Reinforcement Learning for Flight Control

AE4-311 Advanced Flight Control

Erik-Jan van Kampen, E.vanKampen@TUDelft.NL
Delft University of Technology

June 6, 2016

Outline

1. Introduction
2. Discrete RL
3. Monte Carlo vs TD
4. Continuous RL
5. RL applied to F-16
6. Current work
7. Assignment

Learning objectives

After this lecture on Reinforcement Learning, the student can:

- Explain the basic concepts of Reinforcement Learning.
- Describe the different discrete RL methods.
- Describe the differences between discrete and continuous RL and the consequences of using function approximators.
- Construct his/her own RL controller for a simple system.

What is Reinforcement Learning?

- Based on human learning
- Interaction with the environment
- Different from supervised learning

Concepts & definitions

- Agent
- State
- Environment
- Action
- Reward
- Value function
- Policy

Types of learning

- Low level
  - Perception and recognition;
  - Repetion/copying
  - Ordering/arranging
- High level
  - Analysis
  - Synthesis
  - Adaptation
  - Application of knowledge

Reinforcement Learning algorithms exhibit high level learning by adapting to changing circumstances and by finding non-trivial solutions, for example when applied to nonminimum phase systems.

Why use Reinforcement Learning?

- Adaptive control: Learn how to control an aircraft after either the aircraft itself or its environment has changed beyond the scope of the original controller.
- Black-box appoach: Learn from scratch without knowing anything about the internal dynamics of the system.

Figure: Airbus A300 struck by missile over Baghdad on Nov 22, 2003

RL system layout

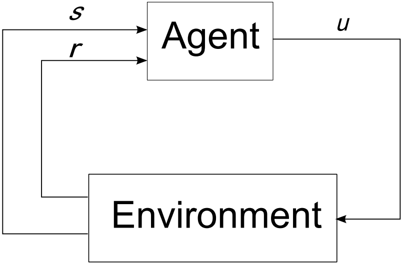


Figure: Reinforcement Learning system

- s: State
- u: Action
- r: Reward

Policy

How does the agent decide what action to perform next?

Policy

A policy $\pi_t(s,u)$ is a mapping from the state s to the action u:

$$\pi_t(s,u) = P(u_t = u \,|\, s_t = s) \tag{1}$$

which is sometimes written as:

$$u = \pi(s) \tag{2}$$

Some types of policies are:

- Random
- Greedy
- ϵ-greedy

Reward

The reward signal represents the feedback that the agent receives from the environment. It can be:

- Discrete or Continuous
- Instantanious or Delayed
- Low level (supervised learning) or High level (autonomous)

The reward function has to be chosen by the designer of the RL-controller and will greatly influence the performance of the controller.

Value function

How can we store and use the information obtained from the environment?

Value function

The value function $V^{\pi}(s)$is the expected sum of future rewards R_t, when starting from state s and following policy π:

$$V^{\pi}(s) = E_{\pi}\left\{R_t \,|s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s\right\} \tag{3}$$

with
γ: Discount rate

Bellman optimality equation

The value function has a recursive property:

Recursive Value function

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \,|s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^{2} r_{t+3} + \ldots \,|s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \,|s_t = s\right\}
\end{aligned}
\tag{4}
$$

The Bellman optimality equation is used to evaluate the performance of value function approximations. These value fuction approximations are required for continuous RL.

Value function

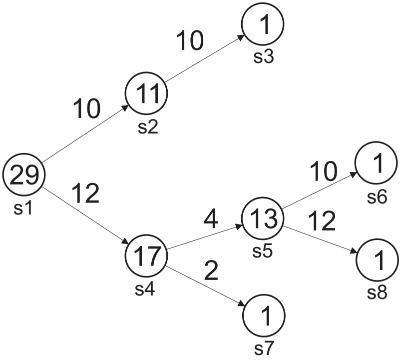
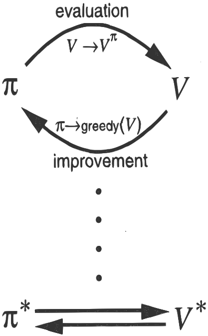
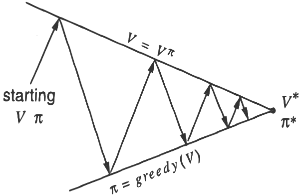


Figure: Recursive property of the value function - Greedy policy, $\gamma = 1$

Value iteration

By iteration between updating of the value function and updating of the policy, the agent will converge to the optimal value function and the corresponding optimal policy.





Source: *Reinforcement Learning*, Sutton & Barto, 1998

Figure: Value-Policy update iteration

Q-values

The Q-value, or action-value, is an extension of value function with the next action:

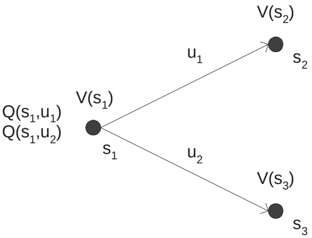


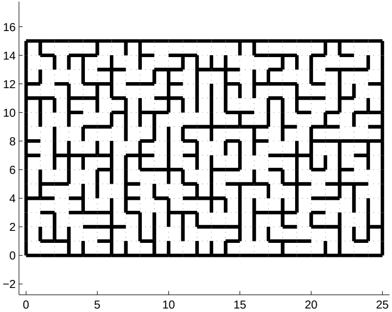
Figure: Q-Value versus Value Function

$$Q^{\pi}(s,u) = E_{\pi}\left\{R_t \mid s_t = s, u_t = u\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, u_t = u\right\} \tag{5}$$

Discrete RL

Maze example: Find optimal value function and policy

- Discrete states (cells) and actions (up,down,left,right).
- Reward function: -1 when running into wall, $+3$ when reaching target state, otherwise 0.
- Policy during learning: random.



Maze example

Algorithm:

- Make grid with N_{rows} times $N_{columns}$ discrete states.
- Set value function to random value for each state.
- Determine initial policy: random.
- While $V_{new} - V_{old} > \epsilon$
  - For each state, take action according to policy (Use matrix computations!)
  - Evaluate reward and update value function estimate.
- (Optional) Repeat with policy adapted to new value function (greedy).

Maze example: Value function

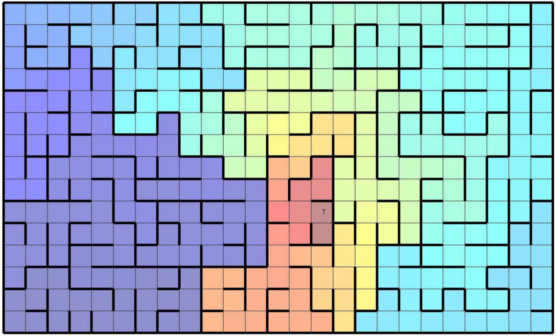


Figure: Value function for the maze example

$$V^{\pi}(s) = E_{\pi}\left\{R_t \,|s_t = t\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s\right\} \qquad (6)$$

Maze example: Policy

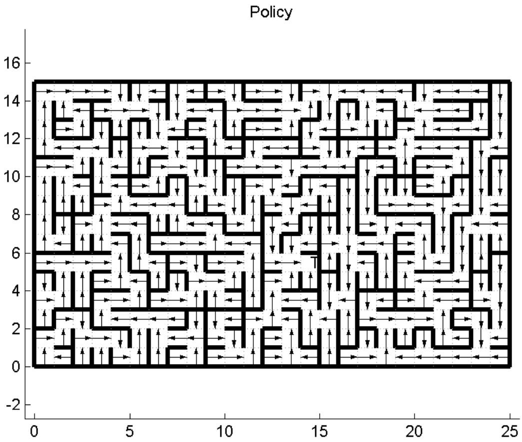


Figure: Greedy policy for the maze example.

Discrete RL

Value function updating

- Dynamic Programming (DP)
- Monte Carlo (MC)
- Temporal Differences (TD(0))
- Eligibility traces TD(λ)

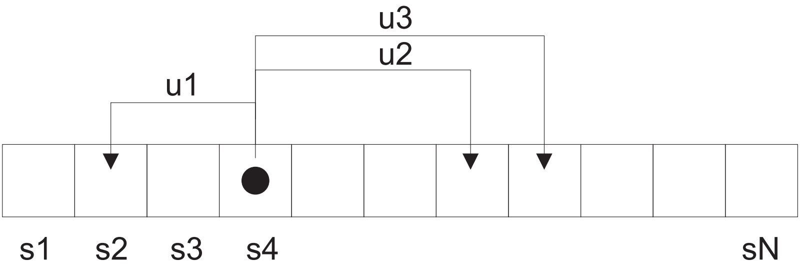


Figure: Example problem with discrete states

Dynamic Programming

Dynamic Programming (DP)

- Requires complete knowledge about state transition probabilities $\mathcal{P}_{ss'}^{a}$.
- Requires complete knowledge about rewards for each state transition $\mathcal{R}_{ss'}^{a}$.

$$\mathcal{P}_{ss'}^{a} = P\left(s_{t+1} = s' | s_t = s, a_t = a\right)$$

$$\mathcal{R}_{ss'}^{a} = E\left[r_{t+1} | a_t = a, s_t = s, s_{t+1} = s'\right]$$

Dynamic Programming

Remember the definition of the value function:

$$V^{\pi}(s) = E_{\pi}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \ldots | s_t = s\right\}$$

which can be expressed recursively as:

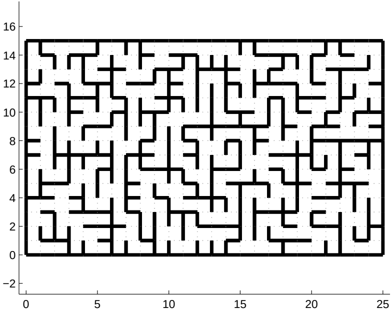
$$V^{\pi}(s) = E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\right\}$$

The DP equivalent is:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}_{ss'}^{a}\left[\mathcal{R}_{ss'}^{a} + \gamma V^{\pi}(s')\right]$$

Dynamic Programming

Maze example:

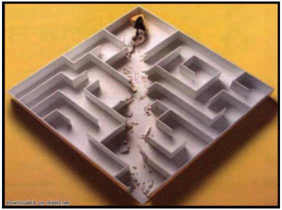


$\mathcal{P}^a_{ss'}$? $\mathcal{R}^a_{ss'}$?

Monte Carlo methods

Sometimes $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ are not known or difficult to compute. In those cases the required information to estimate $V(s)$ can be obtained experimentally in episodes.

In the maze example we let an agent walk through the maze to collect rewards. After the agent has finished the episode, the gathered rewards are used to estimate $V(s)$. This is an example of a Monte Carlo method.



First-visit Monte Carlo

First-visit Monte Carlo

Initialize:

$\pi \leftarrow$ policy to be evaluated.

$V \leftarrow$ an arbitrary state-value function.

$Returns(s) \leftarrow$ an empty list, for all $s \in S$.

Repeat forever:

(a)Generate an episode using π.

(b)For each state s in the episode:

$R \leftarrow$return following first visit to s.

Append R to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

Matlab Demo!. What about updating the policy?

Monte Carlo Exploratory Starts

Monte Carlo ES

Initialize, for all $s \in S, a \in A$:

$Q(s, a) \leftarrow$ arbitrary.

$\pi(s) \leftarrow$ arbitrary.

$Returns(s, a) \leftarrow$ an empty list.

Repeat forever:

(a)Generate an episode using exploratory starts and $\pi(s)$.

(b)For each pair s, a in the episode:

$R \leftarrow$return following first occurrence of s, a.

Append R to $Returns(s, a)$

$Q(s, a) \leftarrow$ average($Returns(s, a)$)

(c)For each s in the episode:

$\pi(s) \leftarrow arg \quad max_a Q(s, a)$

Temporal Difference updating

With Monte Carlo methods we have to wait until the episode is finished before we start updating the value and policy.

TD updating

With TD methods the value function is updated *during* the episode.

$$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right]$$

TD methods learn estimates based on other estimates: **Bootstrapping**.

Temporal Difference updating

TD updating

Initialize:

$\pi \leftarrow$ policy to be evaluated.

$V \leftarrow$ an arbitrary state-value function.

Repeat for each episode:

Initialize s.

Repeat for each step of episode:

$a \leftarrow$ action given by π for s.

Take action a; observe r and next state s'.

$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right]$.

$s \leftarrow s'$.

until s is terminal.

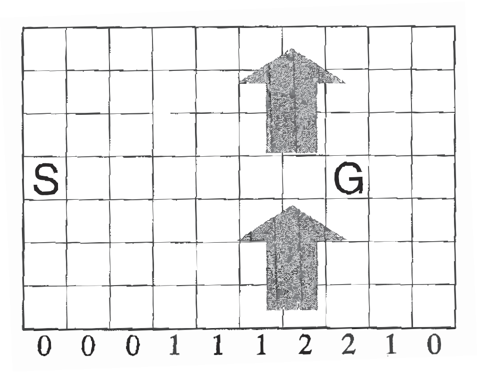
Sarsa: on-policy TD control algorithm

SARSA: **S**tate, **A**ction, **R**eward, next**S**tate, next**A**ction

Sarsa

Initialize $Q(s,a)$arbitrarily
Repeat (for each episode):
    Initialize s.
    Choose a from s using policy derived from Q.
    Repeat (for each step of the episode):
        Take action a; observe r and next state s'.
        Choose a' from s' using policy derived from Q.
        $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$.
        $s \leftarrow s'$.
        $a \leftarrow a'$.
    until s is terminal.

Sarsa example: Windy gridworld



Sarsa example: Windy gridworld

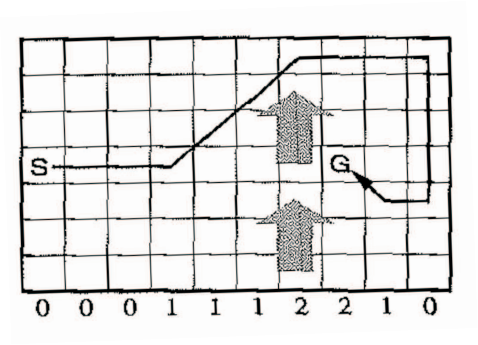
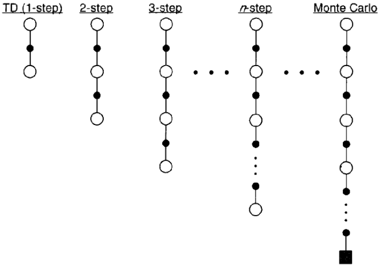


Figure: Windy gridworld optimal policy

TD(λ) Eligibility traces

TD(λ) uses a variable number of visited states to update the value function. In one extreme there is TD(0) which uses only a single state transition, while the other extreme is MC, which uses a whole episode of state transition data.



Continuous RL

For many real world applications the states and actions are continuous and multi-dimensional. For these applications the value function and policy cannot be stored for each state.

This also holds for discrete RL with many dimensions, i.e. assume a system with 10 states, each subdivided in only 20 discrete possibilities. When the value function for one state can be encoded in 1 byte, then the complete discrete value function requires over 10 Terabytes of memory/storage capacity.

So we need to approximate the value function and the policy.

Adaptive Critic Designs (ACD)

- Division of tasks and components: Actor and Critic
- J is an approximation of the value function V
- The Critic is a mapping from state to value function
- The Actor is a mapping from state to action

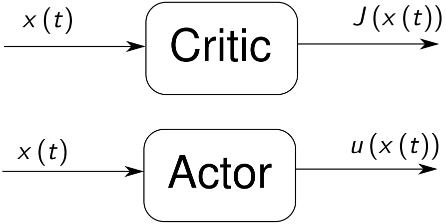


Figure: Actor/Critic

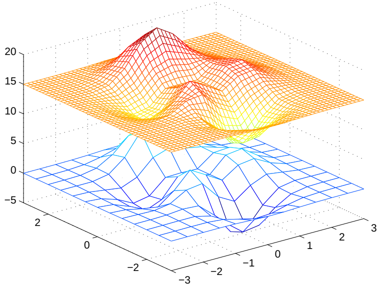
Function Approximators

Function approximators

- Lookup tables (interpolated)
- Splines/polynomials
- Artificial neural networks
- Fuzzy system

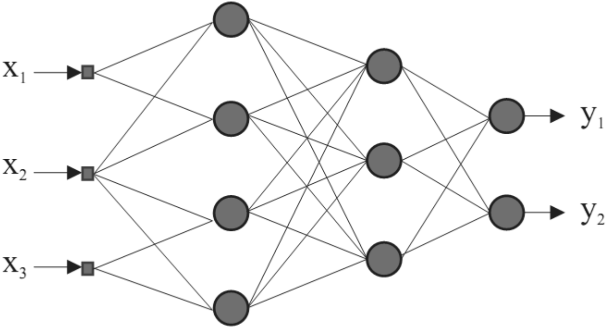
Function Approximators: Interpolation

```
[X,Y] = meshgrid(-3:.5:3);
Z = peaks(X,Y);
[XI,YI] = meshgrid(-3:.125:3);
ZI = interp2(X,Y,Z,XI,YI);
mesh(X,Y,Z), hold, mesh(XI,YI,ZI+15), hold off
axis([-3 3 -3 3 -5 20])
```



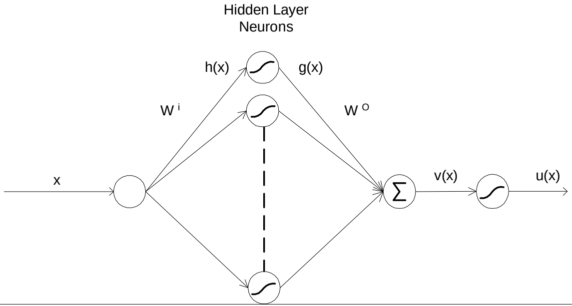
Function Approximators: Artificial Neural Networks

- Simulates biological network of neurons

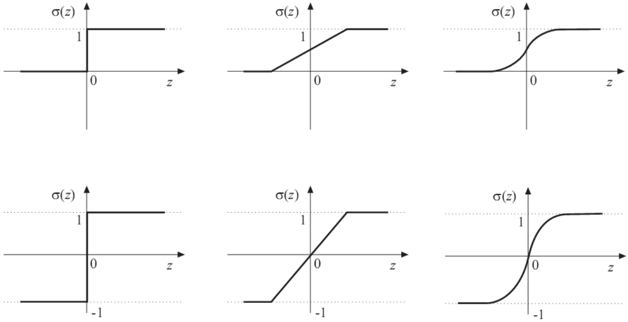


Function Approximators: Artificial Neural Networks

- Nonlinear mapping through activation function
- Radial basis function
- Feedforward network
- Learning/training

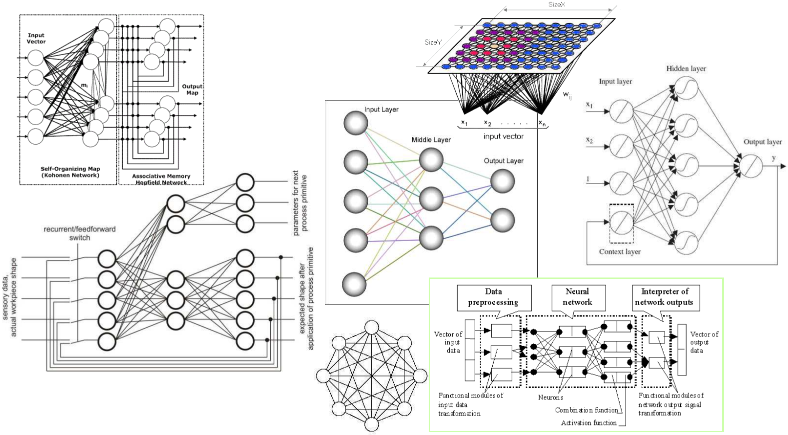


Artificial Neural Network activation functions



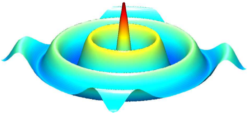
Other types of ANN

- Many different network structures exist today ranging from simple to extremely complex networks.

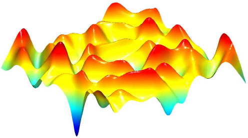


Visual perspective on NN's

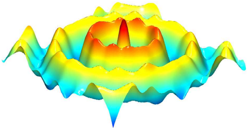
Demonstration of approximation power:



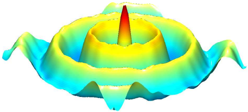
(a) Real IO mapping



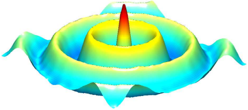
(b) NN 25 neurons



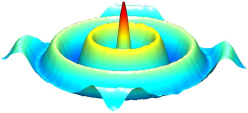
(c) NN 50 neurons



(d) NN 75 neurons



(e) NN 100 neurons



(f) NN 200 neurons

Function Approximators: Splines

- Splines are piecewise polynomials.
- Very high approximation power.
- Spline continuity order can be chosen freely.
- Spline parameters have a spatial location allowing local model modification.
- Naturally capable of fitting scattered datasets

Function Approximators: Splines

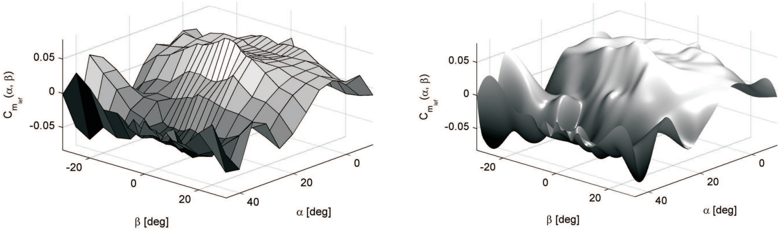


Figure: F-16 Aerodynamic model data represented by data tables (left) and splines (right)

Movies multivariate simplex spline:

- B-coefficients –> `./Movies/bcoefs.wmv`
- C_m model –> `./Movies/CmModel.wmv`

Learning in ACD's

Looking back at the Belmann equation for the recursive value function:

Recursive Value function

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,|s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \,|s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \,|s_t = s\right\}
\end{aligned} \tag{7}
$$

We can make a continuous version of this:

$$J(t) = r(t+1) + \gamma J(t+1) \tag{8}$$

And construct the Temporal Difference (TD) error as:

$$TD = r(t+1) + \gamma J(t+1) - J(t) \tag{9}$$

Learning in ACD's

The errors made by the actor and the critic are used to improve the function approximation:

Critic

The task of the critic is to approximate the value function:

$$\begin{aligned} e_c(t) &= J(t-1) - [\gamma J(t) - r(t)] \\ E_c(t) &= \tfrac{1}{2} e_c^2(t) \end{aligned} \tag{10}$$

Actor

The task of the actor is to reach a goal value J^*:

$$\begin{aligned} e_a(t) &= J(t) - J^*(t) \\ E_a(t) &= \tfrac{1}{2} e_a^2(t) \\ \\ \text{if}\, r(x) &\leq 0 \quad \forall x \quad \text{then} \quad J^* = 0 \end{aligned} \tag{11}$$

ADHDP - Action Dependent Heuristic Dynamic Programming

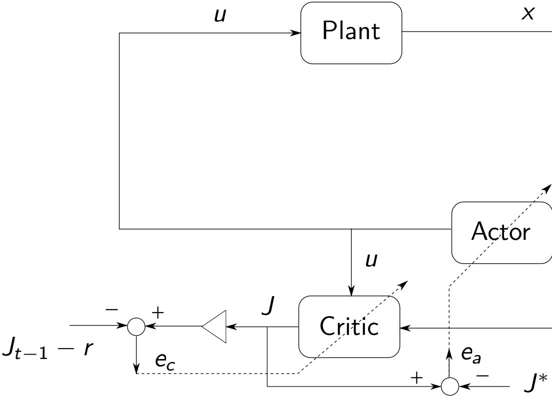


Figure: ADHDP

ADHDP

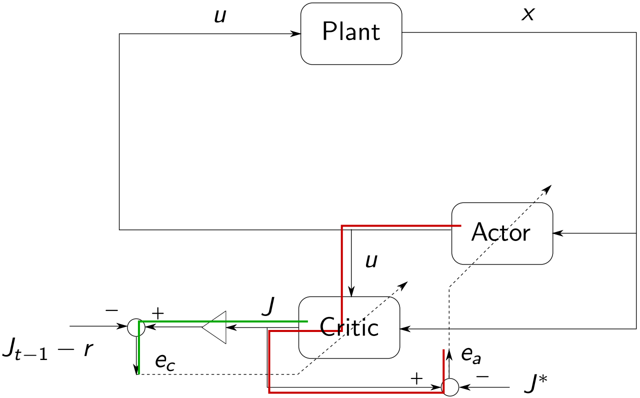


Figure: ADHDP error backpropagation

ADHDP

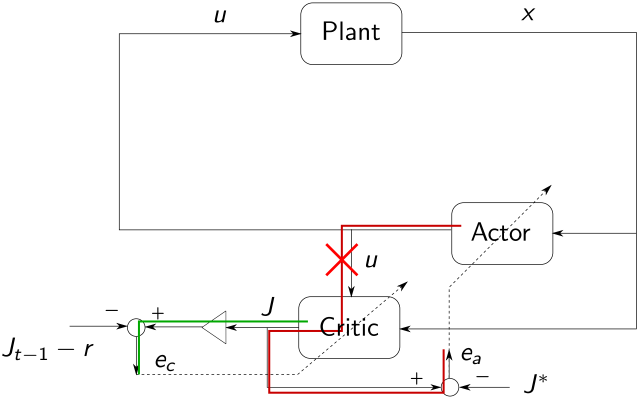


Figure: ADHDP error backpropagation

ADHDP

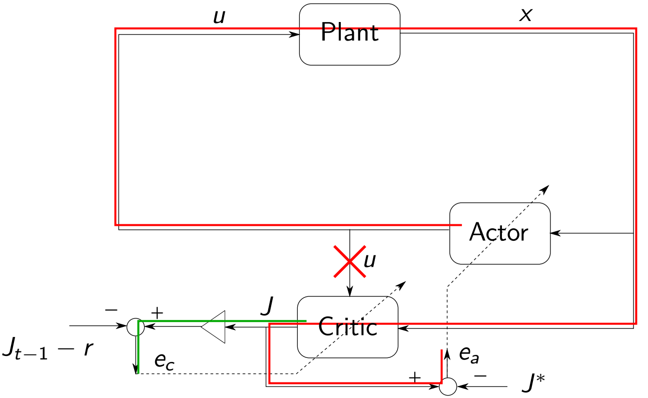


Figure: ADHDP error backpropagation

ADHDP

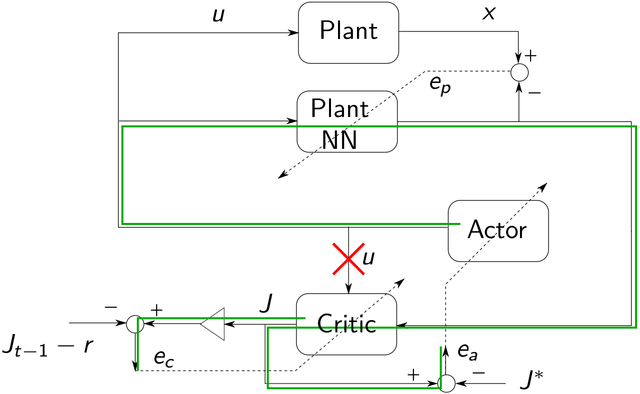


Figure: ADHDP error backpropagation

Error Backpropagation

Actor

$$E_a(t) = \frac{1}{2} e_a^2(t)$$

$$e_a(t) = J(t) - J^*(t)$$

$$w_{a_k}(t+1) = w_{a_k}(t) + \Delta w_{a_k}(t)$$

$$\Delta w_{a_k}(t) = \beta_a(t)\left[-\frac{\partial E_a(t)}{\partial w_{a_k}(t)}\right]$$

$$\frac{\partial E_a(t)}{\partial w_{a_k}(t)} = \frac{\partial E_a(t)}{\partial e_a(t)} \frac{\partial e_a(t)}{\partial J(t)} \frac{\partial J(t)}{\partial u(t)} \frac{\partial u(t)}{\partial w_{a_k}(t)}$$

$$\frac{\partial E_a(t+1)}{\partial w_{a_k}(t)} = \frac{\partial E_a(t+1)}{\partial e_a(t+1)} \frac{\partial e_a(t+1)}{\partial J(t+1)} \frac{\partial J(t+1)}{\partial x(t+1)} \frac{\partial x(t+1)}{\partial u(t)} \frac{\partial u(t)}{\partial w_{a_k}(t)}$$

Error Backpropagation

Critic

$$E_c(t) = \frac{1}{2} e_c^2(t)$$

$$e_c(t) = J(t+1) - [\gamma J(t) - r(t)]$$

$$w_c(t+1) = w_c(t) + \Delta w_c(t)$$

$$\Delta w_c(t) = \beta_c(t)\left[-\frac{\partial E_c(t)}{\partial w_c(t)}\right]$$

$$\frac{\partial E_c(t)}{\partial w_c(t)} = \frac{\partial E_c(t)}{\partial e_c(t)} \frac{\partial e_c(t)}{\partial J(t)} \frac{\partial J(t)}{\partial w_c(t)}$$

Cart and pole system

- Cart and pole system
- Discrete or continuous
- Position control

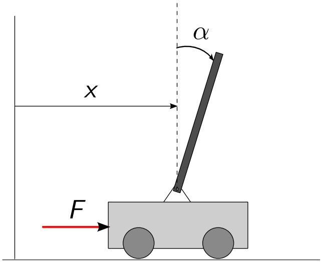


Figure: Cart and pole system

System dynamics

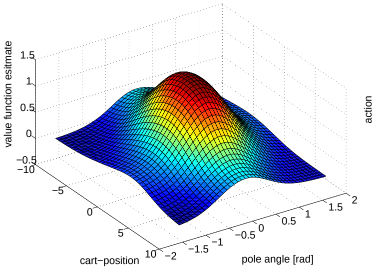
$$\ddot{\alpha} = \frac{g\sin(\alpha) - \cos(\alpha)\frac{F + m_p L_p \dot{\alpha}^2 \sin(\alpha)}{m_p + m_c}}{L_p\left(\frac{4}{3} - \frac{m_p}{m_p + m_c}\cos^2(\alpha)\right)}$$

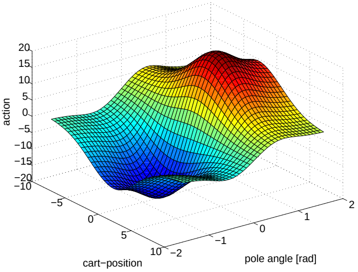
$$\ddot{x} = \frac{F + m_p L_p\left(\dot{\alpha}^2 \sin(\alpha) - \ddot{\alpha}\cos(\alpha)\right)}{m_p + m_c} \tag{12}$$

With:

- g: Gravitational acceleration.
- m_p: Mass of the pole.
- m_c: Mass of the cart.
- L_p: Length of the pole.

Example





Movies!

- Training –> `./Movies/training.avi`
- Step –> `./Movies/step.avi`

Results cart and pole system: Discrete action

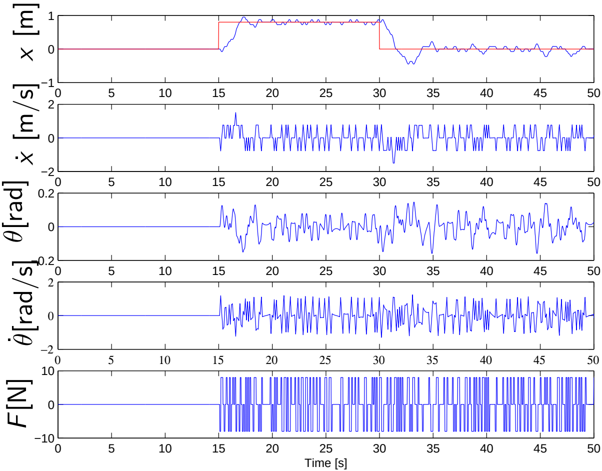


Figure: Step simulation for a discrete RL controller

Results cart and pole system: Continuous action

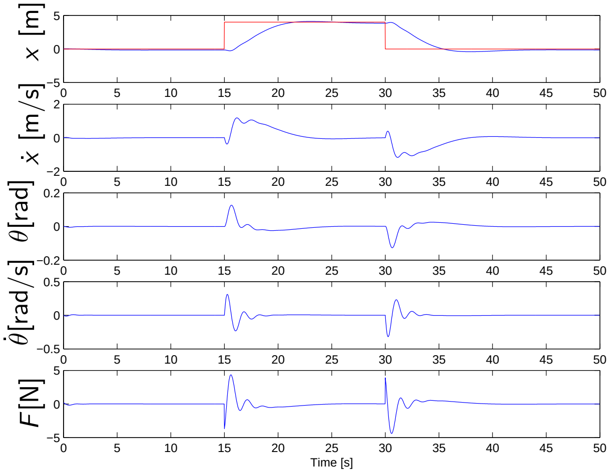


Figure: Step simulation for a continuous RL controller

RL applied to the F-16 fighter aircraft

Goal:

Design a RL controller that can:

1. Learn and control the baseline dynamics of an F-16 model.
2. Adapt itself when changes occur in either the environment or the dynamics.

The F-16 model

Assumptions

- The aircraft is modeled as a rigid body in space.
- The aircraft is assumed to be symmetric in the x_b-z_b.
- The rotation of the earth in space as well as the curvature of the earth are neglected.

The equations of motion are:

$$\dot{u} = rv - qw - g\sin\theta + \frac{q_d S}{m} C_X + \frac{T}{m}$$

$$\dot{v} = pw - ru + g\cos\theta\sin\phi + \frac{q_d S}{m} C_Y$$

$$\dot{w} = qu - pv + g\cos\theta\cos\phi + \frac{q_d S}{m} C_Z$$

$$\dot{p} I_X - \dot{r} I_{XZ} = pq I_{XZ} - qr\left(I_Z - I_Y\right) + q_d S b C_l$$

$$\dot{q} I_Y = pr\left(I_Z - I_X\right) - \left(p^2 - r^2\right) I_{XZ} + q_d S c C_m - r H_{eng}$$

$$\dot{r} I_Z - \dot{p} I_{XZ} = pq\left(I_X - I_Y\right) - qr I_{XZ} + q_d S b C_n + q H_{eng}$$

F-16 Aerodynamic model

The aerodynamic coefficients in the equations of motion, are written as a sum of contributing force and moment coefficients, some related to the control surfaces, others to the basic aerodynamic properties of the aircraft itself:

$$C_X = C_{X_0}(\alpha, \delta_e) + \left(\frac{qc}{2V_T}\right) C_{X_q}$$

$$C_Y = -3.50e^{-4}\beta + 1.83e^{-5}\delta_a + 5.0e^{-5}\delta_r + \left(\frac{b}{2V_T}\right)\left(C_{Y_p}(\alpha)\, p + C_{Y_r}(\alpha)\, r\right)$$

$$C_Z = C_{Z_0}(\alpha)\left(1 - \beta^2\right) - 1.33e^{-4}\delta_e + \left(\frac{qc}{2V_T}\right) C_{Z_q}(\alpha)$$

$$C_l = C_{l_0}(\alpha, \beta) + \Delta C_{l,\delta_a}\delta_a + \Delta C_{l,\delta_r}\delta_r + \left(\frac{b}{2V_T}\right)\left(C_{l_p}(\alpha)\, p + C_{l_r}(\alpha)\, r\right)$$

$$C_m = C_{m_0}(\alpha, \delta_e) + \left(\frac{qc}{2V_T}\right) C_{m_q}(\alpha) + \left(x_{c.g._{ref}} - x_{c.g.}\right) C_Z$$

$$C_n = C_{n_0}(\alpha, \beta) + \Delta C_{n,\delta_a}\delta_a + \Delta C_{n,\delta_r}\delta_r + \left(\frac{b}{2V_T}\right) \ldots$$

$$\left(C_{n_p}(\alpha)\, p + C_{n_r}(\alpha)\, r\right) - \left(\frac{c}{b}\right)\left(x_{c.g._{ref}} - x_{c.g.}\right) C_Y$$

F-16 Engine model

$$P_c(\delta_t) = \begin{cases} 64.96\delta_t & \text{if } \delta_t \leq 0.77 \\ 217.38\delta_t - 117.38 & \text{if } \delta_t > 0.77 \end{cases}$$

$$\dot{P}_a = \frac{1}{\tau_{eng}\left(P_c - P_a\right)}$$

$$P_c = \begin{cases} P_c & \text{if } P_c \geq 50 \text{ and } P_a \geq 50 \\ 60 & \text{if } P_c \geq 50 \text{ and } P_a < 50 \\ 40 & \text{if } P_c < 50 \text{ and } P_a \geq 50 \\ P_c & \text{if } P_c < 50 \text{ and } P_a < 50 \end{cases}$$

$$\frac{1}{\tau_{eng}} = \begin{cases} 5.0 & \text{if } P_c \geq 50 \text{ and } P_a \geq 50 \\ f(P_c - P_a) & \text{if } P_c \geq 50 \text{ and } P_a < 50 \\ 5.0 & \text{if } P_c < 50 \text{ and } P_a \geq 50 \\ f(P_c - P_a) & \text{if } P_c < 50 \text{ and } P_a < 50 \end{cases}$$

F-16 Engine model

$$f(P_c - P_a) = \begin{cases} 1.0 & \text{if } (P_c - P_a) \leq 25 \\ 0.1 & \text{if } (P_c - P_a) \geq 50 \\ 1.9 - 0.036\,(P_c - P_a) & \text{if } 25.0 < (P_c - P_a) < 50.0 \end{cases}$$

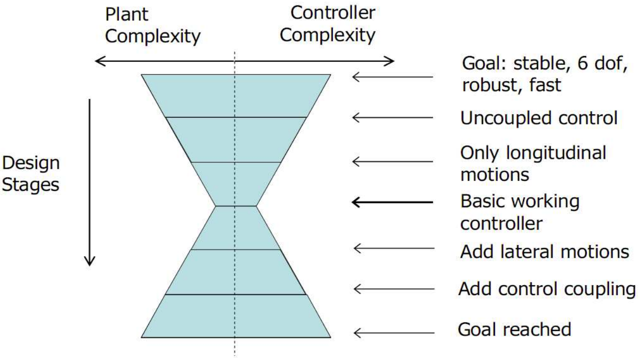
$$T = \begin{cases} T_{idle} + (T_{mil} - T_{idle})\left(\frac{P_a}{50}\right) & \text{if } P_a < 50 \\ T_{mil} + (T_{max} - T_{mil})\left(\frac{P_a - 50}{50}\right) & \text{if } P_a \geq 50 \end{cases}$$

Summary F-16 model

Nonlinear and coupled 6 DOF equations of motion, with underlying table based aerodynamic and engine data.

Control overview

The step from a discrete maze example to a continuous full F-16 model is too large, so we will start with some simplifications.



Control overview

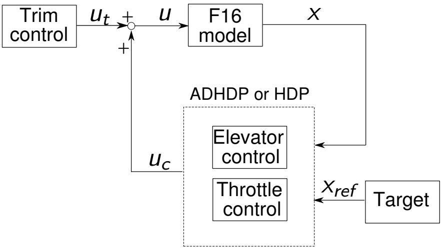


Figure: Setup of RL control System for the F16

Control setup

- Longitudinal
- 2-channels: elevator/throttle
- Starting from trimmed state

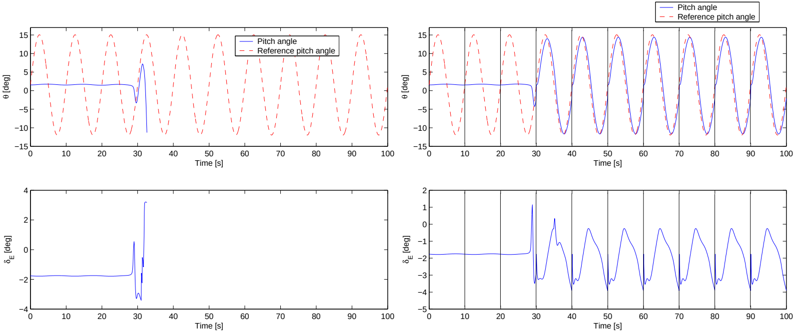
Offline and online learning

1. Start with offline learning to learn baseline dynamics.
2. Switch to online learning to adapt to changes in the dynamics.

Offline learning setup:

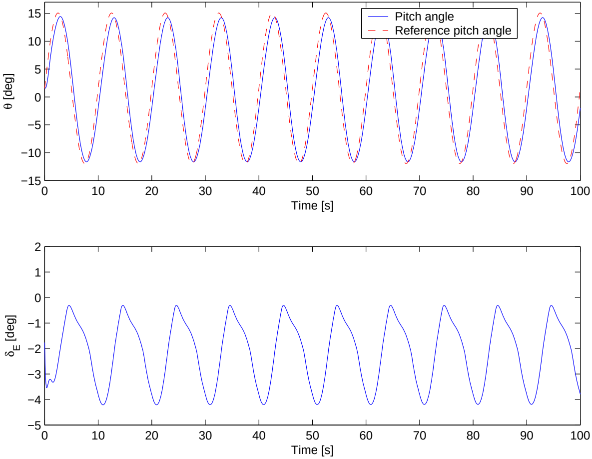
- Task: follow sinusoidal reference pitch angle while maintaining constant airspeed.
- 100 trials, each with different initial network weights.
- Each trial lasts at most 200 seconds.
- Evaluation: success ratio, percentage of trials where RMS of pitch angle error is below a certain threshold.

Online versus Offline learning

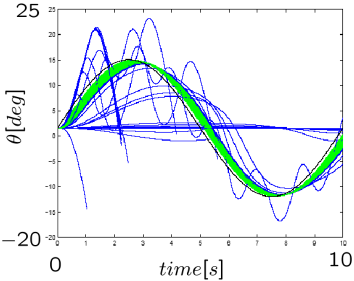


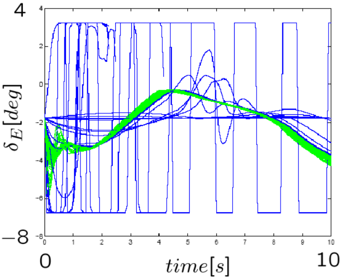
Online versus Offline learning

The trained RL-controller from the offline learning phase can be directly applied to the online learning phase:



Results offline learning





	ADHDP	HDP
$\Delta\theta_{RMS}$	4.45°	3.19°
success ratio	40%	78%

Online learning

- Start with actor, critic and plant neural network weights from one of the successful offline simulations.
- During the online simulation instantly change the plant dynamics.

Remember:

$$C_m = C_{m_0}(\alpha, \delta_e) + \left(\frac{qc}{2V_T}\right) C_{m_q}(\alpha) + \left(x_{c.g._{ref}} - x_{c.g.}\right) C_Z \qquad (13)$$

Changing parameters:

- Pitch damping coefficient: C_{m_q}
  - $C_{m_q} = 0$
  - $C_{m_q} = -C_{m_q}$
- Center of gravity $x_{c.g.}$

Set C_{m_q} to 0 after 30 s.

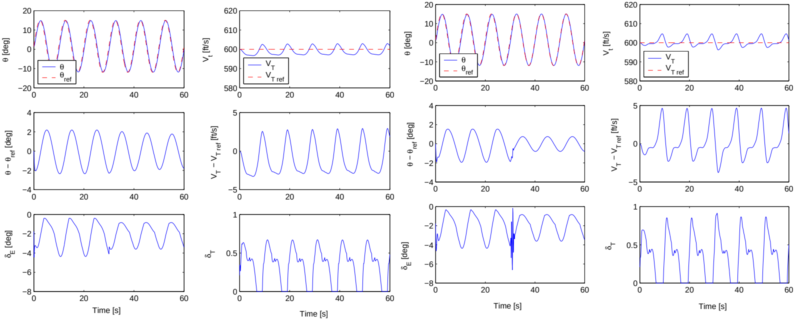


Figure: Left ADHDP, right HDP.

Set C_{m_q} to $-C_{m_q}$ after 30 s.

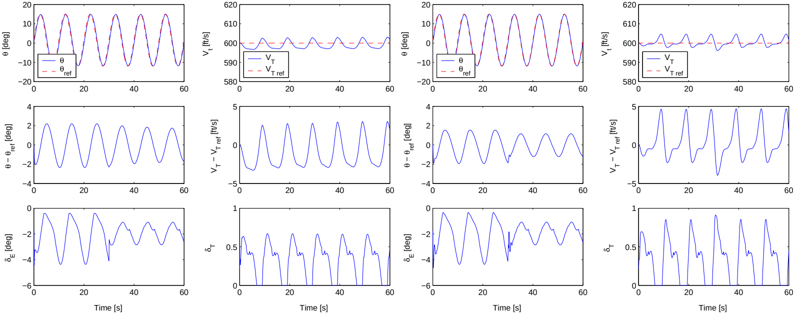


Figure: Left ADHDP, right HDP.

Change sign of $\left[x_{c.g.} - x_{c.g._{ref}}\right]$ after 30 s.

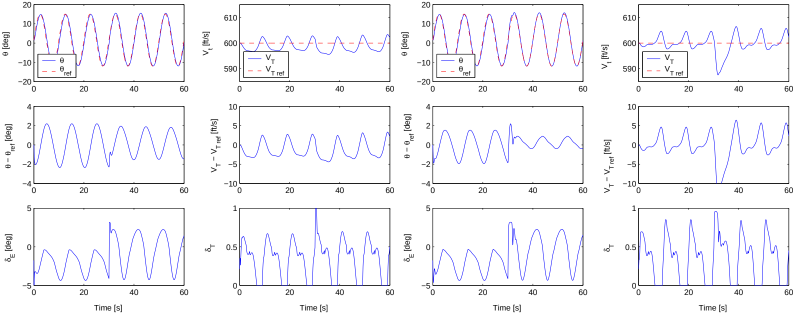


Figure: Left ADHDP, right HDP.

Current work - PhD-students at C&S

- Incremental Approximate Dynamic Programming, Ye Zhou
- Safe exploration for Reinforcement Learning, Tommaso Mannucci
- Intelligent flight control for Micro Aerial Vehicles, Jaime Junell

- Junell, J. L., Kampen, E.-J. Van, Visser, C. C. de, & Chu, Q. P. (2015). Reinforcement Learning Applied to a Quadrotor Guidance Law in Autonomous Flight. In AIAA Guidance, Navigation, and Control Conference. American Institute of Aeronautics and Astronautics. doi:doi:10.2514/6.2015-1990
- Mannucci, T., Kampen, E.-J. Van, Visser, C. C. de, & Chu, Q. P. (2015). SHERPA: a safe exploration algorithm for Reinforcement Learning controllers. In AIAA Guidance, Navigation, and Control Conference. American Institute of Aeronautics and Astronautics. doi:doi:10.2514/6.2015-1757
- Mannucci, T., van Kampen, E., de Visser, C. C., & Chu, Q. P. (2016). Graph based dynamic policy for UAV navigation. In 2016 AIAA Guidance, Navigation, and Control Conference.
- Mannucci, T., van Kampen, E., de Visser, C. C., & Chu, Q. P. (2016). A novel approach with safety metrics for real-time exploration of uncertain environments. In 2016 AIAA Guidance, Navigation, and Control Conference.
- Zhou, Y., Kampen, E.-J. Van, & Chu, Q. P. (2016). An Incremental Approximate Dynamic Programming Flight Controller Based on Output Feedback. In AIAA Guidance, Navigation, and Control Conference. American Institute of Aeronautics and Astronautics. doi:doi:10.2514/6.2016-0360
- Kampen, E.-J. van, Chu, Q. P., & Mulder, J. A. (2006). Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics. In AIAA Guidance, Navigation, and Control Conference and Exhibit. American Institute of Aeronautics and Astronautics. doi:doi:10.2514/6.2006-6429
- Junell, J., Mannucci, T., Zhou, Y., & Kampen, E.-J. Van. (2016). Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning. In AIAA Guidance, Navigation, and Control Conference. American Institute of Aeronautics and Astronautics. doi:doi:10.2514/6.2016-1387

Assignment

Develop a Reinforcement Learning controller for ANY system! You can choose for one of the examples below, but also apply the RL controler for different system.

Rules and requirements

- Discrete or Continuous RL
- Add a short report that includes the system description, RL controller design and some results.
- Show the learning effect and show a sensitivity analysis to the learning parameters.
- Please show me your plans before working out the details!

Some examples from the RL book by Sutton and Barto

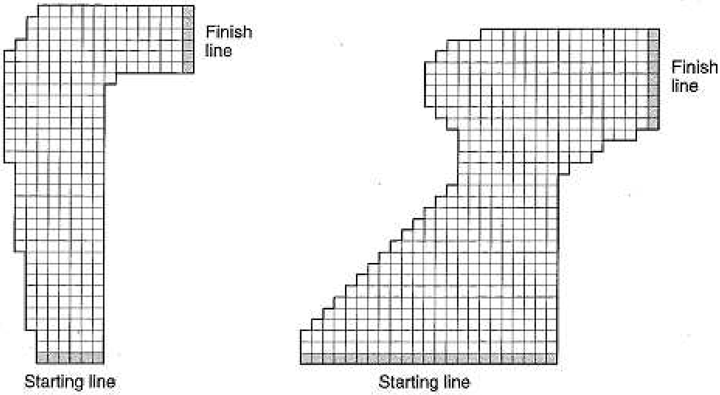


Figure 5.8 A couple of right turns for the racetrack task.

Some examples from the RL book by Sutton and Barto

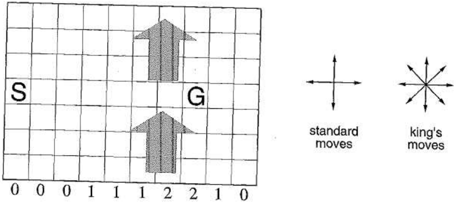
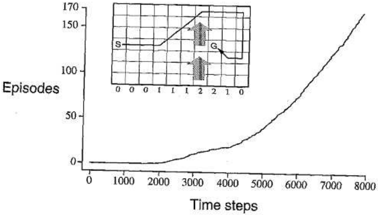
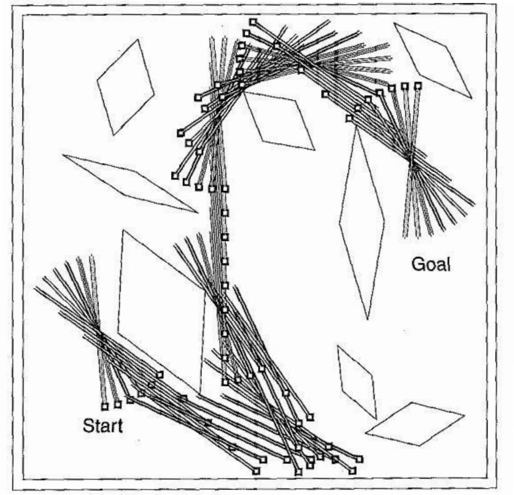


Figure 6.10 Gridworld in which movement is altered by a location-dependent, upward "wind."



Some examples from the RL book by Sutton and Barto



Some examples from the RL book by Sutton and Barto

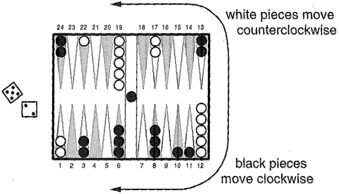


Figure 11.1 A backgammon position.

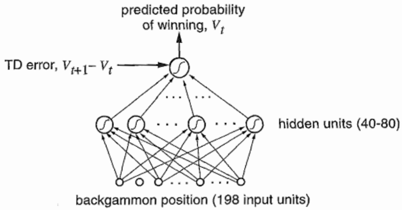


Figure 11.2 The neural network used in TD-Gammon.

Some examples from the RL book by Sutton and Barto

goal: Raise tip above line

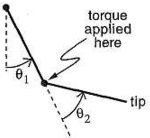


Figure 11.4 The acrobot.

$$\ddot{\theta}_1 = -d_1^{-1}(d_2\ddot{\theta}_2 + \phi_1)$$

$$\ddot{\theta}_2 = \left(m_2 l_{c2}^2 + I_2 - \frac{d_2^2}{d_1}\right)^{-1}\left(\tau + \frac{d_2}{d_1}\phi_1 - m_2 l_1 l_{c2}\dot{\theta}_1^2 \sin\theta_2 - \phi_2\right)$$

$$d_1 = m_1 l_{c1}^2 + m_2(l_1^2 + l_{c2}^2 + 2 l_1 l_{c2}\cos\theta_2) + I_1 + I_2$$

$$d_2 = m_2(l_{c2}^2 + l_1 l_{c2}\cos\theta_2) + I_2$$

$$\phi_1 = -m_2 l_1 l_{c2}\dot{\theta}_2^2 \sin\theta_2 - 2 m_2 l_1 l_{c2}\dot{\theta}_2\dot{\theta}_1 \sin\theta_2 + (m_1 l_{c1} + m_2 l_1) g \cos(\theta_1 - \pi/2) + \phi_2$$

$$\phi_2 = m_2 l_{c2} g \cos(\theta_1 + \theta_2 - \pi/2)$$

Some examples from the RL book by Sutton and Barto

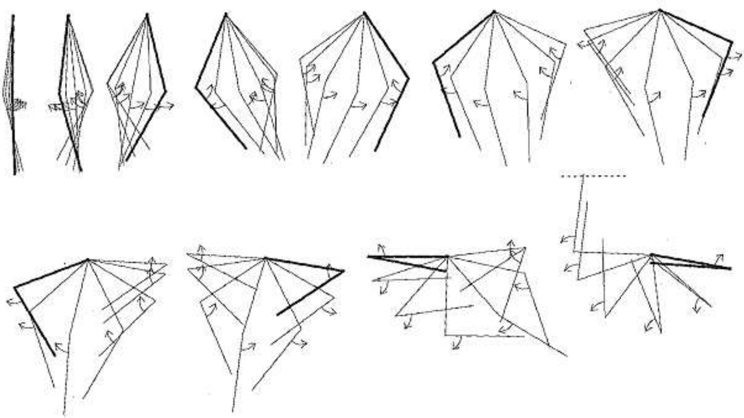


Figure 11.7 A typical learned behavior of the acrobot. Each group is a series of consecutive positions, the thicker line being the first. The arrow indicates the torque applied at the second joint.

Some examples from the RL book by Sutton and Barto

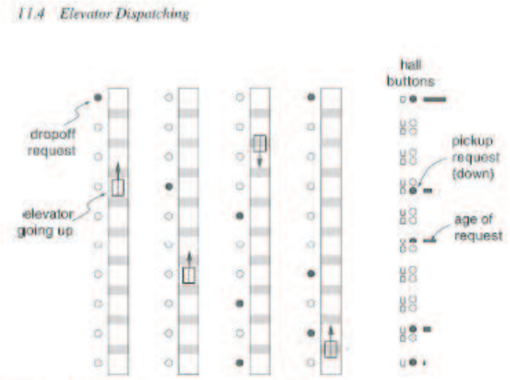


Figure 11.8 Four elevators in a ten-story building.

Other options

- Checkers
- Chess
- Poker
- etc.

Complexity of the system will be taken into account during the grading of the assignment.

Conclusions

Conclusions Reinforcement Learning

- Learning by interaction with the environment.
- Value function stores expected reward information.
- Continuous implementation requires function approximators.
- Discrete RL algorithms:
  - Dynamic Programming
  - Monte Carlo
  - Temporal Differences
  - Eligibility Traces

Want to read more? –> RL, Sutton and Barto, ISBN 0-262-19398-1.

Reinforcement Learning for Flight Control

AE4-311 Advanced Flight Control

Erik-Jan van Kampen, E.vanKampen@TUDelft.NL
Delft University of Technology
June 6, 2016