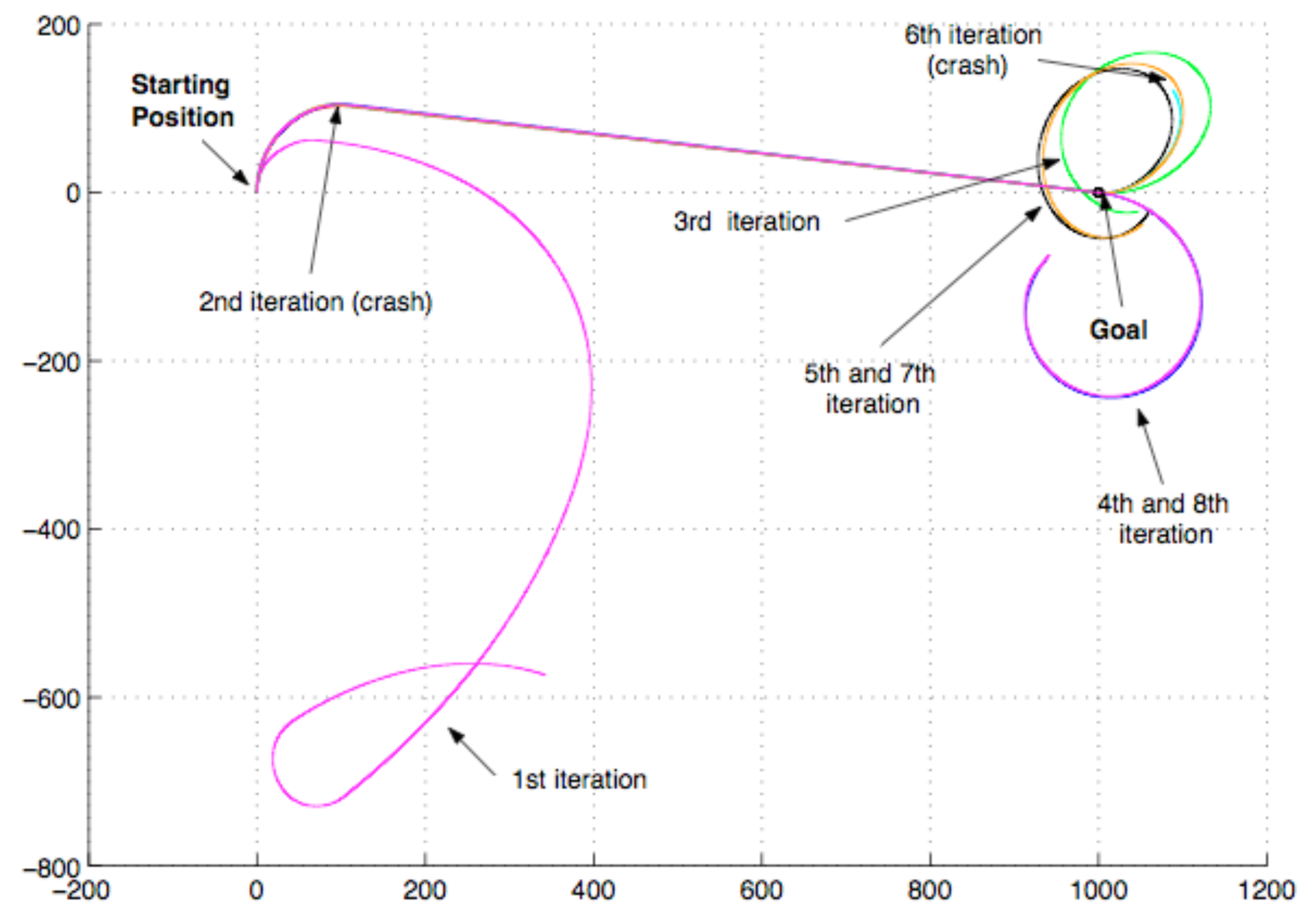
θ angle of handlebar, ω vertical angle of bike, ψ angle to goal

Action space: 5 discrete actions (torque applied to handle, displacement of rider)

Feature space: 20 basis functions…

$$(\ 1,\ \omega,\ \dot{\omega},\ \omega^2,\ \dot{\omega}^2, \omega\dot{\omega},\ \theta,\ \dot{\theta},\ \theta^2,\ \dot{\theta}^2,\ \theta\dot{\theta},\ \omega\theta,\ \omega\theta^2,\ \omega^2\theta,\ \psi,\ \psi^2,\ \psi\theta,\ \bar{\psi},\ \bar{\psi}^2,\ \bar{\psi}\theta\ )^{\mathsf{T}}$$





Fitted Q-iteration

In Batch-Mode RL it is also much easier to use **non-linear function approximators**

- Many of them **only exists in the batch setup**, e.g. regression trees
- **No catastrophic forgetting**, e.g., for neural networks.
- Strong divergence problems, fixed for Neural Networks by ensuring that there is a goal state where the Q-Function value is always zero (see Lange et al. below).

Fitted Q-iteration uses non-linear function approximators for **approximate value iteration.**

Ernst, Geurts and Wehenkel, *Tree-Based Batch Mode Reinforcement Learning, JMLR 2005*
Lange, Gabel and Riedmiller. *Batch Reinforcement Learning*, *Reinforcement Learning: State of the Art*

Fitted Q-iteration

Given: Dataset $D = \left\{ \boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i \right\}_{i=1\ldots N}$

Algorithm:

Initialize $Q^{[0]}(\boldsymbol{s}, \boldsymbol{a}) = 0$, input data: $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{s}_1^T & \boldsymbol{a}_1^T \\ \vdots & \\ \boldsymbol{s}_N^T & \boldsymbol{a}_N^T \end{bmatrix}$

for k = 1 to L

Generate target values: $\tilde{q}_i^{[k]} = r_i + \gamma \max_{\boldsymbol{a}'} Q^{[k-1]}(\boldsymbol{s}'_i, \boldsymbol{a}')$

Learn new Q-function: $Q^{[k]}(\boldsymbol{s}, \boldsymbol{a}) \leftarrow \mathrm{Regress}(\boldsymbol{X}, \tilde{\boldsymbol{q}}^{[k]})$

end

➡ Like Value-Iteration, but we use supervised learning methods to approximate the Q-function at each iteration k

Fitted Q-iteration

Some Remarks:

- Regression does the **expectation** for us

$$Q^{[k]}(\boldsymbol{s}, \boldsymbol{a}) \approx \mathbb{E}_{\mathcal{P}}\left[r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \max_{\boldsymbol{a}'} Q^{[k-1]}(\boldsymbol{s}', \boldsymbol{a}')\right]$$

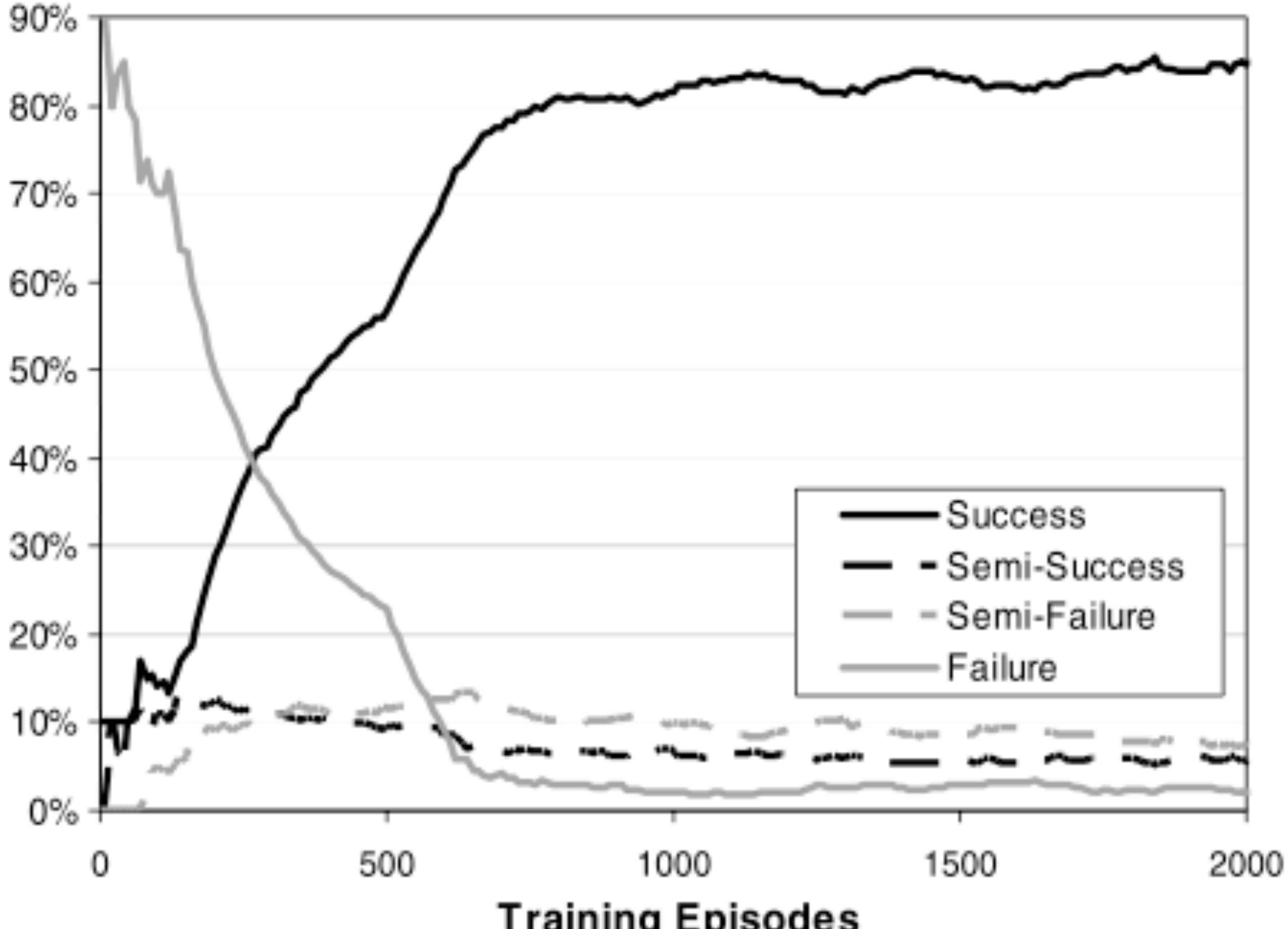
- The max operator is still **hard to solve for continous action spaces**

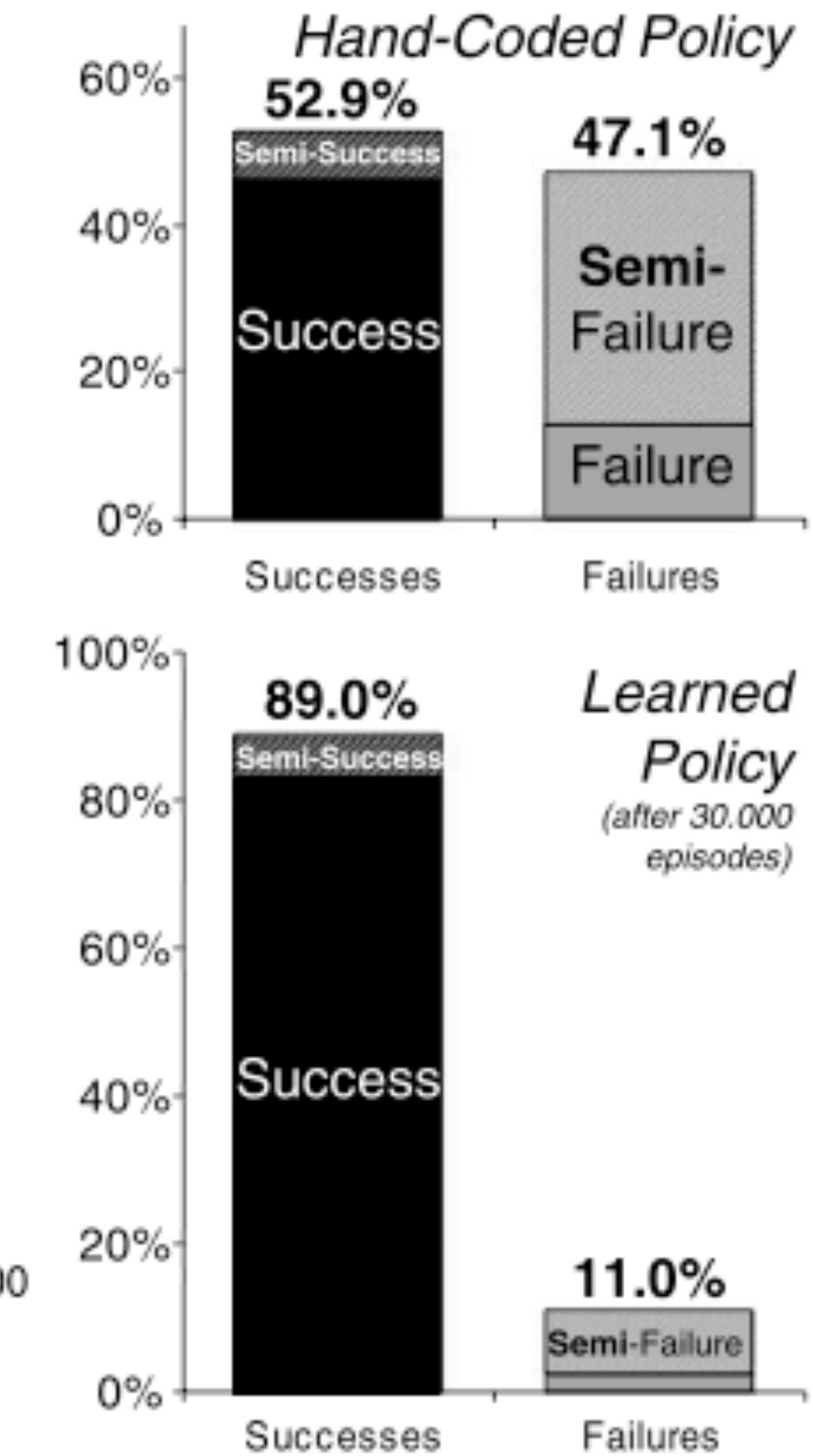
For continuous actions, see: Neumann and Peters, *Fitted Q-iteration by Advantage weighted regression*, NIPS, 2008

Case Study I: Learning Defense

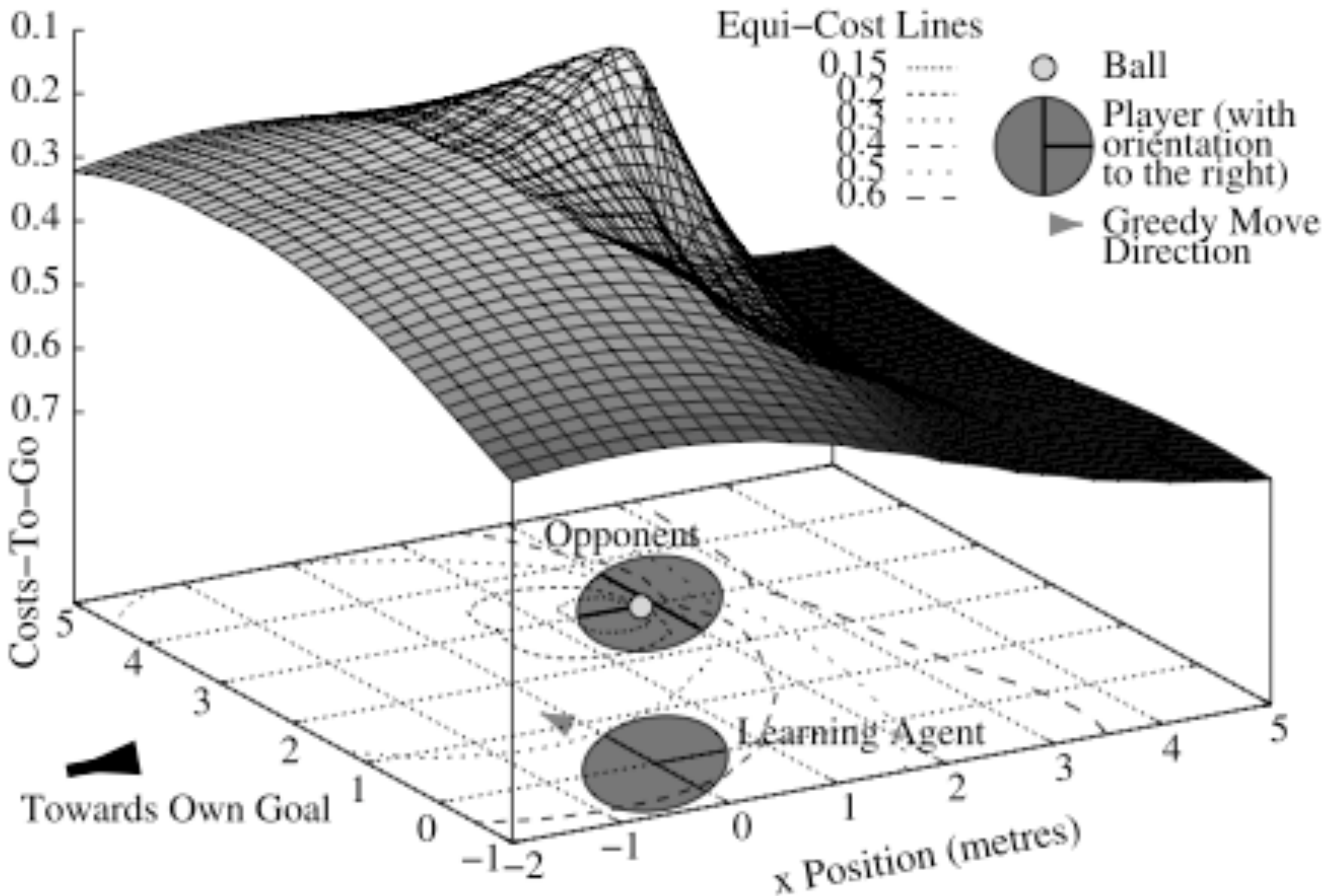
Within the RL framework, we model the ADB learning task as a terminal state problem with both terminal goal S^+ and failure states S^-. Intermediate steps are punished by constant costs of $c = 0.05$, whereas $J(s) = 0.0$ for $s \in S^+$ and $J(s) = 1.0$ for $s \in S^-$ by definition (cf. Eq. 8).

Success

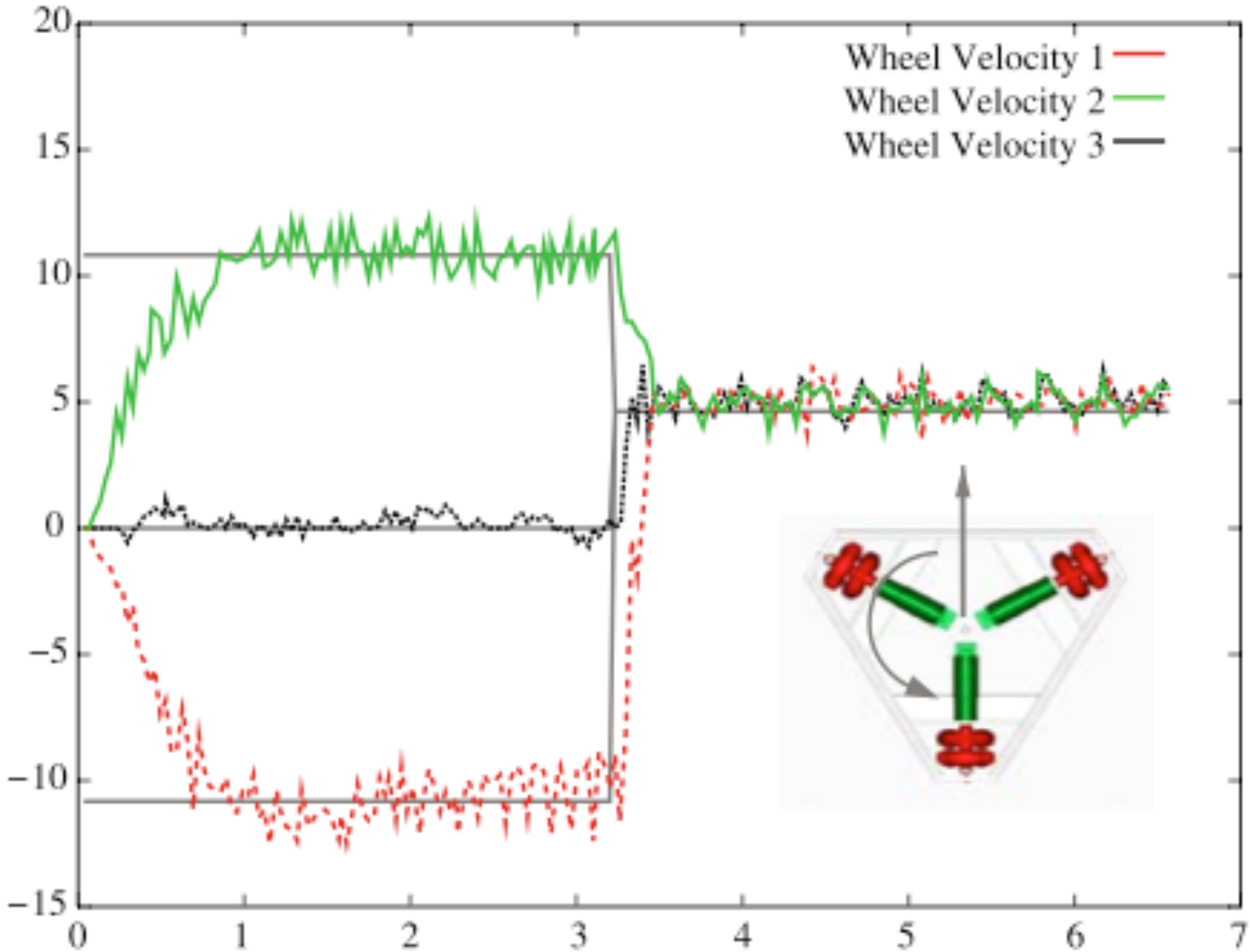

90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
500
1000
1500
2000
Training Episodes
Success
Semi-Success
Semi-Failure
Failure


Hand-Coded Policy
60%
40%
20%
0%
52.9%
Semi-Success
Success
47.1%
Semi-
Failure
Failure
Successes
Failures
Learned Policy
(after 30.000 episodes)
100%
80%
60%
40%
20%
0%
89.0%
Semi-Success
Success
11.0%
Semi-Failure
Successes
Failures

Dueling Behavior

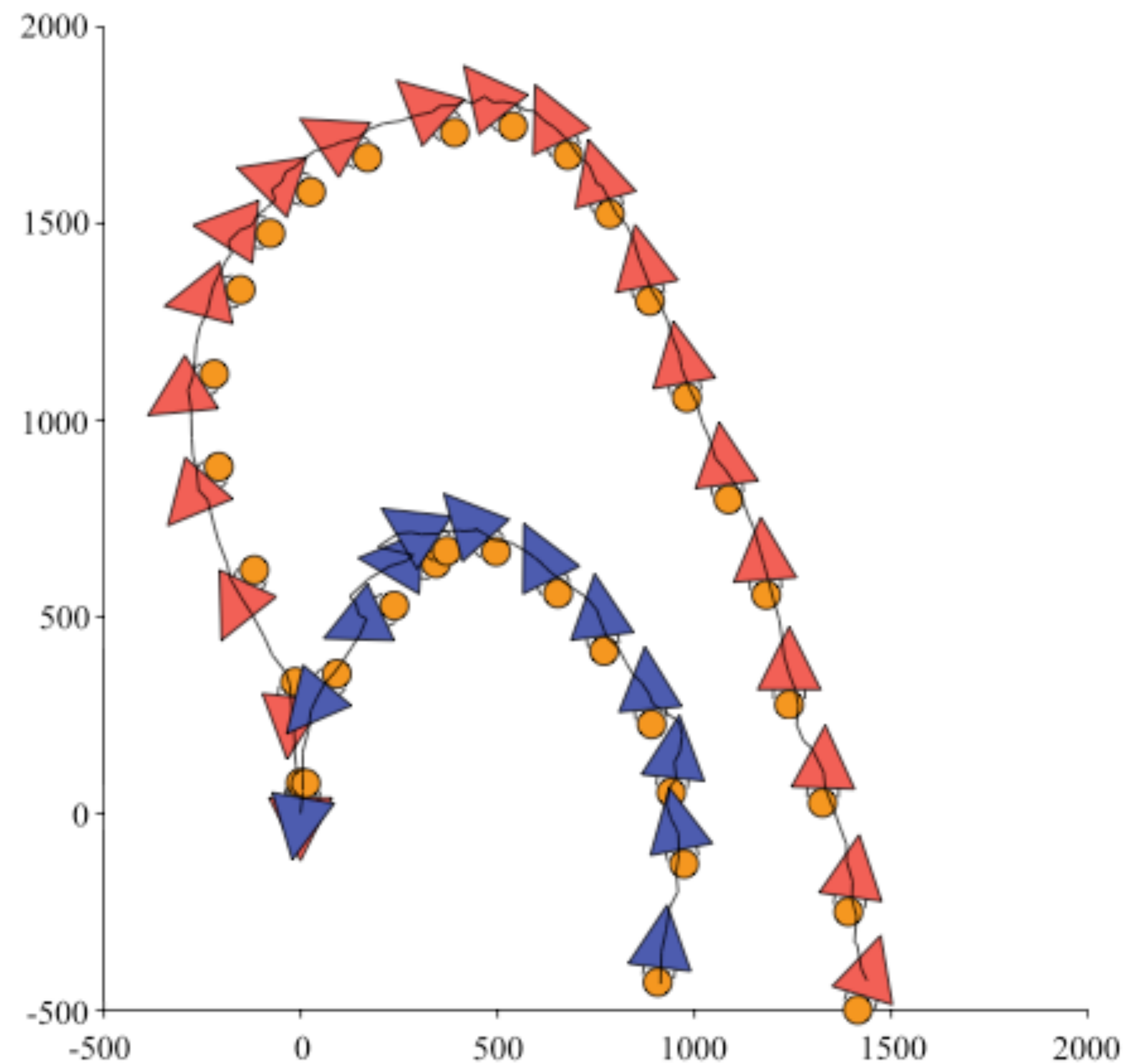


Case Study II: Learning Motor Speeds



$$c(s, a, s') = c(s) = \begin{cases} 0 & \text{if } |\dot{\omega}_d - \dot{\omega}| < \delta, \\ 0.01 & \text{else.} \end{cases}$$

Case Study III: Learning to Dribble



$$Q^{target}(s,a) := \begin{cases} 1.0, & \text{if } s' \in S^-, \\ 0.01, & \text{if } s' \in S^+, \\ 0.01 + \min_b \tilde{Q}(s',b), & \text{else} \end{cases}$$

Value Function Methods

- ... have been the driving reinforcement learning approach in the 1990s.
- You can do loads of cool things with them: Learn Chess at professional level, learn **Backgammon and Checkers at Grandmaster-Level** ... and winning the **Robot Soccer Cup** with a minimum of man power.

So, why are they not always the method of choice?

- You need to fill-up you state-action space up with sufficient samples.
- Another curse of dimensionality with an exponential explosion.
- Errors in the Value function approximation might have a catastrophic effect on the policy, can be very hard to control
- However, it scales better as we only need samples at relevant locations.