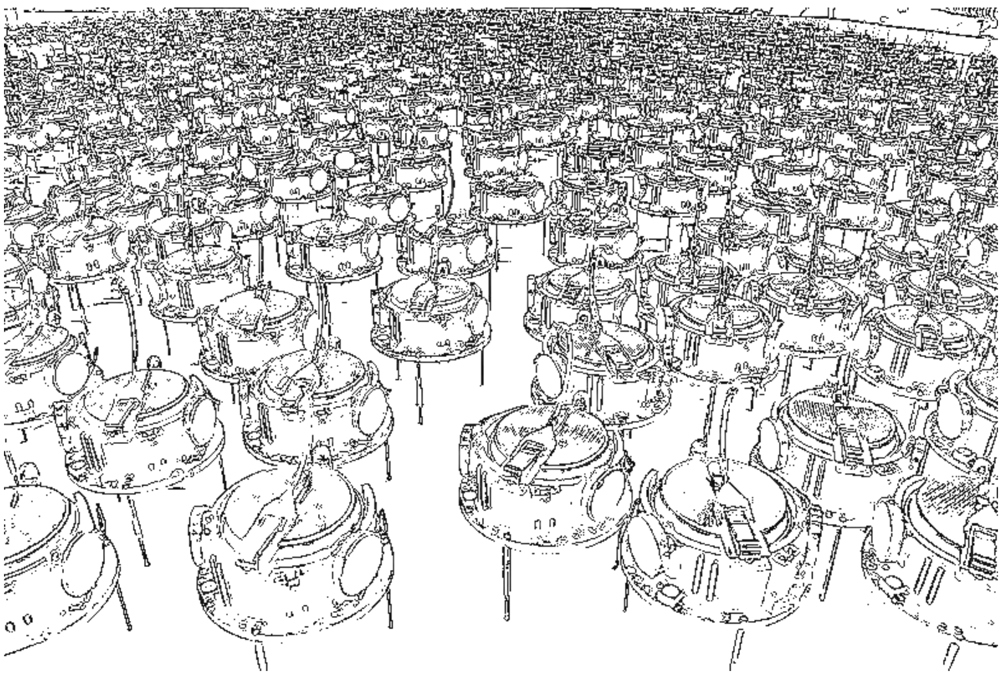
Intelligent Multi Agent Systems



Single Agent Reinforcement Learning

Gerhard Neumann

Agenda

- ➡ So far, we have only considered simple games with no/very simple state dynamics
- ➡ In a realistic setup, we have the agent's living and interacting with a complex environment
- ➡ We want to maximize the future reward considering future states of the agents
- ➡ Can be formulated as a Markov Decision Process (MDP)
- ➡ Learning in an MDP: Reinforcement Learning

Today: We will start with a single agent

Outline

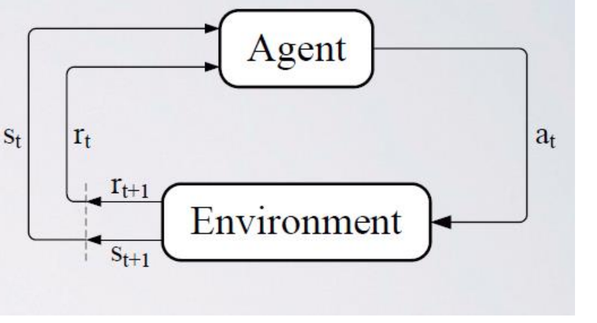
Known Model: Markov Decision Processes

- ➡ Value Functions and Policy Evaluation
- ➡ Policy Iteration and Value Iteration

Unknown Model: Reinforcement Learning

- ➡ Temporal Difference Learning
- ➡ TD-Learning for Control: Q-Learning and Sarsa
- ➡ Value Function Approximation
- ➡ Least-Squares Temporal Difference Learning

Markov Decision Processes (MDP)



A MDP is defined by:

- its state space $\boldsymbol{s} \in \mathcal{S}$
- its action space $\boldsymbol{a} \in \mathcal{A}$
- its transition dynamics $\mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$
- its reward function $r(\boldsymbol{s}, \boldsymbol{a})$
- and its initial state probabilities $\mu_0(\boldsymbol{s})$

Markov property:

$$\mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \dots) = \mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$$

- Transition dynamics depends on only of current time step

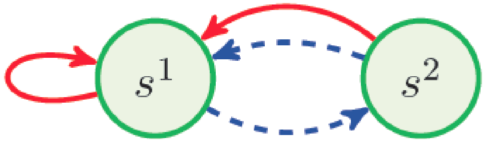
Optimality Objective

The goal of the agent is to find an optimal policy π^* that maximizes its expected long term reward J_π

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

- $0 \leq \gamma < 1$ … discount factor
- Discount Factor trades-off long term vs. immediate reward
- **Time Horizon**: Infinite

Example: Two State Problem



States: s^1, s^2

Actions: red (a^1) and blue (a^2) edges

Transition:

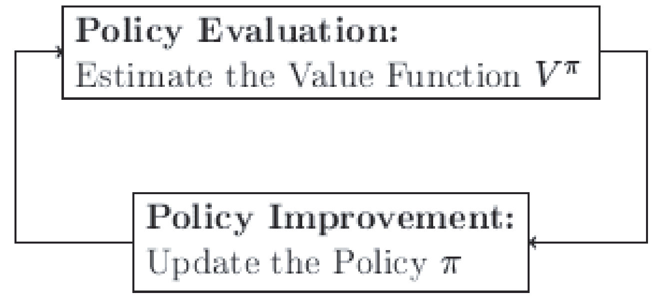
$$\mathcal{P}(s^1|s^1,a^1)=1,\ \mathcal{P}(s^2|s^1,a^1)=0,\ \mathcal{P}(s^1|s^1,a^2)=0,\ \mathcal{P}(s^2|s^1,a^2)=1$$

$$\mathcal{P}(s^1|s^2,a^1)=1,\ \mathcal{P}(s^2|s^2,a^1)=0,\ \mathcal{P}(s^1|s^2,a^2)=1,\ \mathcal{P}(s^2|s^2,a^2)=0$$

Rewards: $r(s^1)=1,\ r(s^2)=0$

Policy: What is the optimal policy?

How do we find an optimal policy?



Typically done iteratively:

1. Policy Evaluation:

Estimate quality of states (and actions) with current policy

2. Policy Improvement:

Improve policy by taking actions with the highest quality

This algorithm is called **Policy Iteration**

Value functions

Value function $V^\pi(\boldsymbol{s})$ **:**

Expected long-term reward when beeing in state $\boldsymbol{s}$ and following policy $\pi(\boldsymbol{a}|\boldsymbol{s})$

$$V^\pi(\boldsymbol{s}) = E_{\mathcal{P},\pi}\Big[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) | \boldsymbol{s}_0 = \boldsymbol{s}\Big]$$

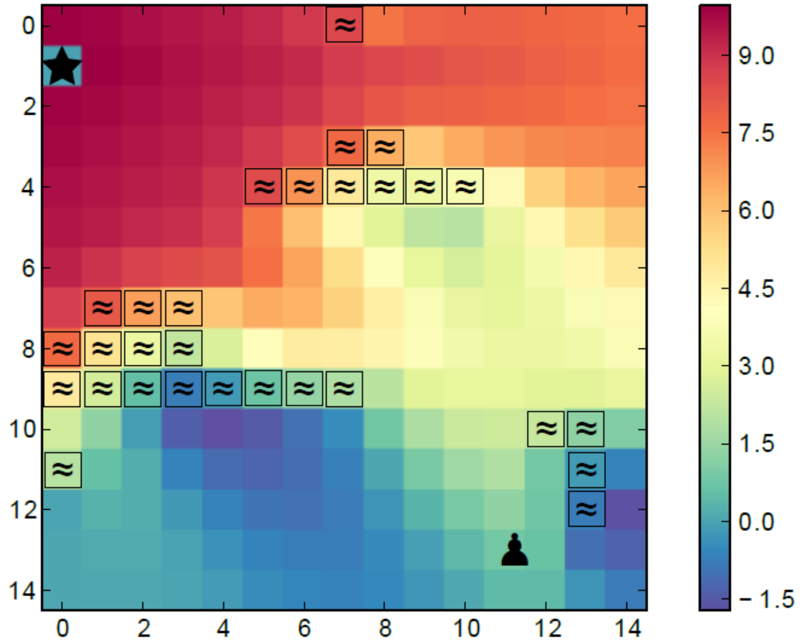
V(s) is a Quality measure for state $\boldsymbol{s}$

➡ „How good“ is it to be in state s under policy $\pi(\boldsymbol{a}|\boldsymbol{s})$?

Value functions

Illustration:

Policy always goes directly to the star
Going through puddles is punished



State-Action Value Functions

Q-function $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$:

Long-term reward for taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$ and subsequently following policy $\pi(\boldsymbol{a}|\boldsymbol{s})$

$$Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\mathcal{P},\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) | \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}\right]$$

Q(s,a) is a quality measure for taking action a in state a

➡ „How good" is it to take action a in state s under policy $\pi(\boldsymbol{a}|\boldsymbol{s})$?

Value Functions

They can be computed from each other

- Computing V-Function from Q-Function

$$V^\pi(\boldsymbol{s}) = \mathbb{E}_\pi\left[Q^\pi(\boldsymbol{s},\boldsymbol{a})|\boldsymbol{s}\right] = \int \pi(\boldsymbol{a}|\boldsymbol{s})Q^\pi(\boldsymbol{s},\boldsymbol{a})d\boldsymbol{a}$$

  - Expectation over the policy

- Computing Q-Function from V-Function

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

  - Uses value of the next state

Recursive Updates

… both functions can also be estimated recursively

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

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

➡ If I know the value of the next state $\boldsymbol{s}'$, I can compute the value of the current state

Iterating these equations converges to the true V or Q function

Policy Evaluation Algorithm

Algorithm (for discrete states)

Init: $V_0^\pi(s) \leftarrow 0, \forall s \text{ and } k = 0$

Repeat

Compute Q-Function (for each state action pair)

$$Q_{k+1}^\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k^\pi(s')$$

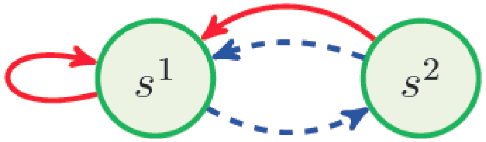
Compute V-Function (for each state)

$$V_{k+1}^\pi(s) = \sum_a \pi(a|s) Q_{k+1}^\pi(s,a)$$

$$k = k + 1$$

until convergence

Example: Two State Problem



States: s^1, s^2

Actions: red (a^1) and blue (a^2) edges

Transition:

$$\mathcal{P}(s^1|s^1,a^1)=1,\ \mathcal{P}(s^2|s^1,a^1)=0,\ \mathcal{P}(s^1|s^1,a^2)=0,\ \mathcal{P}(s^2|s^1,a^2)=1$$

$$\mathcal{P}(s^1|s^2,a^1)=1,\ \mathcal{P}(s^2|s^2,a^1)=0,\ \mathcal{P}(s^1|s^2,a^2)=1,\ \mathcal{P}(s^2|s^2,a^2)=0$$

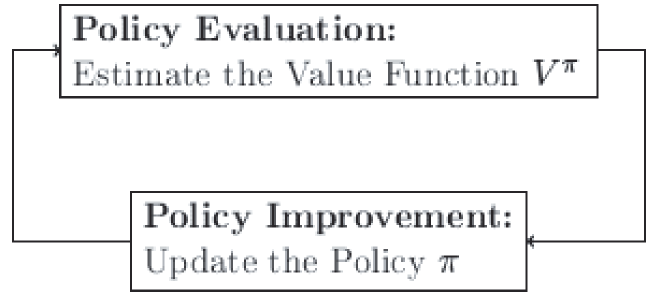
Rewards: $r(s^1)=1,\ r(s^2)=0$

Policy Evaluation: Policy $\pi(a|s)=0.5, \forall s, a$

Compute V(s) and Q(s,a): Blackboard

How do we find an optimal policy?

Typically done iteratively:



1. Policy Evaluation:

Estimate quality of states (and actions) with current policy

2. Policy Improvement:

Improve policy by taking actions with the highest quality

This algorithm is called **Policy Iteration**

How do we find an optimal policy?

Policy Improvement:

Improve policy by taking actions with the highest quality

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \begin{cases} 1, & \text{if } \boldsymbol{a} = \operatorname{argmax}_{\boldsymbol{a}'} Q^\pi(\boldsymbol{s}, \boldsymbol{a}') \\ 0, & \text{otherwise} \end{cases}$$

This policy update is called greedy, as it greedily maximizes the Q-function (without exploration)

Iterating Policy Evaluation and Policy Improvement converges to the optimal policy and is called Policy Iteration

Policy Iteration Algorithm

Init: $V_0^\pi(s) \leftarrow 0, \pi \leftarrow \text{uniform}$

Repeat

Repeat $k = k + 1$

Compute Q-Function (for each state action pair)

$$Q_{k+1}^\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k^\pi(s')$$

Compute V-Function (for each state)

$$V_{k+1}^\pi(s) = \sum_a \pi(a|s) Q_{k+1}^\pi(s,a)$$

until convergence of V

Update Policy:

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \begin{cases} 1, & \text{if } \boldsymbol{a} = \text{argmax}_{\boldsymbol{a}'} Q^\pi(\boldsymbol{s}, \boldsymbol{a}') \\ 0, & \text{otherwise} \end{cases}$$

until convergence of policy

Value iteration

Can we also **stop policy evaluation before convergence** and perform a policy update?

⇨ Yes! We will still converge to the optimal policy !

„Extreme" case: Stop policy evaluation after 1 iteration

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

⇨ This equation is called the Bellman Equation $V^*(\boldsymbol{s})$

⇨ Iterating this equation computes the value function of the optimal policy

Value Iteration

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

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

More Identities:

➡Computing optimal V-Function from optimal Q-Function

$$V^*(\boldsymbol{s}) = \max_{\boldsymbol{a}} Q^*(\boldsymbol{s}, \boldsymbol{a})$$

… using the definition of the optimal policy

➡Computing optimal Q-Function from optimal V-Function

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

The Value Iteration Algorithm

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

Repeat

$$k = k + 1$$

Compute optimal Q-Function

$$Q^*_{k+1}(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*_k(s')$$

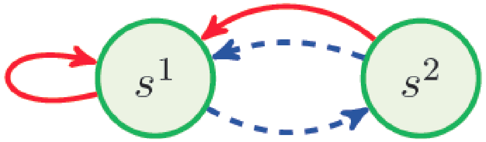
Compute optimal V-Function

$$V^*_{k+1}(s) = \max_a Q^*_{k+1}(s,a)$$

until convergence of V

This algorithm is also called Dynamic Programming

Example: Two State Problem



States: s^1, s^2

Actions: red (a^1) and blue (a^2) edges

Transition:

$$\mathcal{P}(s^1|s^1,a^1)=1,\ \mathcal{P}(s^2|s^1,a^1)=0,\ \mathcal{P}(s^1|s^1,a^2)=0,\ \mathcal{P}(s^2|s^1,a^2)=1$$

$$\mathcal{P}(s^1|s^2,a^1)=1,\ \mathcal{P}(s^2|s^2,a^1)=0,\ \mathcal{P}(s^1|s^2,a^2)=1,\ \mathcal{P}(s^2|s^2,a^2)=0$$

Rewards: $r(s^1)=1,\ r(s^2)=0$

Value iteration:

Compute V*(s) and Q*(s,a): Blackboard

Wrap Up

To compute an optimal policy, we can either use

Policy Iteration:

➡Fix current policy and evaluate its value function

➡Compute new policy greedily on the value function

Value Iteration:

➡Directly iterate optimal value function

➡Policy can be obtained afterwards from optimal Q-Function

Wrap-Up: Dynamic Programing

We now know how to compute optimal policies if we know the model

Wait, there is a catch!

Unfortunately, we can only do this in 2 cases

- Discrete Systems

  Easy: integrals turn into sums

  …but the world is not discrete!

- Linear Systems, Quadratic Reward, Gaussian Noise (LQR)

  … but the world is not linear!

Dynamic Programing

In all other cases, we have to use approximations!

Why?

1. Representation of the V-function:

   How to represent V in continuous state spaces?

2. We need to solve:

   $\max_{\boldsymbol{a}} Q^*(\boldsymbol{s}, \boldsymbol{a})$: difficult in continuous action spaces

   $\mathbb{E}_{\mathcal{P}}\left[V^*(\boldsymbol{s}') \middle| \boldsymbol{s}, \boldsymbol{a}\right]$: difficult for arbitrary functions V and models $\mathcal{P}$

Outline

Known Model: Markov Decision Processes

- Value Functions and Policy Evaluation
- Policy Iteration and Value Iteration

Unknown Model: Reinforcement Learning

- Temporal Difference Learning
- TD-Learning for Control: Q-Learning and Sarsa
- Value Function Approximation
- Least-Squares Temporal Difference Learning

Temporal Difference Learning

If the model is not known, we have to rely on trajectory data

$$\mathcal{D} = \{s_t, a_t, r_t, s_{t+1}\}_{t=1\dots N}$$ discrete state-actions for now

Temporal Difference Methods:

- Online methods, update after each sample
- Use the temporal difference error for the update

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

- Update of the Value Function:

$$\begin{aligned} V_{t+1}(s_t) &= V_t(s_t) + \alpha_t(r_t + \gamma V_t(s_{t+1}) - V_t(s_t)) \\ &= V_t(s_t) + \alpha_t \delta_t \end{aligned}$$

  - Learning Rate: α_t

Temporal difference learning

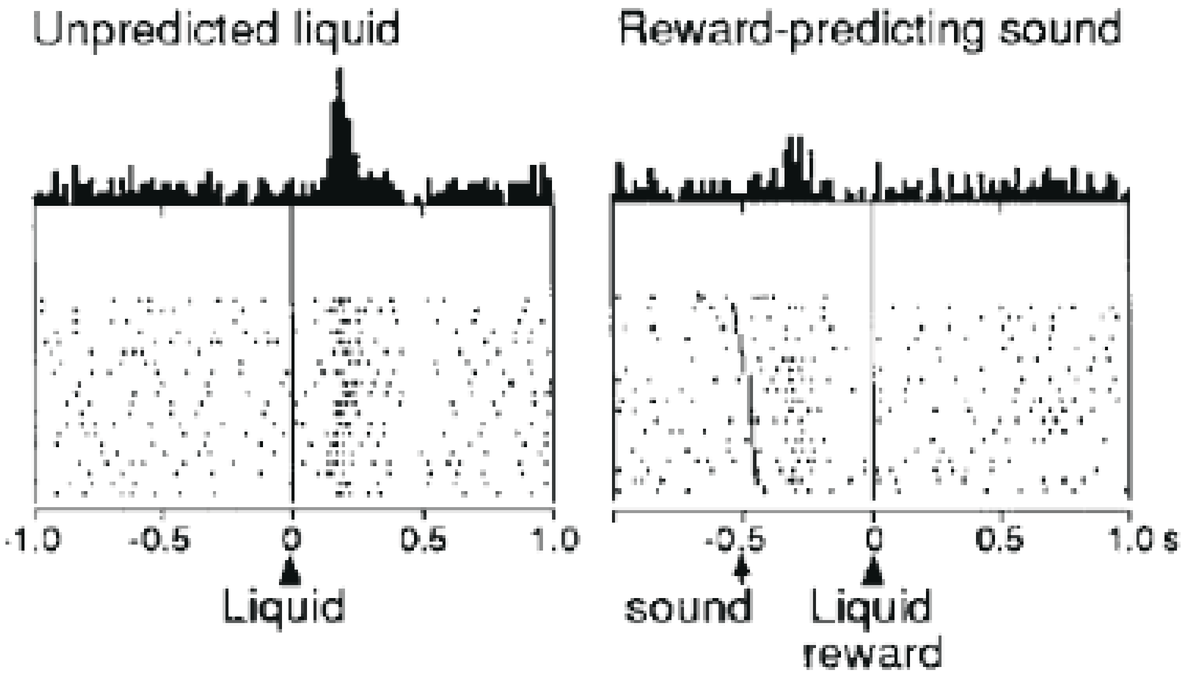
The TD error update

$$\begin{aligned} V_{t+1}(s_t) &= V_t(s_t) + \alpha_t(r_t + \gamma V_t(s_{t+1}) - V_t(s_t)) \\ &= (1-\alpha_t) \underbrace{V_t(s_t)}_{\text{current estimate}} + \alpha_t \underbrace{(r_t + \gamma V_t(s_{t+1}))}_{\text{1-step prediction } = \hat{V}_t(s_t)} \end{aligned}$$

interpolates between the one-time step lookahead prediction and the current estimate of the value function

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

Dopamine as TD-error?

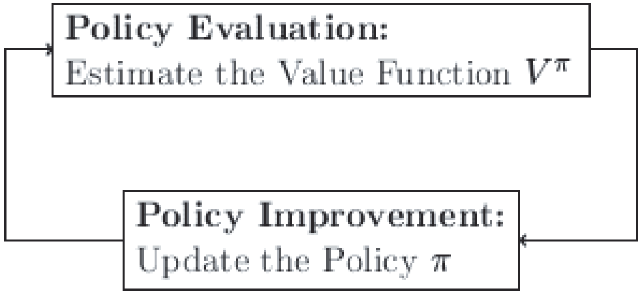


Monkey brains seem to have it...

- Unpredicted Reward (Liquid) ⇨ Positive TD Error ⇨ higher firing rate of neurons
- Predicted Reward (Sound Event) ⇨ No TD Error
- Unpredicted Predictor (Sound) ⇨ Positive TD Error ⇨ high firing rate of neurons

TD learning for control

So far: Policy evaluation with TD methods



Can we also do the policy improvement step with samples?

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \begin{cases} 1, \text{ if } \boldsymbol{a} = \text{argmax}_{\boldsymbol{a}'} Q^\pi(\boldsymbol{s}, \boldsymbol{a}') \\ 0, \text{ otherwise} \end{cases}$$

TD learning for control

Policy Improvement Step:

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \begin{cases} 1, & \text{if } \boldsymbol{a} = \operatorname{argmax}_{\boldsymbol{a}'} Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}') \\ 0, & \text{otherwise} \end{cases}$$

For control, we need to know the Q-function

- Just knowing the V-function is only sufficient if we would have the model!

$$\begin{aligned} Q_{t+1}(s_t, a_t) &= Q_t(s_t, a_t) + \alpha_t(r_t + \gamma Q_t(s_{t+1}, a_?) - Q_t(s_t, a_t)) \\ &= Q_t(s_t, a_t) + \alpha_t \delta_t \end{aligned}$$

Exploration

We do not know the real Q-values:

- By using the greedy policy, we might miss out better actions
- So we still **need to explore**!
- **Exploration-Exploitation Tradeoff**, one of the hardest problems in RL

Common Exploration Policies (discrete actions)

- **Epsilon-Greedy Policy:**

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

Take random action with probability ϵ

- **Soft-Max Policy:**

Higher Q-Value, higher probability.

$$\pi(a|s) = \frac{\exp\left(Q(s,a)/\beta\right)}{\sum_{a'} \exp\left(Q(s,a')/\beta\right)}$$

$\beta \ldots$ temperature

SARSA and Q-Learning

Update equations for learning the Q-function

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \left(r_t + \gamma Q_t(s_{t+1}, a_?) - Q_t(s_t, a_t)\right)$$

Two different methods to estimate

- **Q-learning:** $a_? = \arg\max'_a Q(s_{t+1}, a')$

  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

Both methods perform quite similarly

Outline

Known Model: Markov Decision Processes

- Value Functions and Policy Evaluation
- Policy Iteration and Value Iteration

Unknown Model: Reinforcement Learning

- Temporal Difference Learning
- TD-Learning for Control: Q-Learning and Sarsa
- Value Function Approximation
- Least-Squares Temporal Difference Learning

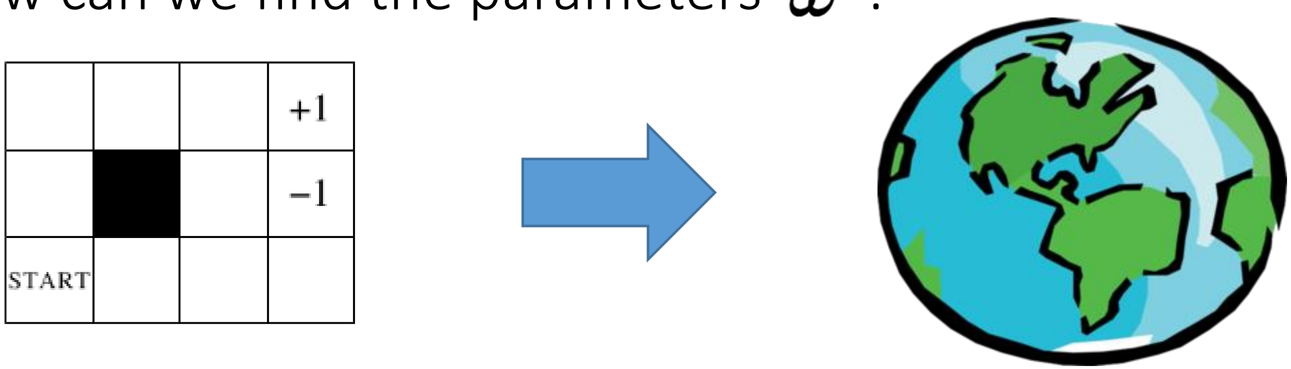
Value Function Approximation

Yet, the world is not discrete…

- ⇨ In most cases value function can not be represented exactly
- ⇨ We need to approximate it !
- ⇨ 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 $\boldsymbol{\omega}$?



Approximating the Value Function

Ok, lets use again the Dynamic Programming idea of the **recursive V-function estimation**:

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

Bootstrapping (BS): Use the old approximation $V_{\boldsymbol{\omega}_{\mathrm{old}}}$ to get the target values for a new approximation

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

How can we **minimize** this function ?

⇨ Lets use **stochastic gradient descent**

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:**

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

Learning rate α_k

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

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

TD-Learning with Function Approximation

Stochastic gradient descent on our error function *MSE_{BS}*

$$\mathrm{MSE}_{\mathrm{BS}} = 1/N \sum_t \Big(\underbrace{r_t + V_{\boldsymbol{\omega}_{\mathrm{old}}}(\boldsymbol{s}_{t+1})}_{\text{1-step prediction w. old model}} - \underbrace{V_{\boldsymbol{\omega}}(\boldsymbol{s}_t)}_{\text{currente estimate}} \Big)^2$$

Update rule (for current time step t)

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

where $\delta_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_{t+1}) - V_{\boldsymbol{\omega}_t}(\boldsymbol{s}_t)$ is again the temporal difference error

TD-Learning with Function Approximation

We just derived the TD-Learning Update rule with linear FA

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

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

The discrete case is a special case with tabular features

$$\phi(s^i) = \boldsymbol{e}_i \ldots \ i\text{th unit vector}$$

The $\boldsymbol{\omega}$ vector directly contains the values for each state

$$V_{\boldsymbol{\omega}}(s^i) = \boldsymbol{\phi}(s^i)^T\boldsymbol{\omega} = \omega_i$$

Plugging in the tabular features:

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha(r_t + \gamma V_t(\boldsymbol{s}_{t+1}) - V_t(\boldsymbol{s}_t))\boldsymbol{e}_i$$

Only the ith element will change

$$V_{t+1}(s_t^i) = \boldsymbol{\omega}_{t+1,i} = \boldsymbol{\omega}_{t,i} + \alpha(r_t + \gamma V_t(\boldsymbol{s}_{t+1}) - V_t(\boldsymbol{s}_t))$$

Q-Learning with FA

➡ **Q-learning:**

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_t \left(r_t + \gamma \max_{\boldsymbol{a}'} Q_t(\boldsymbol{s}_{t+1}, \boldsymbol{a}') - Q_t(\boldsymbol{s}_t, \boldsymbol{a}_t) \right) \boldsymbol{\phi}(\boldsymbol{s}_t)^T$$

➡ **SARSA:**

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha_t \left(r_t + \gamma Q_t(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) - Q_t(\boldsymbol{s}_t, \boldsymbol{a}_t) \right) \boldsymbol{\phi}(\boldsymbol{s}_t)^T$$

Some results with Q-Learning

Backgammon: World-Champion Level

Atari-Games: Human Level Performance

- No prior knowledge
- Pixels as input of the algorithm

Outline

Known Model: Markov Decision Processes

- Value Functions and Policy Evaluation
- Policy Iteration and Value Iteration

Unknown Model: Reinforcement Learning

- Temporal Difference Learning
- TD-Learning for Control: Q-Learning and Sarsa
- Value Function Approximation
- Least-Squares Temporal Difference Learning

Batch-Mode Reinforcement Learning

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

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

only once

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

- **Least-Squares Temporal Difference (LSTD) Learning**
- **Fitted Q-Iteration (not in this lecture...)**

Computationally much more expensive then TD-learning!

Least-Squares Temporal Difference (LSTD)

Minimize the bootstrapped *MSE* objective (*MSE_{BS}*)

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

Least-Squares Solution: The solution for omega can be obtained in closed form

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

State-Feature matrix: $\boldsymbol{\Phi} = [\boldsymbol{\phi}(\boldsymbol{s}_1), \boldsymbol{\phi}(\boldsymbol{s}_2), \ldots, \boldsymbol{\phi}(\boldsymbol{s}_N)]^T$

Next-State Feature matrix: $\boldsymbol{\Phi}' = [\boldsymbol{\phi}(\boldsymbol{s}_1'), \boldsymbol{\phi}(\boldsymbol{s}_2'), \ldots, \boldsymbol{\phi}(\boldsymbol{s}_N')]^T$

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 find a fixed point, i.e. $\boldsymbol{\omega}_{\text{old}} = \boldsymbol{\omega}$

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

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

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

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

Least-Squares Temporal Difference (LSTD)

LSQ: Adaptation for learning the Q-function

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

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

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

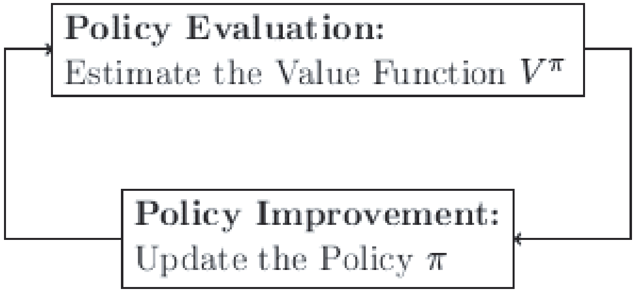
Update Policy with…

➡ **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:**

$$\pi(a|s) = \frac{\exp\left(Q(s,a)/\beta\right)}{\sum_{a'} \exp\left(Q(s,a')/\beta\right)}$$



Learning to Ride a Bicycle

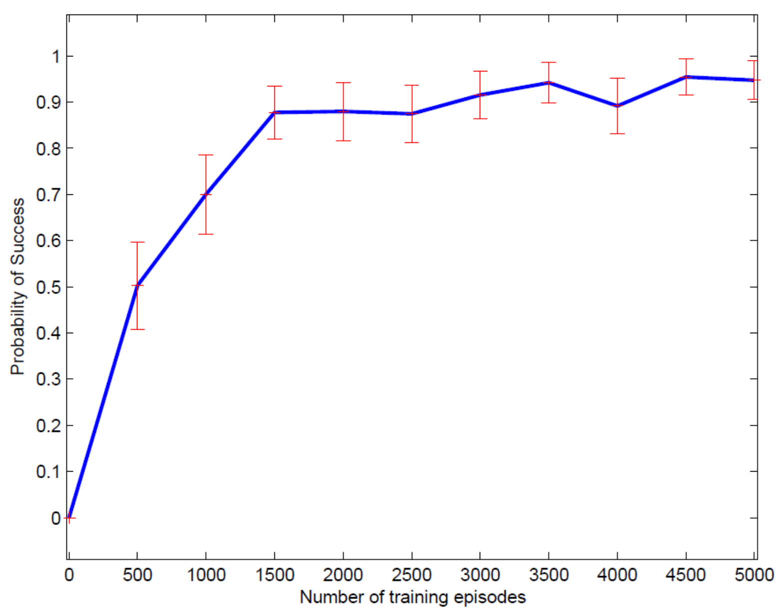
State space: $s = [\theta, \dot{\theta}, \omega, \dot{\omega}, \ddot{\omega}, \psi]$

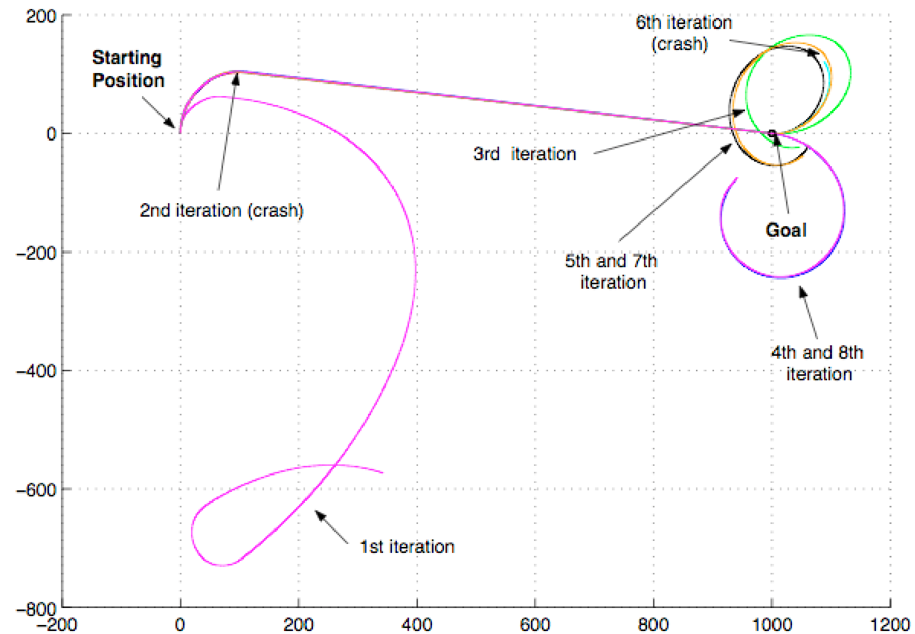
θ angle of handlebar, ω vertical angle of bike, ψ angle to goal

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

Feature space: 20 basis functions…

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





Summary &Outlook

In order to do optimal decisions, we can learn the value function

- ➡ Temporal Difference Learning
- ➡ Value Function Approximation
- ➡ Batch Reinforcement Learning

Other approaches for reinforcement learning:

- ➡ Policy Search:
  - ➡ Directly optimize parameters of the policy
  - ➡ Avoids V-Function approximation errors
  - ➡ Good for continuous actions
- ➡ Model-Based Methods:
  - ➡ Learn Model of the MDP
  - ➡ Use planning to obtain optimal policy