

Lecture Schedule

Dynamical programming

1 The finite-horizon decision problem

31 January

2 Dynamical Programming

7 February

3 DP reformulations and introduction to Control

14 February

Control

4 Discretization and PID control

21 February

5 Direct methods and control by optimization

28 February

6 Linear-quadratic problems in control

7 March

7 Linearization and iterative LQR

14 March

Syllabus: <https://02465material.pages.compute.dtu.dk/02465public>

Help improve lecture by giving feedback on DTU learn

Reinforcement learning

8 Exploration and Bandits

21 March

9 Policy and value iteration

4 April

10 Monte-carlo methods and TD learning

11 April

11 Model-Free Control with tabular and linear methods

18 April

12 Eligibility traces and value-function approximations

25 April

13 Q-learning and deep-Q learning

2 May

Reading material:

- [Her24, Chapter 4] Introduction

Learning Objectives

- Introduction and key definitions
- Python and object-oriented programming

Course practicalities

Course webpage



02465material.pages.compute.dtu.dk/02465public/index.html



Q Search ctrl + k

- Information ▼
- Models and Environments ▼
- Exercises ▼
- Projects ▼



Contents ▼

Practicalities

Time and place: 📍 Building B341, auditorium 21, 08:00–12:00
DTU Learn: 🏠 [02465](#)
Exercise code: 📄 <https://lab.compute.dtu.dk/02465material/02465students.git>
Course descriptions: 📖 [kurser.dtu.dk](#)
Discord: 🗨️ [Discord channel \(invitation link\)](#)
Campus-wide python support: 🐍 [pythonsupport.dtu.dk](#)
Contact: 📧 Tue Herlau, tuhe@dtu.dk

Note

This page is automatically updated with typos, etc. I therefore recommend bookmarking it and using the newest version of the exercises.

Course schedule

The schedule and reading can be found below. Click on the titles to read the exercise and project descriptions.

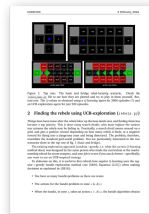
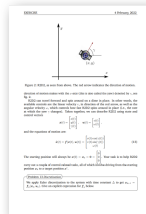
#	Date	Title	Reading	Homework	Exercise	Slides
	Jan 26th, 2024	Installation and self-test	Chapter 1-3 ★, [Her24]		[PDF]	
1	Feb 2nd, 2024	The finite-horizon decision problem	Chapter 4, [Her24]	1, 2	[PDF]	[1x] [6x]
2	Feb 9th, 2024	Dynamical Programming	Chapter 5-6.2, [Her24]	1, 2	[PDF]	[1x] [6x]
3	Feb 16th, 2024	DP reformulations and introduction to Control	Section 6.3; Chapter 10-11, [Her24]	1, 2	[PDF]	[1x] [6x]
4	Feb 23th, 2024	Discretization and PID control	Chapter 12-14, [Her24]	1,2	[PDF]	[1x] [6x]
	Feb 29th, 2024	</> Project 1: Dynamical Programming				
5	Mar 1st, 2024	Direct methods and control by optimization	Chapter 15, [Her24]	1,	[PDF]	[1x] [6x]
6	Mar 8th, 2024	Linear-quadratic problems in control	Chapter 16, [Her24]	tbid	[PDF]	[1x] [6x]
7	Mar 15th, 2024	Linearization and iterative LQR	Chapter 17, [Her24]	tbid	[PDF]	[1x] [6x]
8	Mar 22th, 2024	Exploration and Bandits	Chapter 1; Chapter 2-2.7; 2.9-2.10, [SB16]	tbid	[PDF]	[1x] [6x]

DTU Learn Announcements, assignment hand-ins, quizzes

Course homepage Exercises, projects, slides, documentation, installation,
etc. <https://02465material.pages.compute.dtu.dk/02465public>

Off-hours QA Discord. See link on homepage.

- Exercises
 - (Not confirmed) Building B341, auditorium 21
- Ask **project-related question** online so that everyone has the same information (i.e. not in class)



- Groups of 1, 2 or 3 students

Part 1 Dynamical programming (available now)

Part 2 Control

Part 3 Reinforcement Learning

- The projects are subject to DTUs rules of collaboration/Code of Conduct
 - This includes the individual programming.

Course practicalities

Exam

- The 4-hour written exam will contain:
 - Multiple-choice questions
 - Written-answer questions
 - Programming questions
- Test exams will be online later
- Exercises emphasize code-questions as I believe they test more skills
- Your evaluation is an overall assessment based on the written exam and project work
 - The project work is 20%.

sdtrem2024a Page 4 of 7

Part II: Conceptual questions

Question 6: Consider a control problem where a control signal $u(t) \in \mathbb{R}$ can be applied to a variable $x(t) \in \mathbb{R}$. The variable satisfies an angle, and it satisfy the following differential equation

$$\dot{x} = \sin(x) + u \quad (1)$$

Assume the optimal conditions at starting time $t_0 = 0$ is $x(0) = 0$ and $\dot{x}(0) = 0$. We introduce a state $w(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix}$ and the problem is discretized using Euler discretization with a time step of $\Delta t = 0.2$ to give states w_0, w_1, w_2, \dots . Our goal is to bring the system to a state $w = \begin{bmatrix} \pi \\ 0 \end{bmatrix}$ (which it should stand still).

(a) If we succeed at bringing the system to the target state at time T , how much control $u(t^*)$ do we need to apply to keep it there? Provide an argument for your answer.

(b) Assume we control signal $u(t) = 0$ is applied. According to Euler discretization, what is the value of w_1 at time $t_1 = \Delta t$?

(c) Assume we control signal $u(t) = 0$ is applied. According to Euler discretization, what is the value of w_2 at time $t_2 = 2\Delta t$?

(d) We want to control the system towards $w^* = \begin{bmatrix} \pi \\ 0 \end{bmatrix}$. Suppose we use a general time-independent quadratic cost function with no terminal-state contribution, $c(x) = 0$ and when the contribution from the system are simple

$$u_1(t_1, u_1) = \text{trace}(u_1^T \Lambda u_1)$$

Assume we select $Q_0 = I$, if what is an appropriate value of Λ ?

Question 7: Consider the dynamical programming setting where we plan over a horizon $N > 0$. We consider a problem where

- The terminal cost function is $J_N(x_N) = x_N^2$.
- For all $k = 0, \dots, N-1$ the dynamics is $f_k(x, u, w) = w + x + u + b$.
- The state disturbances are i.i.i.d. $P_k(x_k, u_k) = N(x_k | \mu = 0, \sigma^2 = 1)$.
- The state and actions are real numbers $D_k = A_k(x_k) = \mathbb{R}$.
- The non-terminal costs are only affected by x_k , $J_k(x_k, u_k, w) = x_k^2$.

(Note, the relevant parameters of the problem are a and b . We are concerned with optimal control.)

(a) Although the problem has been formulated as being about dynamical programming, note that the structure of the problem is that of a 1-dimensional LQR model. In the case where $b = 1$, derive the expected future cost $J_k^*(x_k) = \mathbb{E}[J_k(x_k, u_k, w_k) | x_k]$ if we use time step $k = N-1$ and we use state $x_{k-1} = 0$ and take action $u = 1$?

(b) Derive an analytical expression for the optimal policy $J_k^*(x_k)$ in time step $k = N-1$?

(c) Assume that $b = 0$ and $a = \sqrt{2}$. Derive the expected cost function $J_k^*(x_k)$ for all x and all $k \leq N$?

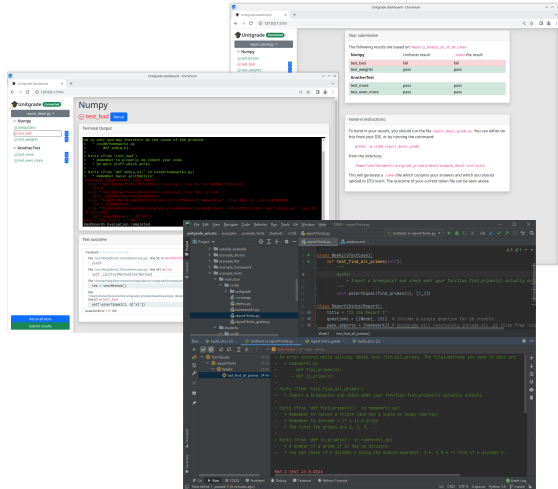


Course practicalities

Creating handins

See videos for week 0

- I hope this can help you debug code
- Example usage:
 - `python -m irlc.project0.fruit_project_grade`
 - Hand in your code/scores by uploading the `.token` file



Question 8

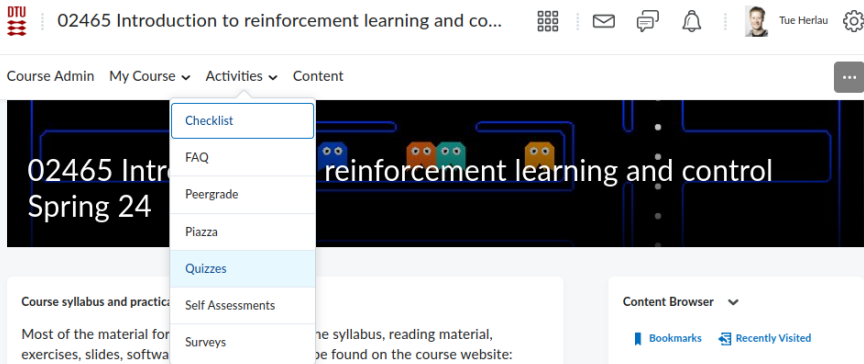
Should we have one or two 5-minute quizzes during the lectures? (Similar to 02450, introduction to machine learning and data mining)



I will try to use quizzes this semester. You can find them under Quizzes on DTU Learn:

Do you use ChatGPT or a similar conversational AI tools in your studies?

- Yes
- No



The screenshot shows the DTU Learn interface for the course '02465 Introduction to reinforcement learning and control'. The course title is partially visible as '02465 Intro Spring 24' and 'reinforcement learning and control'. A navigation menu is open, showing options: Checklist, FAQ, Peergrade, Piazza, Quizzes (highlighted), Self Assessments, and Surveys. The course syllabus and practicalities section is partially visible, mentioning 'Most of the material for exercises, slides, software...'. The content browser shows 'Bookmarks' and 'Recently Visited'.

- ChatTutor allows you to ask questions to **both** TAs and an AI (ChatGPT)
- The platform will collect the data you put in (i.e., same as any other webpage!)
 - But please ask if you have questions!
- Optional offer:
 - Available from next week
 - Work in progress – you will have other options if it is too janky :-).

What is reinforcement learning and control

Types of machine learning

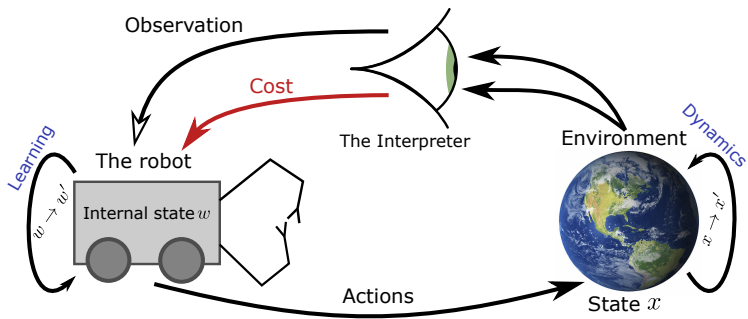


Supervised learning Learn a function $f(x_i) \mapsto \hat{y}_i$ to minimize a **loss**

Unsupervised learning Learn a **structure** to **summarize data**

What is reinforcement learning and control


Sequential decision making

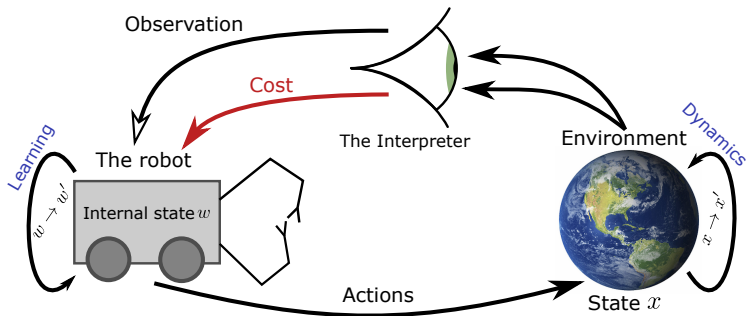


Make decisions, one after another, to bring about a desired outcome

- Observe the world
- Take action
- Obtain cost

Minimize total cost

 `lecture_01_pacman.py`



- Time is really important (sequential, non-i.i.d data)
- Must optimize behavior of dynamical systems using information that becomes progressively available as the systems evolve
- Future cost and state of the system will depend on current actions and state

What is reinforcement learning and control

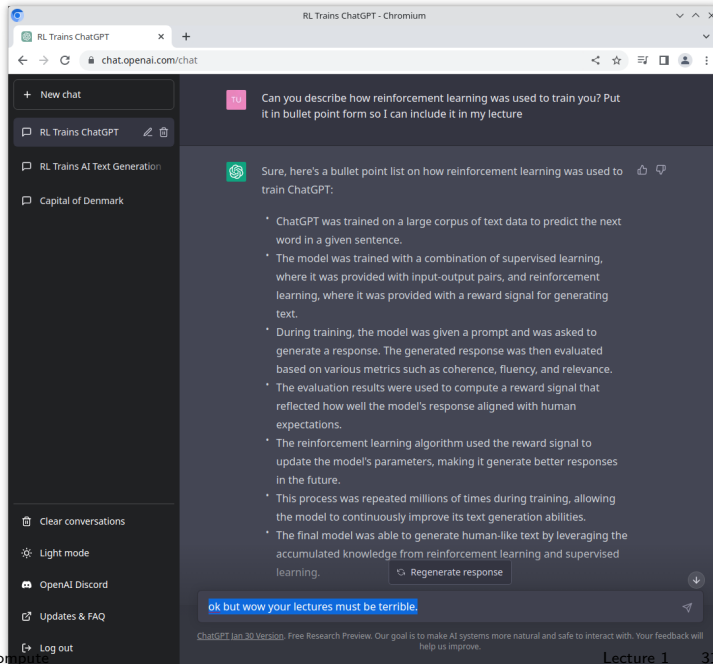
Alpha-Go



- Self-learning Go supercomputer
- Defeated world champion Lee Sedol in 2016
- Notable mentions: Atari/Dota/Starcraft II learners
- General approach: Reinforcement learning + Search trees

What is reinforcement learning and control

ChatGPT



The screenshot shows a web browser window titled "RL Trains ChatGPT - Chromium" with the URL "chat.openai.com/chat". The chat interface shows a user asking: "Can you describe how reinforcement learning was used to train you? Put it in bullet point form so I can include it in my lecture". The AI response is a bulleted list of seven points. At the bottom, a text input field contains the text "ok but wow your lectures must be terrible." and a "Regenerate response" button is visible.

RL Trains ChatGPT

chat.openai.com/chat

+ New chat

RL Trains ChatGPT

RL Trains AI Text Generation

Capital of Denmark

Clear conversations

Light mode

OpenAI Discord

Updates & FAQ

Log out

10 Can you describe how reinforcement learning was used to train you? Put it in bullet point form so I can include it in my lecture

Sure, here's a bullet point list on how reinforcement learning was used to train ChatGPT:

- ChatGPT was trained on a large corpus of text data to predict the next word in a given sentence.
- The model was trained with a combination of supervised learning, where it was provided with input-output pairs, and reinforcement learning, where it was provided with a reward signal for generating text.
- During training, the model was given a prompt and was asked to generate a response. The generated response was then evaluated based on various metrics such as coherence, fluency, and relevance.
- The evaluation results were used to compute a reward signal that reflected how well the model's response aligned with human expectations.
- The reinforcement learning algorithm used the reward signal to update the model's parameters, making it generate better responses in the future.
- This process was repeated millions of times during training, allowing the model to continuously improve its text generation abilities.
- The final model was able to generate human-like text by leveraging the accumulated knowledge from reinforcement learning and supervised learning.

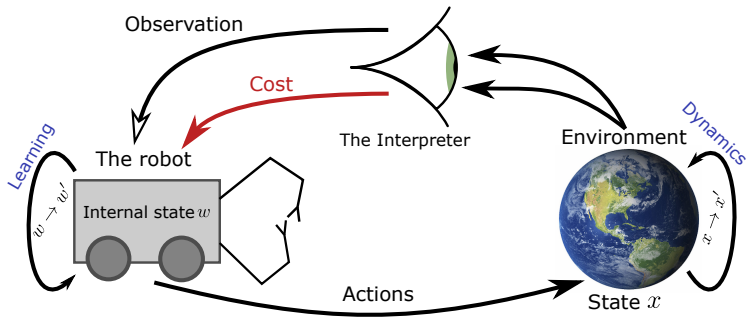
Regenerate response

ok but wow your lectures must be terrible.

ChatGPT Jan 30 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

What is reinforcement learning and control

The decision problem




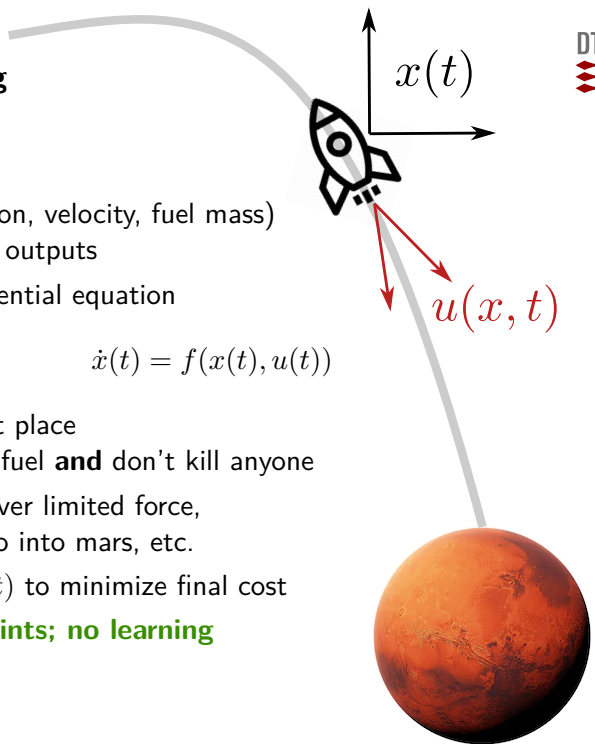
State The configuration of the environment x

Action Either discrete or a vector u

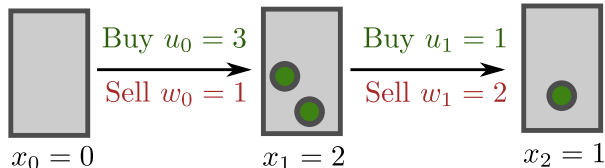
Cost/reward A number. Depends on state x and action u

Example: Mars landing**Time** Continuous**State/Actions** $x(t)$: (Position, velocity, fuel mass)
 $u(t)$: thruster outputs**Dynamics** Smooth differential equation

$$\dot{x}(t) = f(x(t), u(t))$$

Cost Land the right place
and use little fuel **and** don't kill anyone**Constraints** Thrusters deliver limited force,
ship cannot go into mars, etc.**Objective** Determine $u(t)$ to minimize final cost**Really important constraints; no learning** `lecture_01_car_random.py`

Inventory control



- We order a quantity of an item at period $k = 0, \dots, N - 1$ so as to meet a stochastic demand

x_k stock at the beginning of the k th period,

$u_k \geq 0$ stock ordered at the beginning of the k th period.

$w_k \geq 0$ Demand during the k 'th period

- Dynamics: $x_{k+1} = x_k + u_k - w_k$
- Cost per new unit c ; cost to hold x_k units is $r(x_k)$

$$r(x_k) + cu_k$$

- Select actions u_0, \dots, u_{N-1} to minimize cost

We want proven optimal rule for ordering

Example: Atari



States RAM memory state

Observations Pixel-based snapshots $H \times W \times 3$

Actions Discrete joystick actions

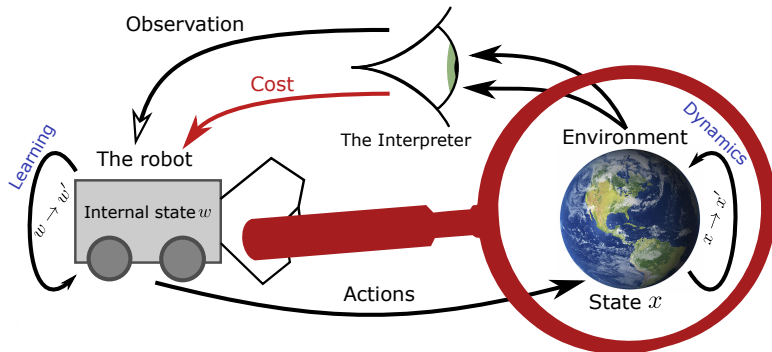


Dynamics Discrete, stochastic (what the emulator does)

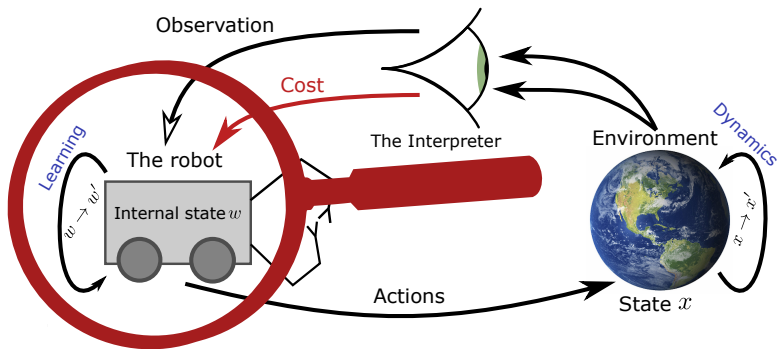
Cost High-score



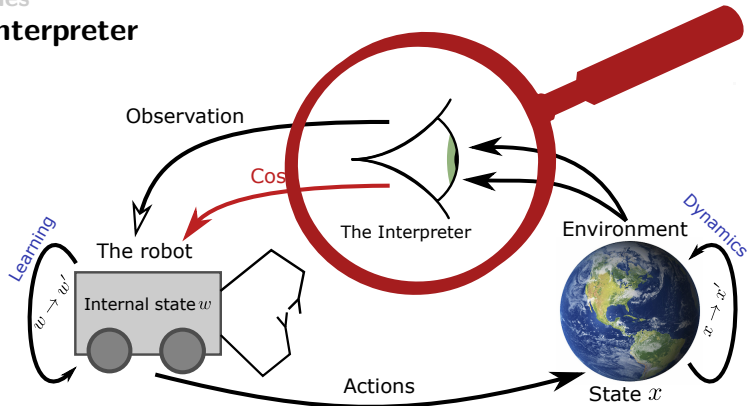
Don't know dynamics; must learn from scratch



- Nature can be stochastic or deterministic
- The problem can be continuous-time or discrete-time
- We can know the dynamics or not



Policy How the robot chooses actions at given times/states



Reward The **immediate** evaluation of current step

Agents goal Maximize **cumulative** reward

Reward Hypothesis

Every desired behavior of the agent can be described by the maximization of expected cumulative reward

Making sense of these distinctions

- Why so many things in one course?
 - Study-line requirement
 - A single problem, and a single solution + tricks
 - A better overview (right tool for the job)
- Today, we will look at the problem

Finite time Problem starts at time 0 and terminates at time N . Indexed as $k = 0, 1, \dots, N$.

State space The states x_k belong to the **state space** \mathcal{S}_k

Control The available controls u_k belong to the **action space** $\mathcal{A}_k(x_k)$, which may depend on x_k

Dynamics

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \dots, N - 1$$

Disturbance/noise A random quantity w_k with distribution

$$w_k \sim P_k(W_k|x_k, u_k)$$

Cost and control

Agent observe x_k , agent choose u_k , environment generates w_k

Cost At each stage k we obtain cost

$$g_k(x_k, u_k, w_k), \quad k = 0, \dots, N-1 \quad \text{and} \quad g_N(x_k) \text{ for } k = N.$$

Action choice Chosen as $u_k = \mu_k(x_k)$ using a function $\mu_k : \mathcal{S}_k \rightarrow \mathcal{A}_k(x_k)$

$$\mu_k(x_k) = \{\text{Action to take in state } x_k \text{ in period } k\}$$

Policy The collection $\pi = \{\mu_0, \mu_1, \dots, \mu_{N-1}\}$

Rollout of policy Given x_0 , select $u_k = \mu_k(x_k)$ to obtain a **trajectory**

$x_0, u_0, x_1, \dots, x_N$ and **accumulated cost**

$$\text{Cost-of-rollout} = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)$$

Expected return (approximate) Generate T rollouts according to π

$$J_\pi(x_0) \approx \frac{1}{T} \sum_{i=1}^T \{\text{Cost-of-rollout } i\}$$

Quiz 1: Discuss and answer on DTU Learn

How do you feel about this argument? Justify your answer:

Decision-making is about determining the appropriate sequence of actions

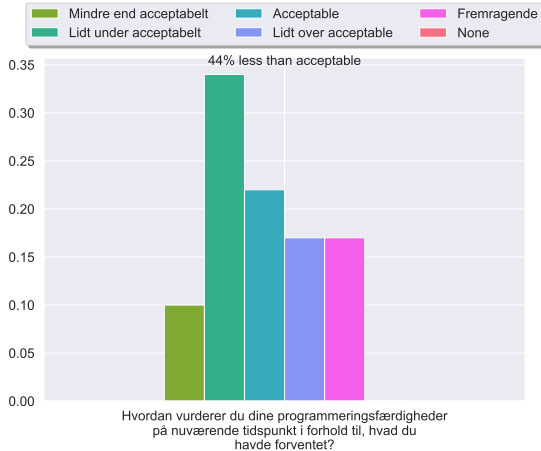
u_0, \dots, u_{N-1} .

Once executed, we get a total cost. Let's say that on average this is $c(\mathbf{u})$.

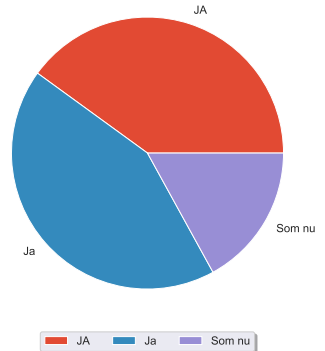
Thus, decision-making is ultimately an optimization problem: Find the sequence that on average minimize the cost:

$$u_0, \dots, u_{N-1} = \arg \min_{\mathbf{u}} c(\mathbf{u}).$$

- a. It is computationally too complicated to solve such an optimization problem
- b. It is infeasible to derive or learn the function $c(\mathbf{u})$
- c. Actually nothing is wrong: It is just not a theoretically interesting/fruitful way to approach decision-making
- d. Something else is wrong with the argument
- e. Don't know



Vil du gerne have mere programmering på 1. år?



Programming

Pre-semester quiz



```
1 # chapter1/lecture1_code.py
2 class MyClass:
3     def __init__(self, a):
4         self.my_variable = a
5
6     def some_function(self):
7         print("The variable I got was", self.my_variable)
8
9 class MyOtherClass(MyClass):
10     def __init__(self, a, b):
11         super().__init__(a)
12         print("I also got", b)
```

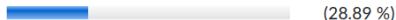
This is new -- I have not used class inheritance before.
The code is mysterious.



I have seen code like this before, but it is not something I
have used. I think I can pick it up.



I have written code that inherit from other classes (i.e.,
something like the second class). I am not an expert, but
it is not something that worries me



This is easy. I have written code like this before and can
reason about what it does.



What I have done:

- Re-structured the project work
- Simplification of exercises + videos
- Course notes on Python + online documentation
- This lecture
- Changed exam format
- Course responsible for the new mandatory programming course (02002/3)

What I hope you will do:

- Decide to learn this – you can!
- Set aside some time in the first block
- Don't give up:
 - Programming was not taught correctly – 100% valid criticism
 - You need to learn new programming techniques through your career

Programming

Pacman game loop (without objects)



```
1 # chapter1/lecture1_code.py
2 walls = np.ndarray( ) # Initialize a walls-variable
3 food = np.ndarray( )
4 pacman_x = 4
5 pacman_y = 6
6
7 for k in range(10):
8     # Use the walls and pacman_x, pacman_y to figure out what actions are available.
9     available_actions = ... # compute using the walls-variable
10    # Do some sort of planning (search?) by using the walls, pacman_x, pacman_y.
11    # select the best possible action
12    # Compute the outcome of the action:
13    pacman_x = pacman_x + action_x
14    pacman_y = pacman_y + action_y
15    # Compute the reward
16    # Let the agent learn based on the outcome and reward
```

(about 500 lines total)

Same with two agents and two environments



```
1 # chapter1/lecture1_code.py
2 for k in range(10):
3     if environment_type == 2:
4         available_actions = ... # compute using the walls-variable
5     else:
6         available_actions = ... # This environment may differ
7     if agent_type == 1: # Agent plan it's actions
8         pass # do planning of first type
9     elif agent_type == 2:
10        pass # do planning of the second type
11    if environment_type == 1: # Compute the outcome of the action:
12        pacman_x = pacman_x + action_x
13        pacman_y = pacman_y + action_y
14        # Compute the cost-function
15    else:
16        pass # Updates relevant for second environment
17        # Compute the cost function
18    if agent_type == 2: # Allow the agent to learn based on cost
19        pass # Learning for the second agent
20    else:
21        pass # Learning method for the first agent
```

```
1 # chapter1/lecture1_code.py
2 env = InventoryEnvironment() # Create an instance of the inventory environment
3 agent = RandomAgent(env) # Create an instance of a random-action agent
4 train(env, agent) # Train the agent
```

Training-function:

```
1 # chapter1/lecture1_code.py
2 def train(env, agent):
3     s = env.reset() # Reset and get first state, x_0
4     for k in range(10):
5         a = agent.pi(s) # The policy computes the action
6         sp, r, done = env.step(a) # Environment computes next state, reward
7         agent.train(s, a, sp, r, done) # Let the agent train
```



(this is a very rough sketch. We'll get to the real training function soon)

Programming

The simplest class



The smallest and friendliest `class`

```
1 >>> class BasicClass: # Classnames are usually upper-case
2 ...     pass         # `pass` is a special keyword which does nothing
3 ...
```

Each class **instance** function like it's own little box of variables:

```
1 >>> a = BasicClass() # Create an instance of the class
2 >>> a.name = "My first class" # You can write data to the class like this
3 >>> b = BasicClass() # Another instance. a and b are not related and can store different data:
4 >>> b.name = "Another class"
5 >>>
6 >>> print("Class a:", a.name)
7 Class a: My first class
8 >>> print("Class b:", b.name)
9 Class b: Another class
```

Programming

A class with a function



```
1 >>> class BasicDog:
2     ...     name = "Unnamed dog" # Each dog-instance will have the property name
3     ...     def read_nametag(self):
4     ...         # This is a class-function. Note we must pass it `self` as a first argument,
5     ...         # instance of the class itself (i.e. the current object). This is how we can
6     ...         print("This dog is named", self.name, "please give me treats!")
7     ...
8 >>> dog = BasicDog()
9 >>> dog.name
10 'Unnamed dog'
```

`self` refers to the class instance

```
1 >>> dog.read_nametag() # Invoke the read_nametag() function. Note we don't pass the obj
2 This dog is named Pluto please give me treats!
```

`def __init__` function is called when the class is created

```
1 >>> class BetterBasicDog:
2 ...     def __init__(self, name):
3 ...         self.name = name
4 ...         self.age = 0
5 ...         print(f"The __init__() function has been called with name='{name}'")
6 ...     def birthday(self):
7 ...         self.age = self.age + 1
8 ...         print("Hurray for", self.name, "you are now", self.age, "years old")
9 ...
```

Arguments can be passed along like this

```
1 >>> d1 = BetterBasicDog("Pluto")           # the __init__ function is now called
2 The __init__() function has been called with name='Pluto'
3 >>> d2 = BetterBasicDog(name="Lassie")     # Also support named arguments
4 The __init__() function has been called with name='Lassie'
```

Functions can change the `state` of the class

```
1 >>> d1.birthday()
2 Hurray for Pluto you are now 1 years old
3 >>> d1.birthday()
4 Hurray for Pluto you are now 2 years old
```

```

1 >>> class BetterBasicDog:
2 ...     def __init__(self, name):
3 ...         self.name = name
4 ...         self.age = 0
5 ...         print(f"The __init__() function has been called with name='{name}'")
6 ...     def birthday(self):
7 ...         self.age = self.age + 1
8 ...         print("Hurray for", self.name, "you are now", self.age, "years old")
9 ...
10 >>> d1 = BetterBasicDog("Pluto")
11 The __init__() function has been called with name='Pluto'

```

```

1 # chapter0pythonC/quiz.py
2 d1 = BetterBasicDog("Pluto")
3 d1.birthday()
4 d1.age = 5
5 d1.name = "Lassie"
6 d1.birthday()

```

- a. Ignore changes and prints out "Hurray for Pluto you are now 1 years old"
- b. Accept changes and prints out "Hurray for Lassie you are now 6 years old"
- c. It gives an error – it is not possible to set the age.
- d. It uses `name` but ignores `age`, so we get:
"Hurray for Lassie you are now 1 years old"

```
1 >>> class Parrot:
2 ...     def __init__(self):
3 ...         self.words = ["Squack!"]
4 ...     def learn(self, word):
5 ...         self.words.append(word)
6 ...     def speak(self):
7 ...         return random.choice(self.words) # Return a random word
8 ...     def vocabulary(self):
9 ...         return self.words
10 ...
```

```
1 >>> parrot = Parrot()
2 >>> words = ["sugar", "sleep well", "(parrot noises)", "*honk*"]
3 >>> for word in words:
4 ...     parrot.learn(word)
5 ...
6 >>> for _ in range(3): # Say three words
7 ...     parrot.speak()
8 ...
9 '(parrot noises)'
10 'Squack!'
11 'sleep well'
12 >>> print("Vocabulary", parrot.vocabulary())
13 Vocabulary ['Squack!', 'sugar', 'sleep well', '(parrot noises)', '*honk*']
```

```
1 >>> class Parrot:
2 ...     def __init__(self):
3 ...         self.words = ["Squack!"]
4 ...     def learn(self, word):
5 ...         self.words.append(word)
6 ...     def speak(self):
7 ...         return random.choice(self.words) # Return a random word
8 ...     def vocabulary(self):
9 ...         return self.words
10 ...
```

`ForgetfulParrot` : Is like the regular `Parrot` , except the learn-function

```
1 >>> class ForgetfulParrot(Parrot):
2 ...     # The Parrot class is used as a template.
3 ...     # All functions in the Parrot-class are therefore 'imported' as default, including 'self.words'
4 ...     def learn(self, word): # This function overwrite the 'actual' learn function in the Parrot class
5 ...         self.words = [word] # This parrot only know a single word
6 ...
```

Inheritance: The functions are "copy-pasted" into the `ForgetfulParrot`

```
1 >>> old_parrot = ForgetfulParrot()
2 >>> old_parrot.learn("damn remote")
3 >>> old_parrot.learn("Jeopardy")
4 >>> print("Vocabulary", old_parrot.vocabulary())
5 Vocabulary ['Jeopardy']
```

More **inheritance**: Make a squeak before and after every word:

```
1 >>> class Parrot:
2 ...     def __init__(self):
3 ...         self.words = ["Squack!"]
4 ...     def learn(self, word):
5 ...         self.words.append(word)
6 ...     def speak(self):
7 ...         return random.choice(self.words) # Return a random word
8 ...     def vocabulary(self):
9 ...         return self.words
10 ...
```

Where is the bug?

```
1 >>> class BadSqueekyParrot(Parrot):
2 ...     def __init__(self, squeek="Quck!"):
3 ...         self.squeek = squeek
4 ...     def speak(self):
5 ...         return f"{self.squeek} {random.choice(self.words)} {self.squeek}"
6 ...
7 >>> squeeky = BadSqueekyParrot(squeek="Kvak-Kvak")
8 >>> squeeky.learn("Good night!")
9 Traceback (most recent call last):
10   File "<console>", line 1, in <module>
11   File "<console>", line 5, in learn
12 AttributeError: 'BadSqueekyParrot' object has no attribute 'words'
```

Use `super()` to access functions in the parent class

```

1  >>> class SqueekyParrot(Parrot):
2  ...     def __init__(self, squeek="Quck!"):
3  ...         super().__init__() # Call the 'Parrot' class __init__ method to set up the words-variable.
4  ...         self.squeek = squeek # save the squeek variable
5  ...     def speak(self):
6  ...         word = super().speak() # Use the speak() function defined in the Parrot class.
7  ...         return f"{self.squeek} {word} {self.squeek}"
8  ...
9  >>> squeeky = SqueekyParrot(squeek="Kvak-Kvak")
10 >>> squeeky.learn("Good night!")
11 >>> squeeky.learn("Tell that damn bird to shut it's beak")
12 >>> squeeky.learn("Sugar!")
13 >>> squeeky.speak()
14 'Kvak-Kvak Good night! Kvak-Kvak'
15 >>> squeeky.speak()
16 'Kvak-Kvak Sugar! Kvak-Kvak'

```

Consistency When we inherit from `Parrot`, we **know** the functions should be called `speak`, `learn` (and not `talk`, `practice`)

- Env : (`reset`, `step`, `action_space` and a few other)
- Agent : (`pi`, `train`)

Functionality By using `super().__init__` we saved a single line

- In control theory, we will use inheritance to add simulation-functionality to all models


```

1  # inventory_environment.py
2  class InventoryEnvironment(Env):
3      def __init__(self, N=2):
4          self.N = N                                # planning horizon
5          self.action_space = Discrete(3)          # Possible actions {0, 1, 2}
6          self.observation_space = Discrete(3)     # Possible observations {0, 1, 2}
7
8      def reset(self):
9          self.s = 0                                # reset initial state x0=0
10         self.k = 0                                # reset time step k=0
11         return self.s, {}                         # Return the state we reset to (and an
12
13     def step(self, a):
14         w = np.random.choice(3, p=(.1, .7, .2))   # Generate random disturbance
15         s_next = max(0, min(2, self.s-w+a))        # next state; x_{k+1} = f_k(x_k,
16         reward = -(a + (self.s + a - w)**2)       # reward = -cost = -g_k(x_k,
17         terminated = self.k == self.N-1          # Have we terminated? (i.e. is k=
18         self.s = s_next                           # update environment state
19         self.k += 1                                # update current time step
20         return s_next, reward, terminated, False, {} # return transition information

```

Recall $x_{k+1} = x_k - w_k + a_k$ (clipped at 0 and 2) and e.g. $P(w = 0) = \frac{1}{10}$

```
1 # inventory_environment.py
2 class RandomAgent(Agent):
3     def pi(self, s, k, info=None):
4         """ Return action to take in state s at time step k """
5         return np.random.choice(3) # Return a random action
```

- The policy $\mu_k(x_k)$ corresponding to `pi(x, k, info)`
- A training function which is given x_k , u_k and x_{k+1} plus obtained reward plus additional information
- In each exercise session, you will write at least one agent
- Look at the `Agent`-class
- `truncated=False` ; `info` is 'extra information' (see documentation)

The `train`-function computes an episode as follows:

```
1  # inventory_environment.py
2  def simplified_train(env: Env, agent: Agent) -> float:
3      s, _ = env.reset()
4      J = 0 # Accumulated reward for this rollout
5      for k in range(1000):
6          a = agent.pi(s, k)
7          sp, r, terminated, truncated, metadata = env.step(a)
8          agent.train(s, a, sp, r, terminated)
9          s = sp
10         J += r
11         if terminated or truncated:
12             break
13     return J
```

Above computes the sum-of-reward for one episode:

```
1  # inventory_environment.py
2  env = InventoryEnvironment()
3  agent = RandomAgent(env)
4  stats, _ = train(env, agent, num_episodes=1, verbose=False) # Perform one rollout.
5  print("Accumulated reward of first episode", stats[0]['Accumulated Reward'])
```

Approximate

$$J_{\pi}(x_0) = \mathbb{E} \left[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right] \quad (1)$$

As average over 1000 trajectories

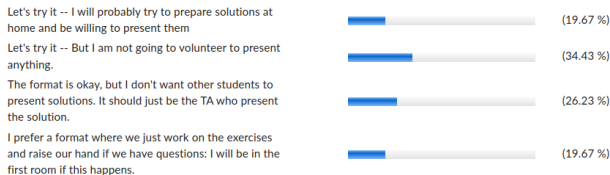
```
1 # inventory_environment.py
2 stats, _ = train(env, agent, num_episodes=1000, verbose=False) # do 1000 rollouts
3 avg_reward = np.mean([stat['Accumulated Reward'] for stat in stats])
4 print("[RandomAgent class] Average cost of random policy J_pi_random(0)=", -avg_reward)
```

Bob has $x_0 = 20$ kroner. He can either:

- Action $u = 0$: Put them in the bank at a 10% interest, thereby ending up with 22 kroner.
- Action $u = 1$: Lend them to a friend.
 - With probability $\frac{1}{4}$ he loses everything
 - With probability $\frac{3}{4}$ his friend gives him 12 kroner (aka one beer) as a thank you, and thus he will have $20 + 12 = 32$ kroner total.


Bobs goal is to decide whether to put his money in the bank, or lend them to his friend. Which one of the following statements are correct:

- The state spaces are $\mathcal{S}_k = \{1, 2, \dots, 32\}$.
- The dynamics is $f_0(x_0, u_0, w_0) = 1.1x_0 + \frac{3}{4}(x_0 + 12u_0)$.
- The action space is $\mathcal{A}_0(x_0) = \{0, 1\}$
- It is not possible to determine an optimal policy since we don't know what Bobs friend will do.



- IT015: Passive exercises; installation problems
- Aud.21 + IT019: Interactive exercises.
Try to prepare and present homework exercises.

1 Bobs financially challenged friend

 Bob has $x_0 = 20$ kroner. He can either:

- Action $u = 0$: Put them in the bank at a 10% interest, thereby ending up with 22 kroner.
- Action $u = 1$: Lend them to a friend.
 - With probability $\frac{1}{4}$ he loses everything
 - With probability $\frac{3}{4}$ his friend gives him 12 kroner (aka one beer) as a thank you, and thus he will have $20 + 12 = 32$ kroner total.



Tue Herlau.

Sequential decision making.

(Freely available online), 2024.