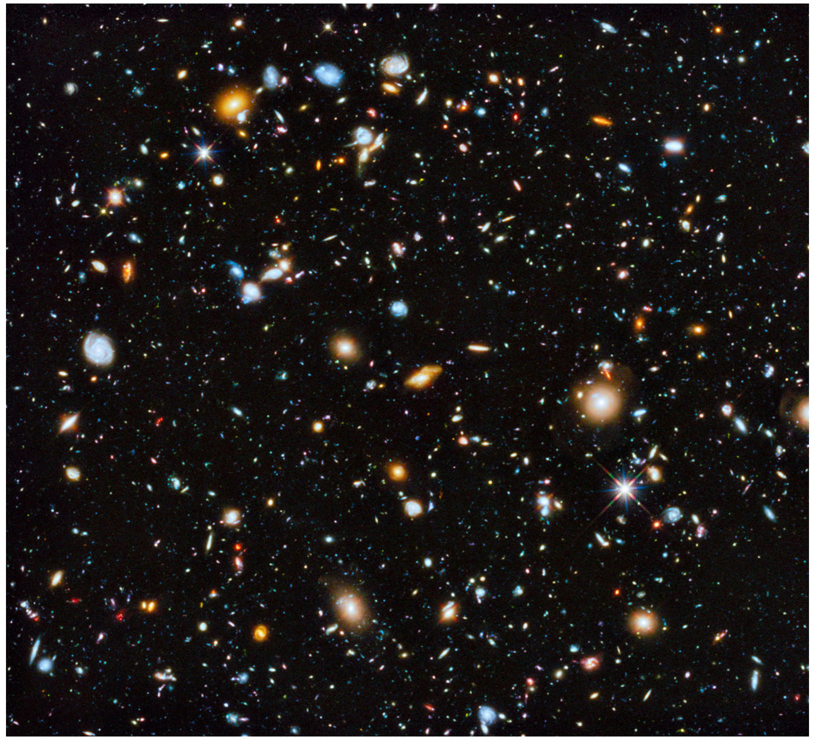
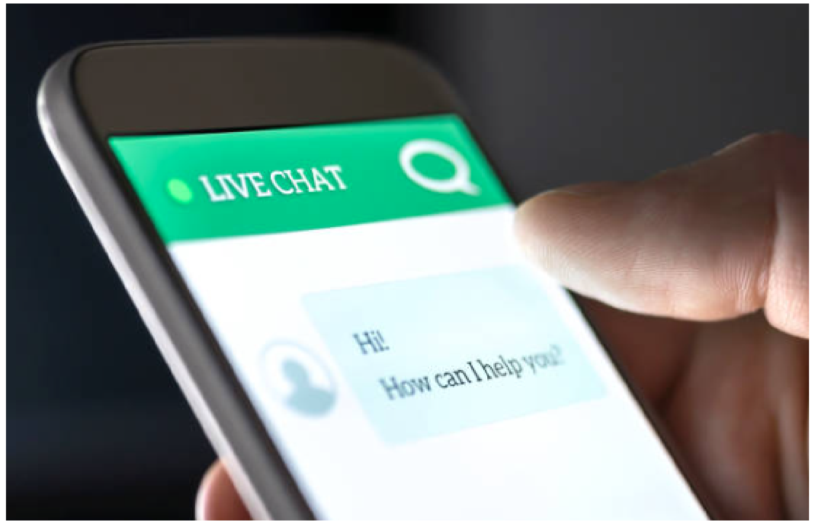
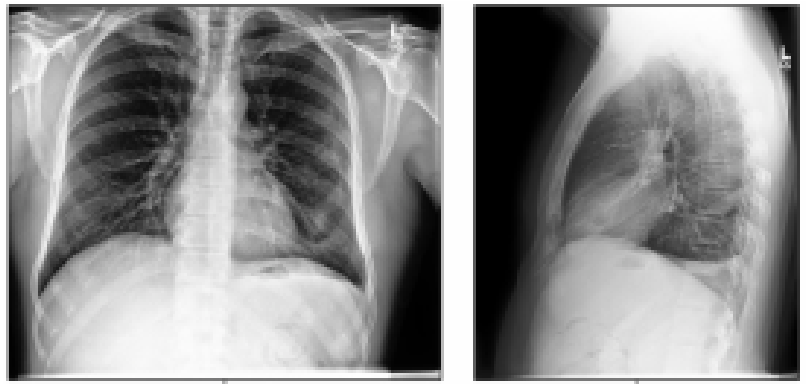
General: Course content



- Let's now talk about what we will cover in this course. In this module, we'll cover the modeling-inference-learning paradigm, which will help us tackle complex real-world problems. Within this paradigm, we'll survey the different types of methods that we will encounter in this course.

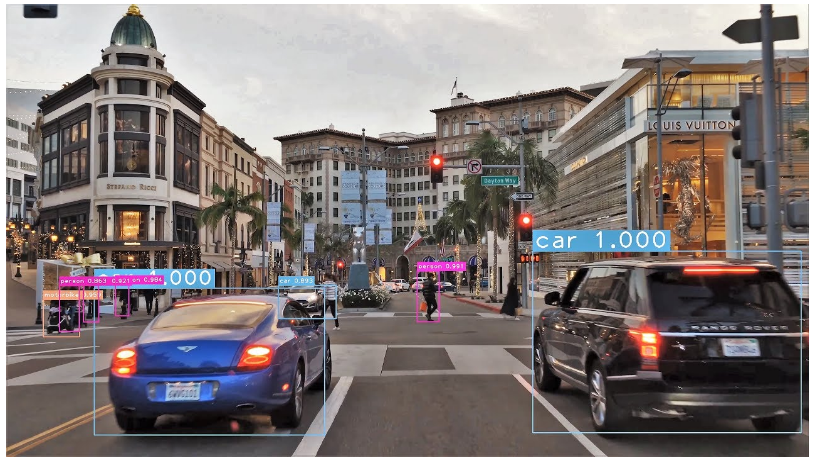
Complex real-world problems



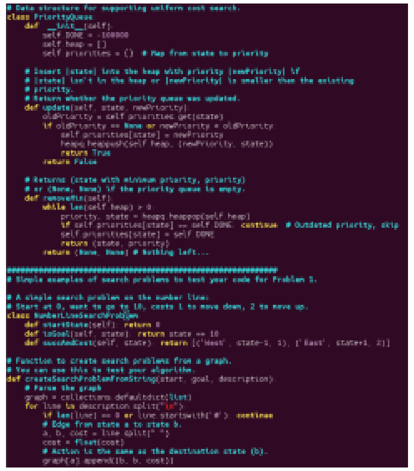


- AI is about using technology to tackle complex real-world problems.
- What are some complex real-world problems that come to mind?
- Self-driving cars have to sense the complex scene in front of them (in real-time!) and make safe decisions that advance the car towards its destination.
- Virtual assistants have to understand the rich subtleties of natural language, infer the user intent, and generate appropriate responses.
- Medical systems have to interpret the subtle, noisy cues in medical images, take into account the messy incomplete patient information, and generate reliable diagnoses that can be trusted by doctors.
- These are really really hard problems. How do you even begin to tackle them?

Bridging the gap



?



```
# Data structure for supporting uniform cost search.
class PriorityQueue:
    def  __init__(self):
        self.DONE = -100000
        self.heap = []
        self.priorities = {}  # Map from state to priority

    # Insert [state] into the heap with priority [newPriority] if
    # [state] isn't in the heap or [newPriority] is smaller than the existing
    # priority.
    # Return whether the priority queue was updated.
    def update(self, state, newPriority):
        oldPriority = self.priorities.get(state)
        if oldPriority == None or newPriority < oldPriority:
            self.priorities[state] = newPriority
            heapq.heappush(self.heap, (newPriority, state))
            return True
        return False

    # Returns (state with minimum priority, priority)
    # or (None, None) if the priority queue is empty.
    def removeMin(self):
        while len(self.heap) > 0:
            priority, state = heapq.heappop(self.heap)
            if self.priorities[state] == self.DONE: continue  # Outdated priority, skip
            self.priorities[state] = self.DONE
            return (state, priority)
        return (None, None) # Nothing left...

############################################################
# Simple examples of search problems to test your code for Problem 1.

# A simple search problem on the number line:
# Start at 0, want to go to 10, costs 1 to move down, 2 to move up.
class NumberLineSearchProblem:
    def startState(self): return 0
    def isGoal(self, state): return state == 10
    def succAndCost(self, state): return [('West', state-1, 1), ('East', state+1, 2)]

# Function to create search problems from a graph.
# You can use this to test your algorithm.
def createSearchProblemFromString(start, goal, description):
    # Parse the graph
    graph = collections.defaultdict(list)
    for line in description.split("\n"):
        if len(line) == 0 or line.startswith('#'): continue
        # Edge from state a to state b.
        a, b, cost = line.split(" ")
        cost = float(cost)
        # Action is the same as the destination state (b).
        graph[a].append((b, b, cost))
```

- At the end of the day, we need to write some code (and possibly build some hardware, too).
- But there is a huge chasm between the problem and the solution.
- You don't want to just start jumping in and writing code without a clear plan of attack.

Paradigm

Modeling

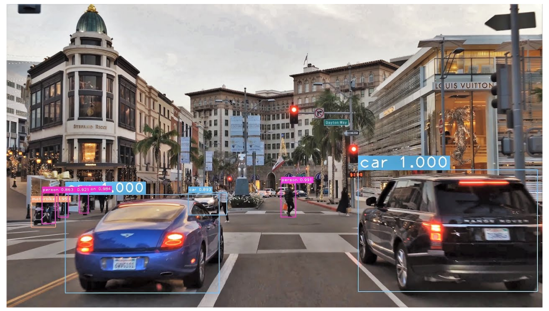
Inference

Learning

- So here is the plan of attack. We will adopt the **modeling-inference-learning** paradigm to help us navigate the solution space.
- There are three pillars, modeling, inference, and learning.
- In reality, the lines between the three pillars are blurry, but this paradigm serves as an ideal and a useful guiding principle.

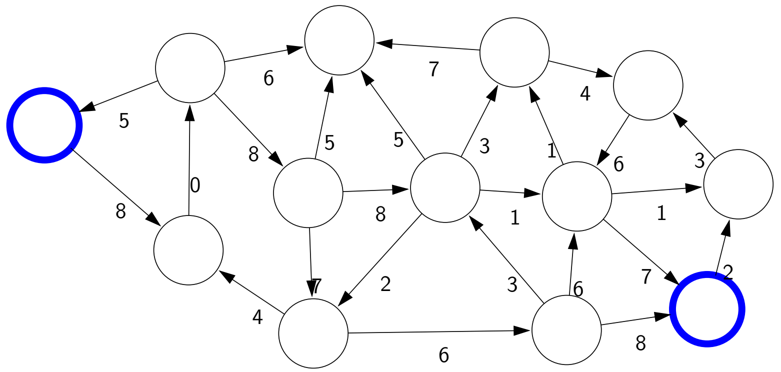
Paradigm: modeling

Real world



Modeling

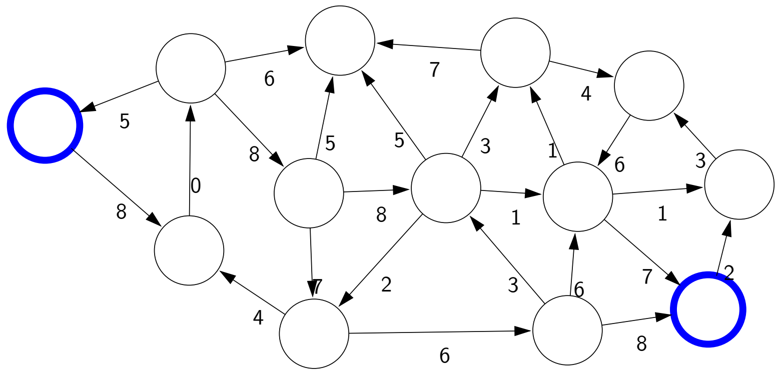
Model



- The first pillar is modeling.
- Modeling takes messy real world problems and packages them into neat formal mathematical objects called **models**, which can be subject to rigorous analysis and can be operated on by computers.
- However, modeling is lossy: not all of the richness of the real world can be captured, and therefore there is an art of modeling: what does one keep versus ignore?
- (An exception to this are games such as Chess, Go, or Sudoku, where the real world is identical to the model.)
- We might formulate the driving problem as a route finding problem as a graph where nodes represent points in the city, edges represent the roads, and the cost of an edge represents the traffic on that road.
- If we do this, all the complexities of perception are ignored. Alternatively, we could make a model that only looks at perception and ignores planning.
- It is worth noting that lossy modeling can lead to errors in the AI system, which can have an adverse impact on people. A major part of developing responsible AI is to understand and mitigate this impact.

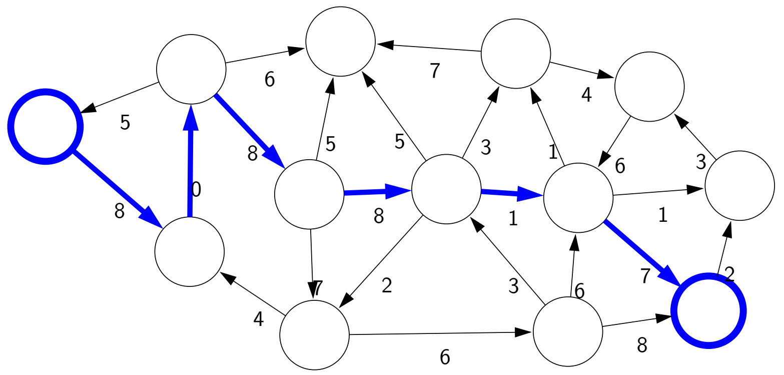
Paradigm: inference

Model



Inference

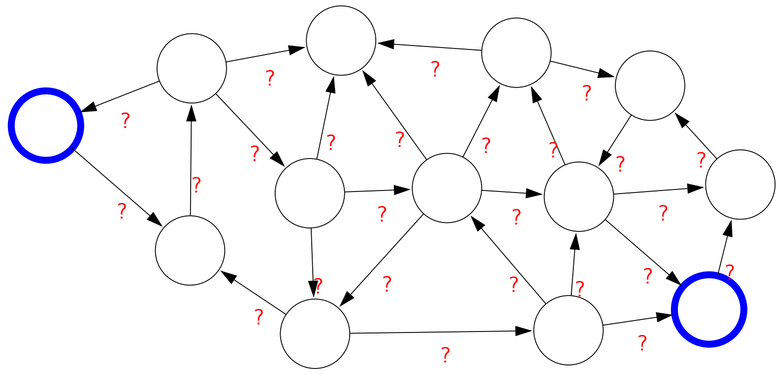
Predictions



- The second pillar is inference. Given a model, the task of **inference** is to answer questions with respect to the model. For example, given the model of the city, one could ask questions such as: what is the shortest path? what is the cheapest path?
- The focus of inference is usually on efficient algorithms that can answer these questions.
- For some models, computational complexity can be a concern (games such as Go), and usually approximations are needed.

Paradigm: learning

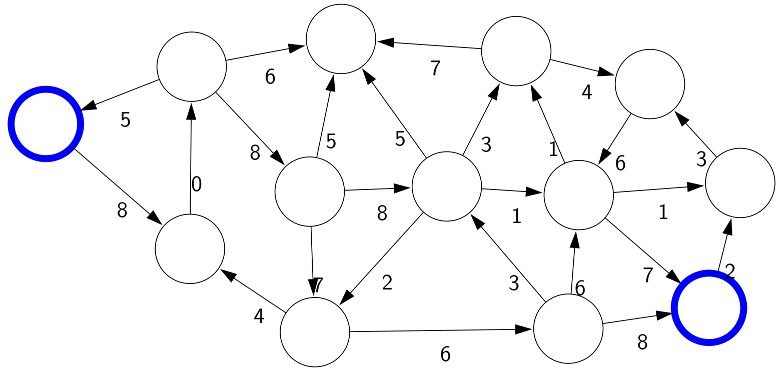
Model without parameters



+data

Learning

Model with parameters



- But where does the model come from? Remember that the real world is rich, so if the model is to be faithful, the model has to be rich as well. But we can't possibly write down such a rich model manually.
- The idea behind (machine) **learning** is to instead get it from data. Instead of constructing a model, one constructs a skeleton of a model (more precisely, a model family), which is a model without parameters. And then if we have the appropriate data, we can run a machine learning algorithm to tune the parameters of the model.
- Many of you are probably only seen models in the context of machine learning (e.g., neural networks), but I want you all to take a very broad view of what a model is. The idea of learning is not tied to a particular model family (e.g., neural networks). Rather, it is more of a philosophy of how to produce models.

Paradigm

Modeling

Inference **Learning**

- To summarize, modeling simplifies the real world, inference answers questions against the model, and learning constructs the model from data.
- Note that each step could be challenging to perform and require approximations. There will likely be tradeoffs too.

Course plan

Machine learning

- Now in this course, we go through different types of models for representing different types of real-world problems.
- We start with models that are low-level (which just take an input and return an output — these are the models that you might be more familiar with in machine learning). Gradually, we progress to high-level models that are based on more logical reasoning.
- Before we start, we talk about machine learning, which can support all models.

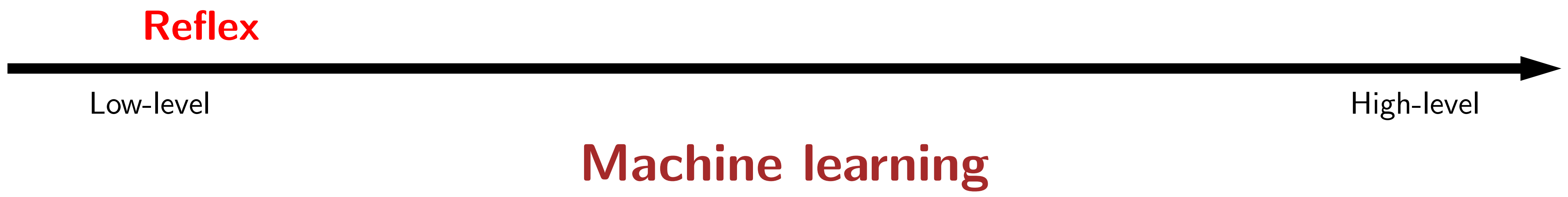
Machine learning

$$\begin{tikzcd}\boxed{\text{Data}} \arrow[r] & \boxed{\text{Model}}\end{tikzcd}$$

- The main driver of recent successes in AI
- Move complexity from ”code” to ”data”
- Requires a leap of faith: **generalization**

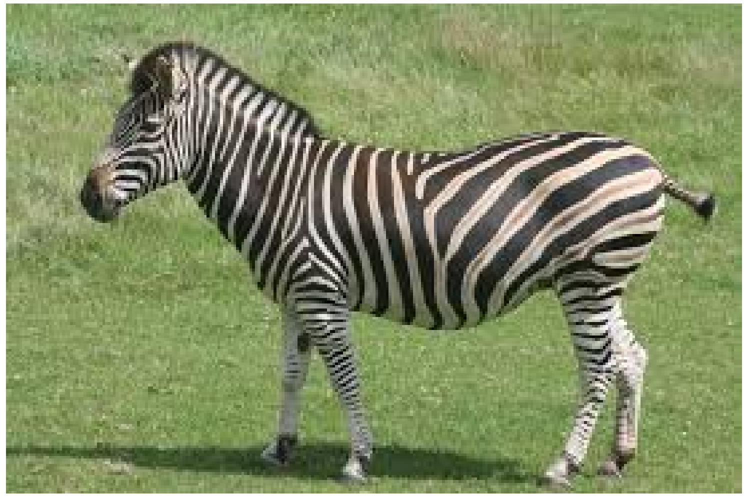
- Supporting all of these models is **machine learning**, which has been arguably the most crucial ingredient powering recent successes in AI. From a practical perspective, machine learning allows us to shift the complexity of the model from code to data; instead of writing a lot of code to describe the model, we instead write a very simple model family (e.g., choice of neural network architecture) and focus on collecting the data to train a model within that family.
- The main conceptually magical part of learning is that if done properly, the trained model will be able to produce good predictions beyond the set of training examples. This leap of faith is called **generalization**, and is, explicitly or implicitly, at the heart of any machine learning algorithm. This can be formalized using tools from probability and statistical learning theory.

Course plan



- We now start our tour of models with reflex models.

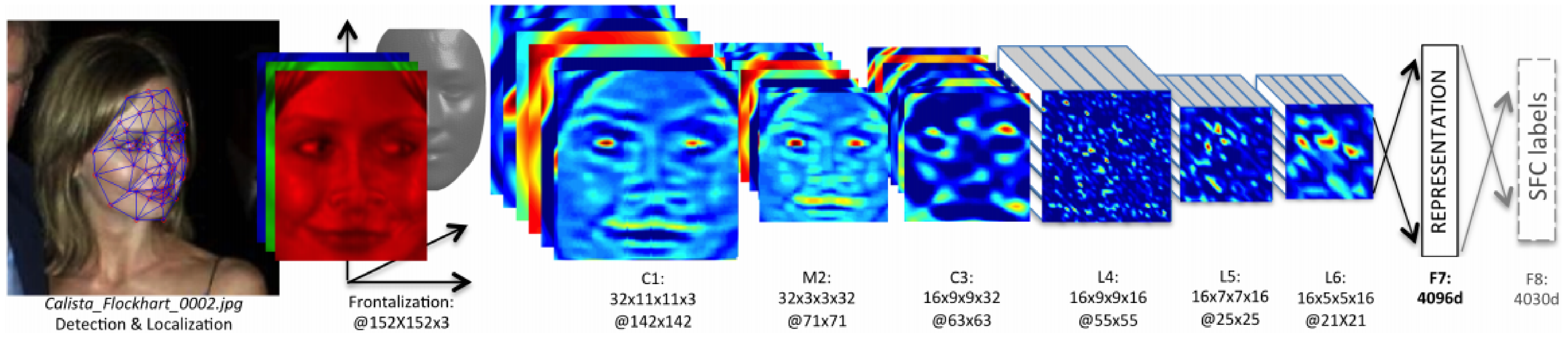
What is this animal?



- Most of you could probably recognize the zebra in that split second.

Reflex-based models

- Examples: linear classifiers, deep neural networks

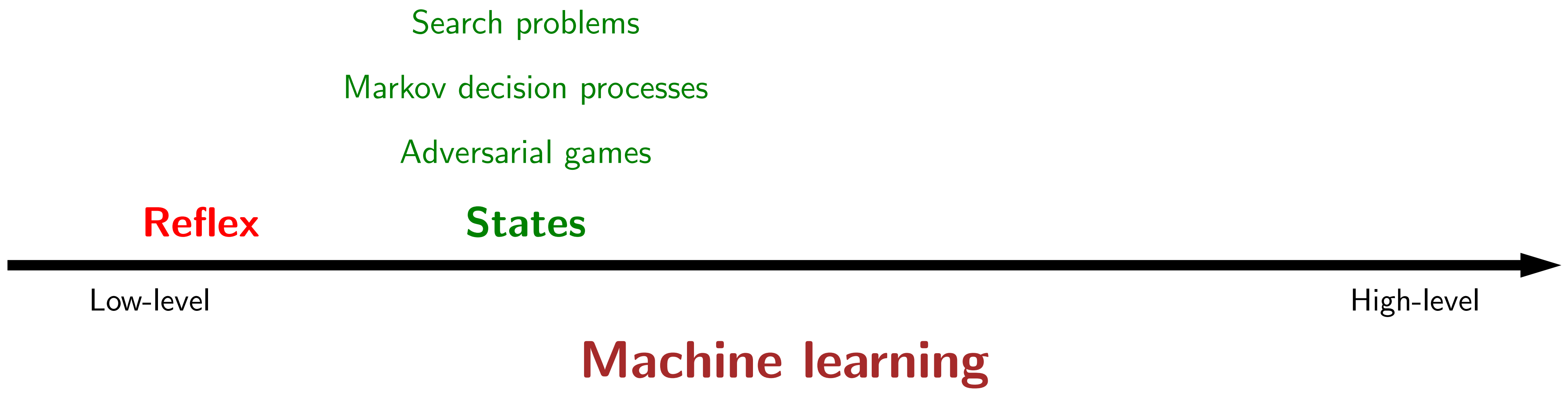


- Most common models in machine learning

- Fully feed-forward (no backtracking)

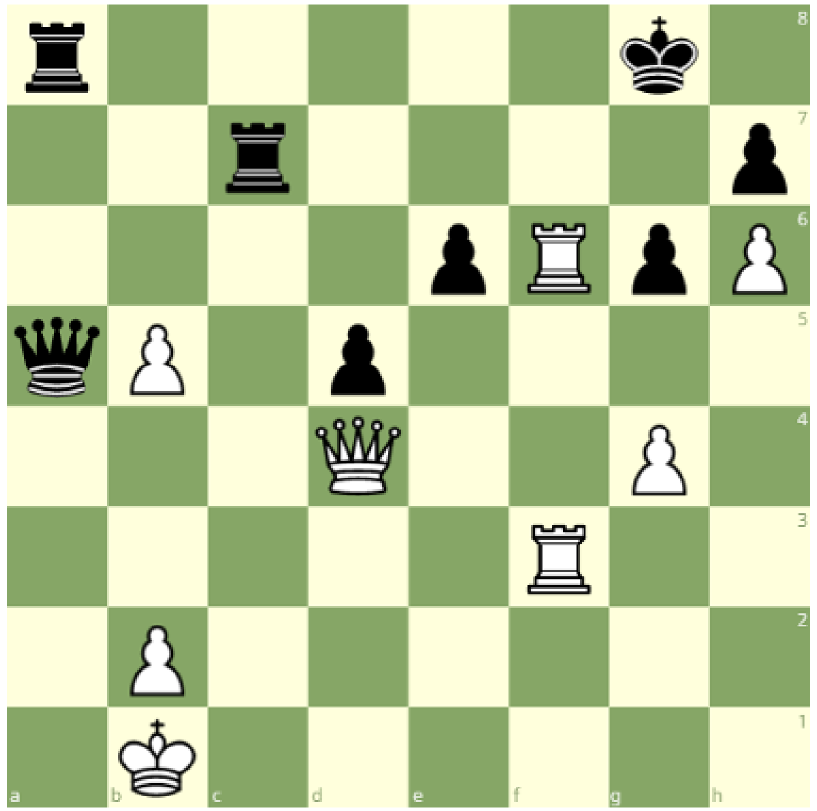
- A reflex-based model simply performs a fixed sequence of computations on a given input. Examples include most models found in machine learning, from simple linear classifiers to deep neural networks.
- The main characteristic of reflex-based models is that their computations are feed-forward; one doesn't backtrack and consider alternative computations.
- Inference is straightforward in these models because it is just running the fixed computations, which makes these models appealing.

Course plan


Search problems
Markov decision processes
Adversarial games
Reflex
States
Low-level
High-level
Machine learning

- Next, we will consider state-based models.

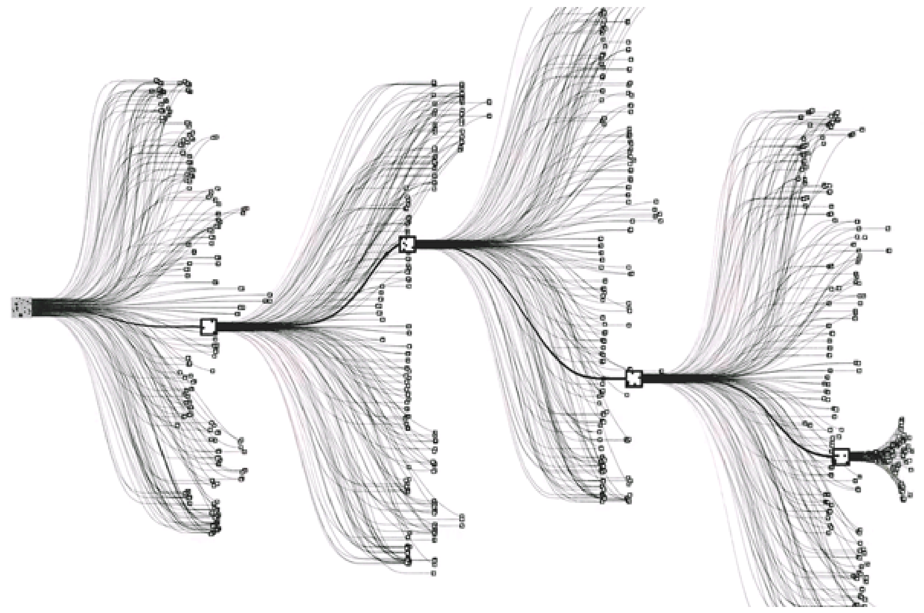
State-based models



White to move

- Consider the task of figuring out what move white should make given a particular chess position.
- Most of us will find this more challenging than recognizing the zebra.

State-based models



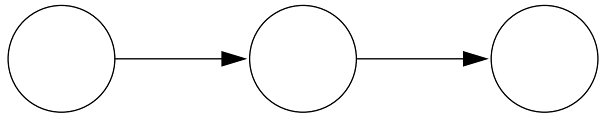
Applications:

- Games: Chess, Go, Pac-Man, Starcraft, etc.
- Robotics: motion planning

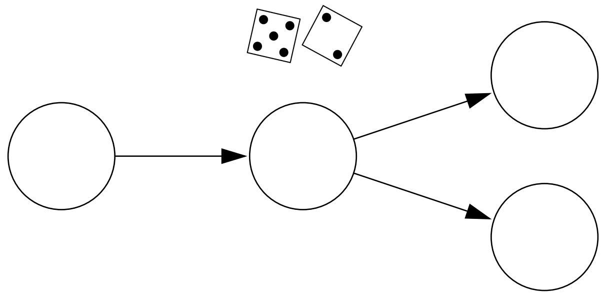
- Reflex-based models are too simple for tasks that require more forethought (e.g., in playing chess or planning a big trip). State-based models overcome this limitation.
- The key idea is, at a high-level, to model the **state** of a world and transitions between states which are triggered by actions. Concretely, one can think of states as nodes in a graph and transitions as edges. This reduction is useful because we understand graphs well and have a lot of efficient algorithms for operating on graphs.

State-based models

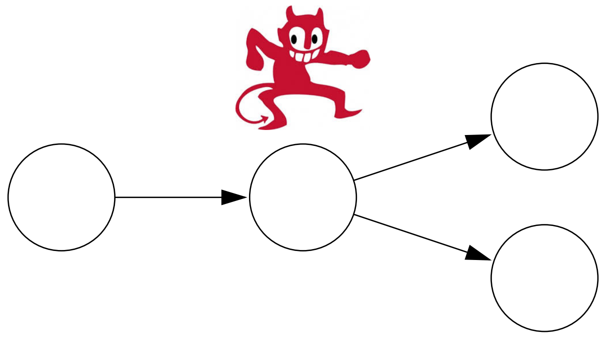
Search problems: you control everything



Markov decision processes: against nature (e.g., Blackjack)

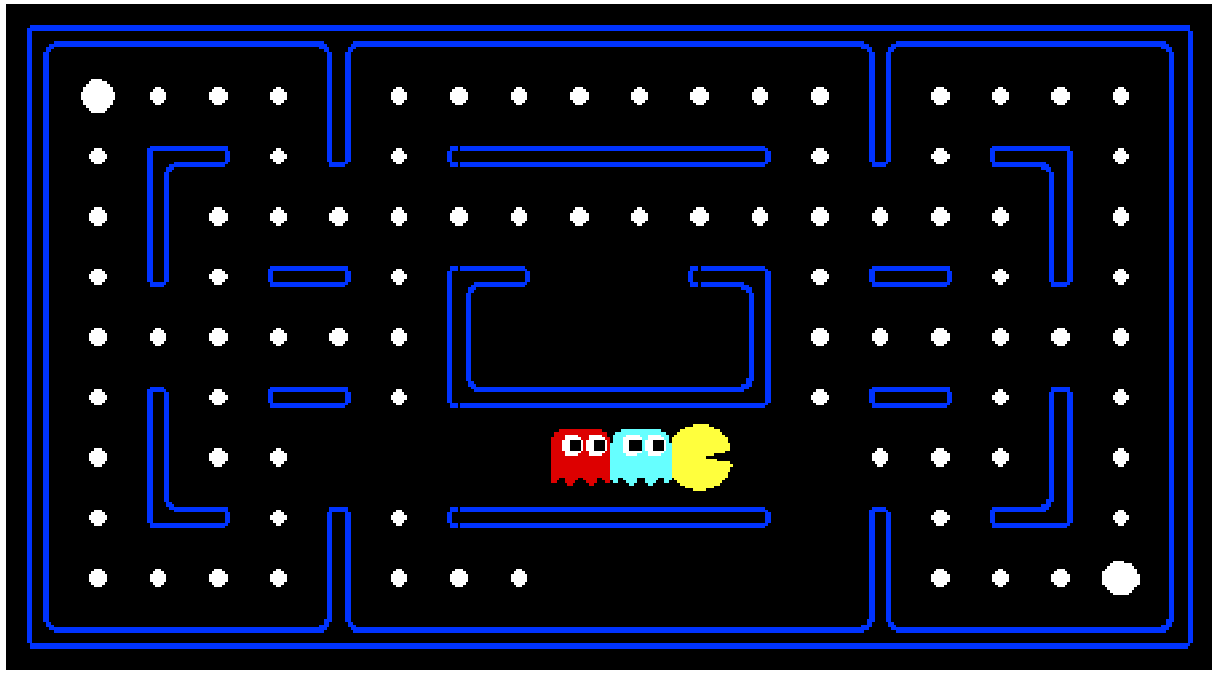


Adversarial games: against opponent (e.g., chess)



- We consider three types of state-based models.
- **Search problems** are models where you are operating in an environment that has no uncertainty. However, in many realistic settings, there are other forces at play.
- **Markov decision processes** handle situations where there is randomness (e.g., Blackjack).
- **Adversarial games**, as the name suggests, handle tasks where there is an opponent who is working against you (e.g., chess).

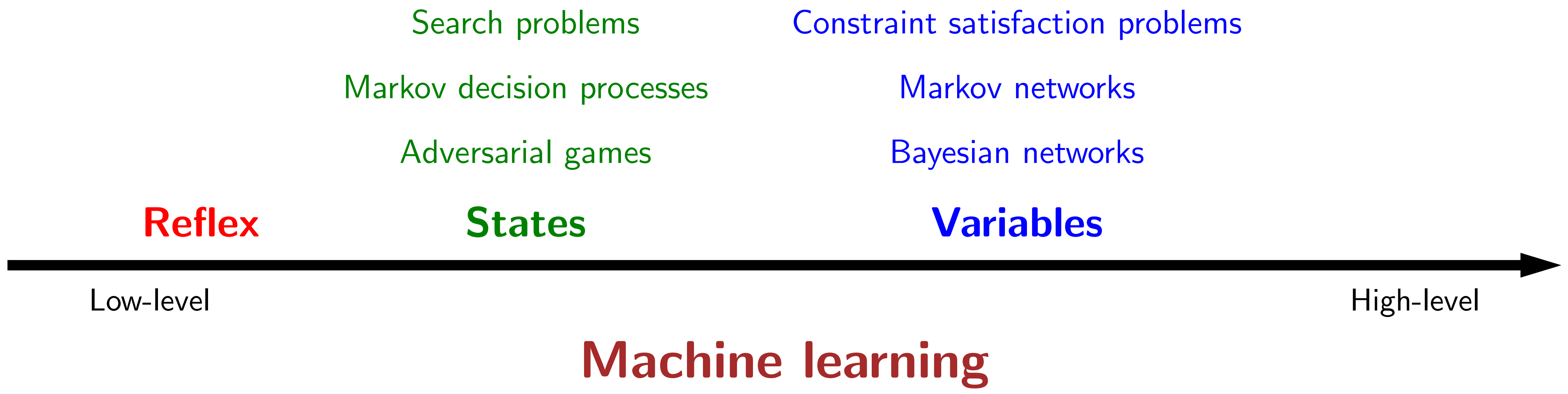
Pac-Man



[demo]

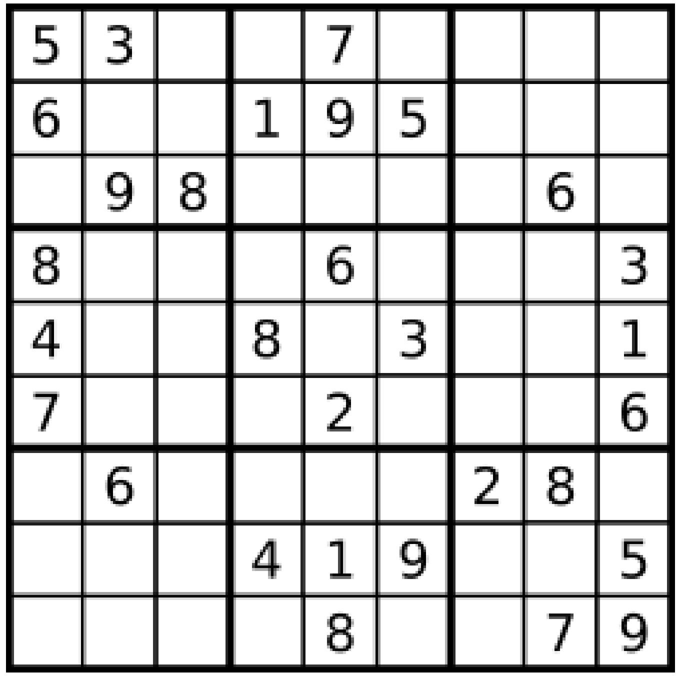
- In one of the homeworks, you will build an agent that plays Pac-Man.
- To whet your appetite, this is what it will look like.
- (demo)
- Think about what the states of this model should be, and how you might come up with the optimal strategy.

Course plan


Search problems
Markov decision processes
Adversarial games
Constraint satisfaction problems
Markov networks
Bayesian networks
Reflex
States
Variables
Low-level
High-level
Machine learning

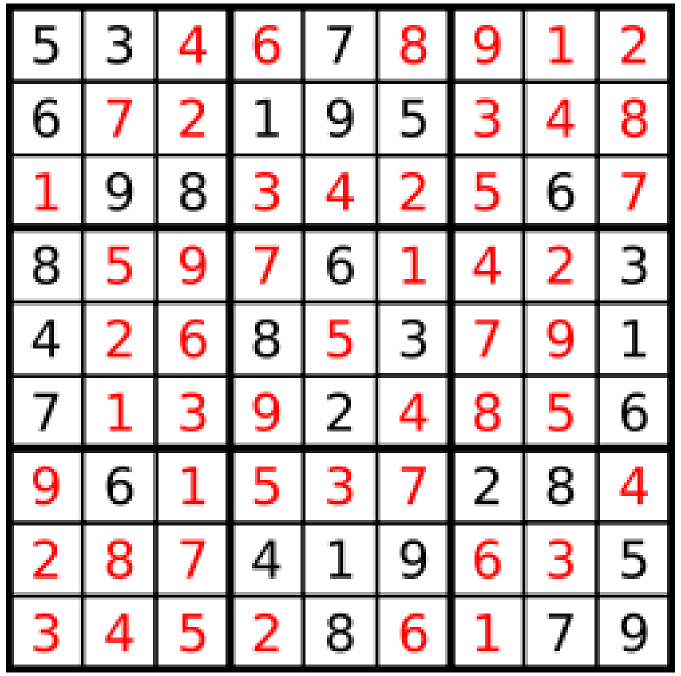
- Next, we will talk about variable-based models.

Sudoku



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

$\rightarrow$



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

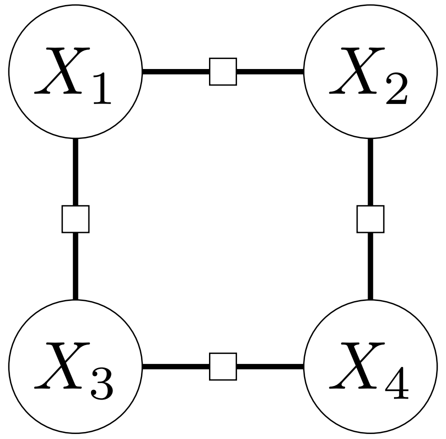
Goal: put digits in blank squares so each row, column, and 3x3 sub-block has digits 1–9

Key: order of filling squares doesn't matter in the evaluation criteria!

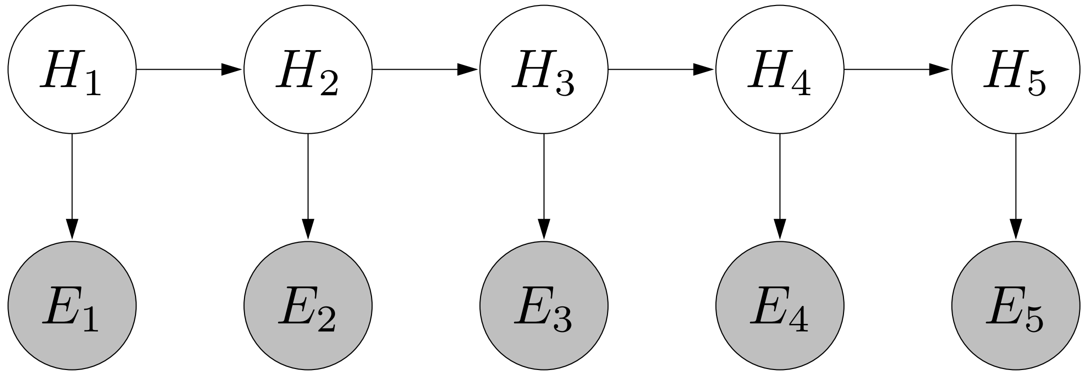
- In state-based models, solutions are procedural: they specify step by step instructions on how to go from A to B.
- In some applications, the order in which things are done isn't important.
- For example, in Sodoku, where the goal is to put digits in the blank squares to satisfy some constraints, all that matters is the final configuration of numbers; you can fill them in in any order.
- Variable-based models allow you to declare you want (it's like a higher-level language) rather than micro-manage how you want the solution to be found.

Variable-based models

Constraint satisfaction problems: hard constraints (e.g., Sudoku, scheduling)



Bayesian networks: soft dependencies (e.g., tracking cars from sensors)



- **Constraint satisfaction problems** are variable-based models where we only have hard constraints. For example, in scheduling, one person can't be in two places at once.
- **Bayesian networks** are variable-based models where variables are random variables which are dependent on each other. For example, the true location of an airplane H_t and its radar reading E_t are related, as are the location H_t and the location at the last time step H_{t-1}. The exact dependency structure is given by the graph and it formally defines a joint probability distribution over all of the variables.

Course plan

Search problems

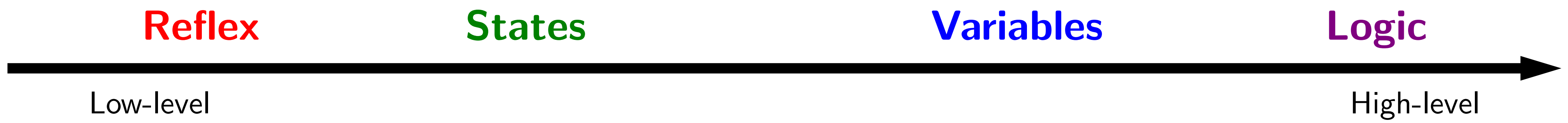
Markov decision processes

Adversarial games

Constraint satisfaction problems

Markov networks

Bayesian networks



Machine learning

- The last topic is logic.

Motivation: virtual assistant

Tell information → ← **Ask** questions

Use natural language!

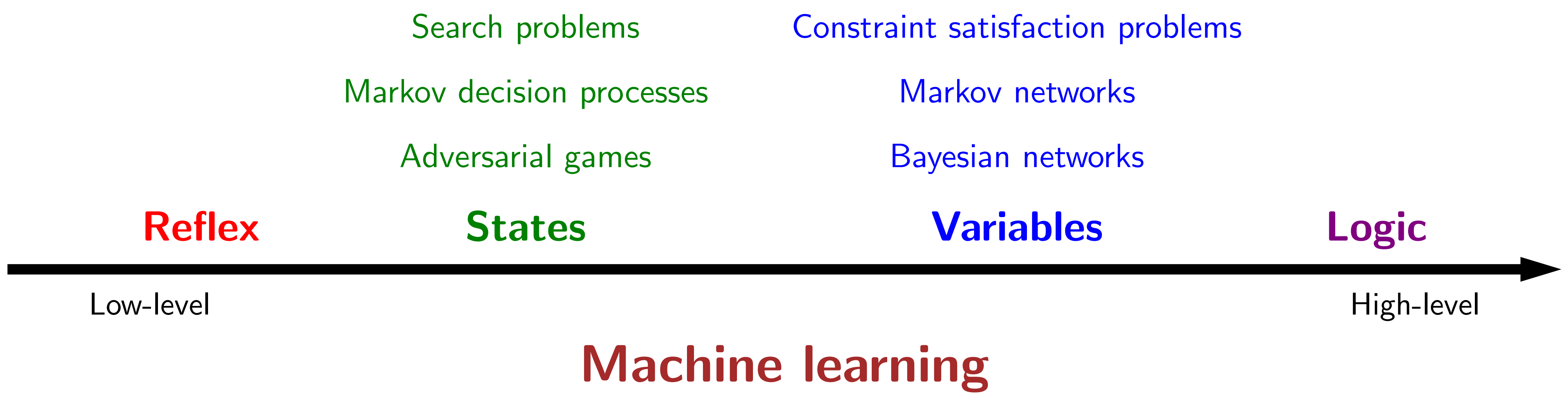
[demo]

Need to:

- Digest **heterogenous** information
- Reason **deeply** with that information

- One motivation for logic is a virtual assistant. A good assistant should be able to remember what you told it and answer questions that require drawing inferences from its knowledge. And you'd want to interact with it using natural language, the tool that humans invented for communication.
- I'll show you a demo which you'll have an opportunity to play with in the final homework.
- (demo)
- Interacting with this system feels very different than a typical machine learning-based system. First, it is adaptive, whereas most ML systems are a fixed function. Second, the types of information we provide it and the types of questions are more heterogenous and more abstract, and we expect the system to realize the full consequences of every single word. (We don't want to tell our personal assistant 100 times that we prefer morning meetings.)
- Recently, with the emergence of large language models like ChatGPT, we are beginning to see virtual assistants like the one demoed here, but with far greater capabilities. However, LLMs are well-known for their hallucinations, whereas the logic-based system is 100% internally consistent.
- One often contrasts logical AI and statistical AI. In this course, we will treat the two as not contradictory but rather complementary. Logic provides a class of models which is higher-level, but still needs to be supported by the groundedness to real data that machine learning offers.

Course plan


Search problems
Markov decision processes
Adversarial games
Constraint satisfaction problems
Markov networks
Bayesian networks
Reflex
States
Variables
Logic
Low-level
High-level
Machine learning

- And this concludes our tour of the topics.
- To summarize, we will discuss models, going from reflex to state-based models to variable-based models to logic. For each, we will instantiate the modeling-inference-learning paradigm.