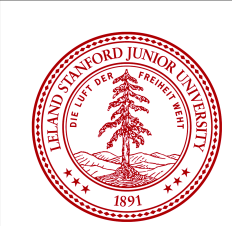


Lecture 1: Overview

Teaching staff

Percy Liang (instructor)

Panupong (Ice) Pasupat (head CA)

Arijit Banerjee

Greg Bodwin

Diego Canales

Adam Goldberg

Ilan Goodman

Will Harvey

Jiaji Hu

Billy Jun

Aparna Krishnan

Janice Lan

Amrit Saxena

Jiawei Yao

Roadmap

Why learn AI?

What topics will you learn?

How will you learn it?

Optimization

~~*What is AI?*~~

What can AI do for you?

- We could have begun this course by giving a definition of AI. However, I think it's more inspiring to start by looking at the impact AI has already had on society.

cs221.stanford.edu/q

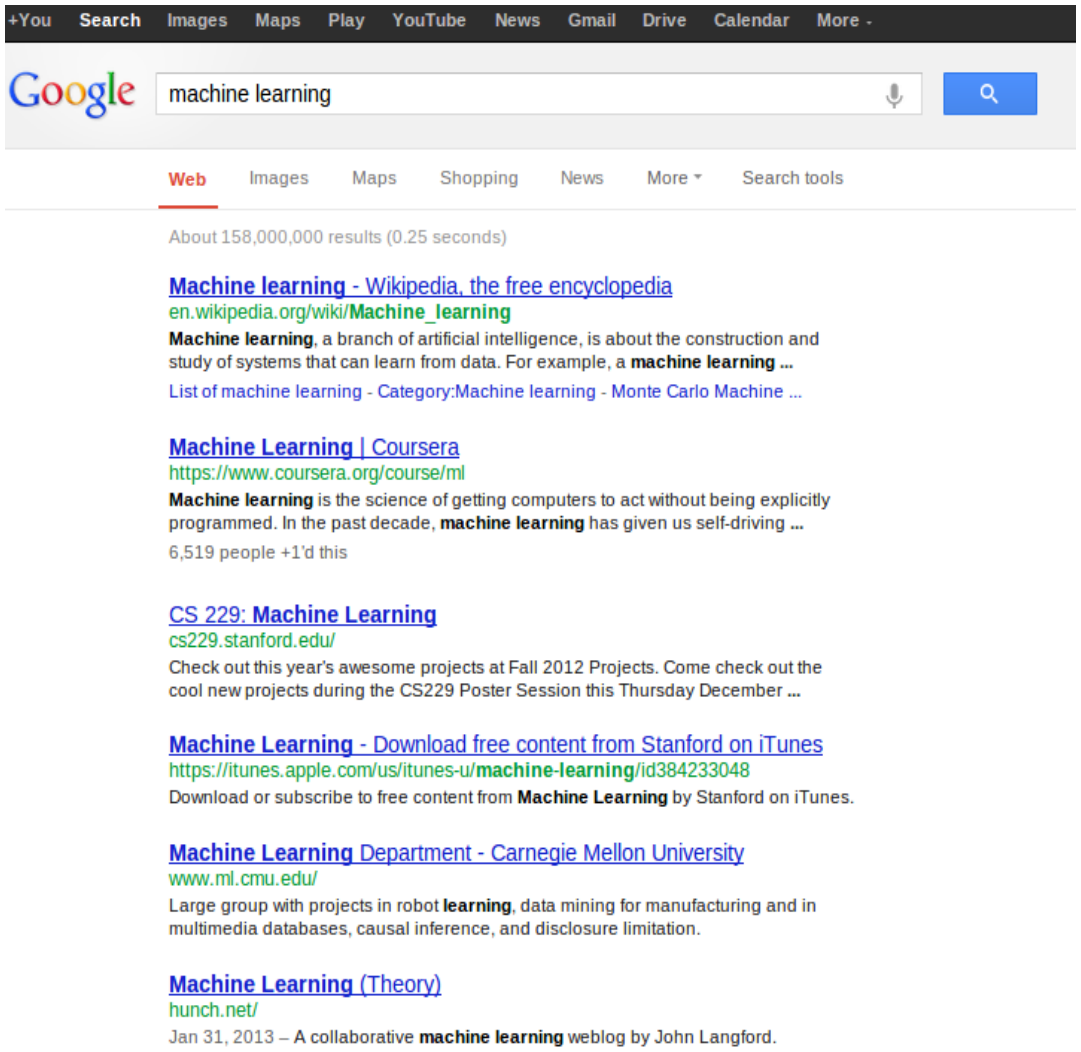
Question

Which do you think would be hardest for an AI to do today?

- translating an article from Chinese to English
- identifying all the chairs in an image
- transcribing a conversation at a party
- folding your laundry
- proving new theorems
- automatically replying to your email

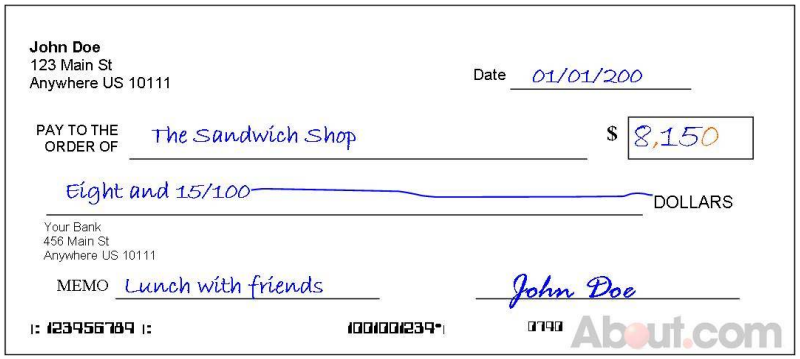
- First, systems have been built for all of the above problems, and some of them (e.g., machine translation) are widely used. But all are still far from perfect, and they tend to break down when the input and environment are noisy and unstructured. All are very fertile areas of research and there is a lot to be done.
- Second, although the problems seem quite diverse, we will see in this class that many of the same techniques and principles can be leveraged. In a way, the promise of AI is its generality, a small set of core tools that have effect on a far-reaching set of problems.

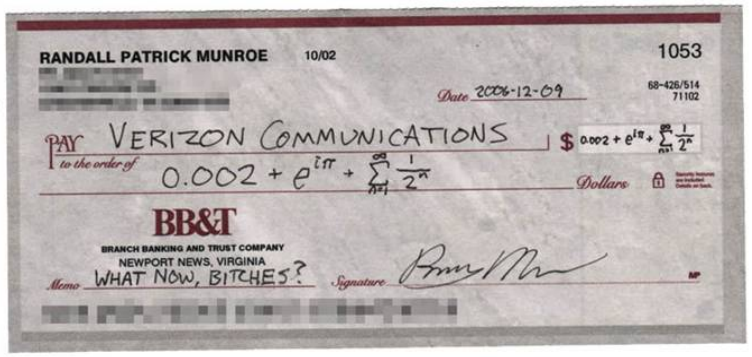
Web page ranking



- When you search the web, ranking algorithms based on machine learning are employed to choose the relevant web pages to show you.

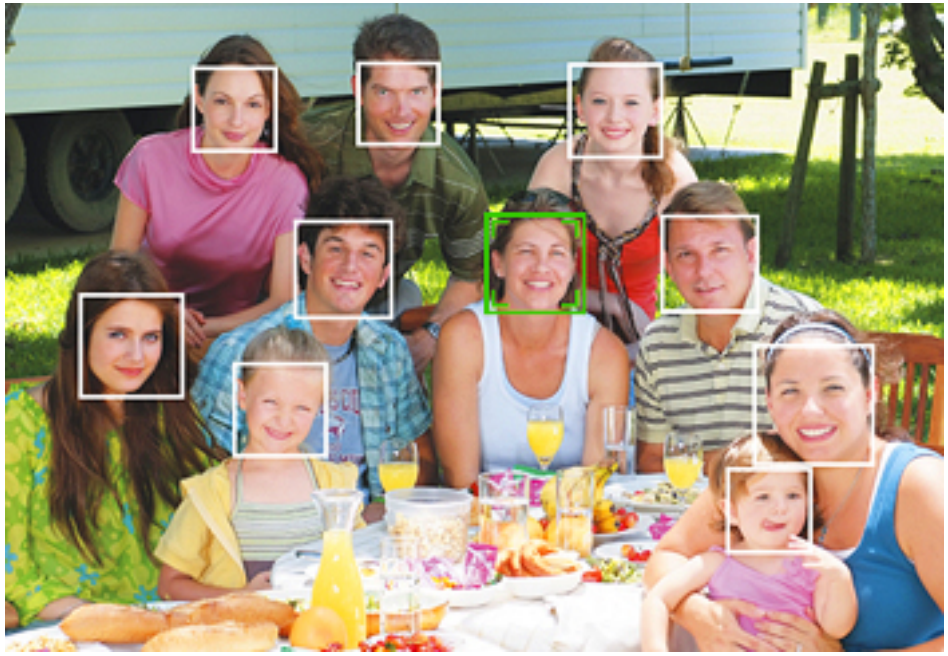
Handwriting recognition





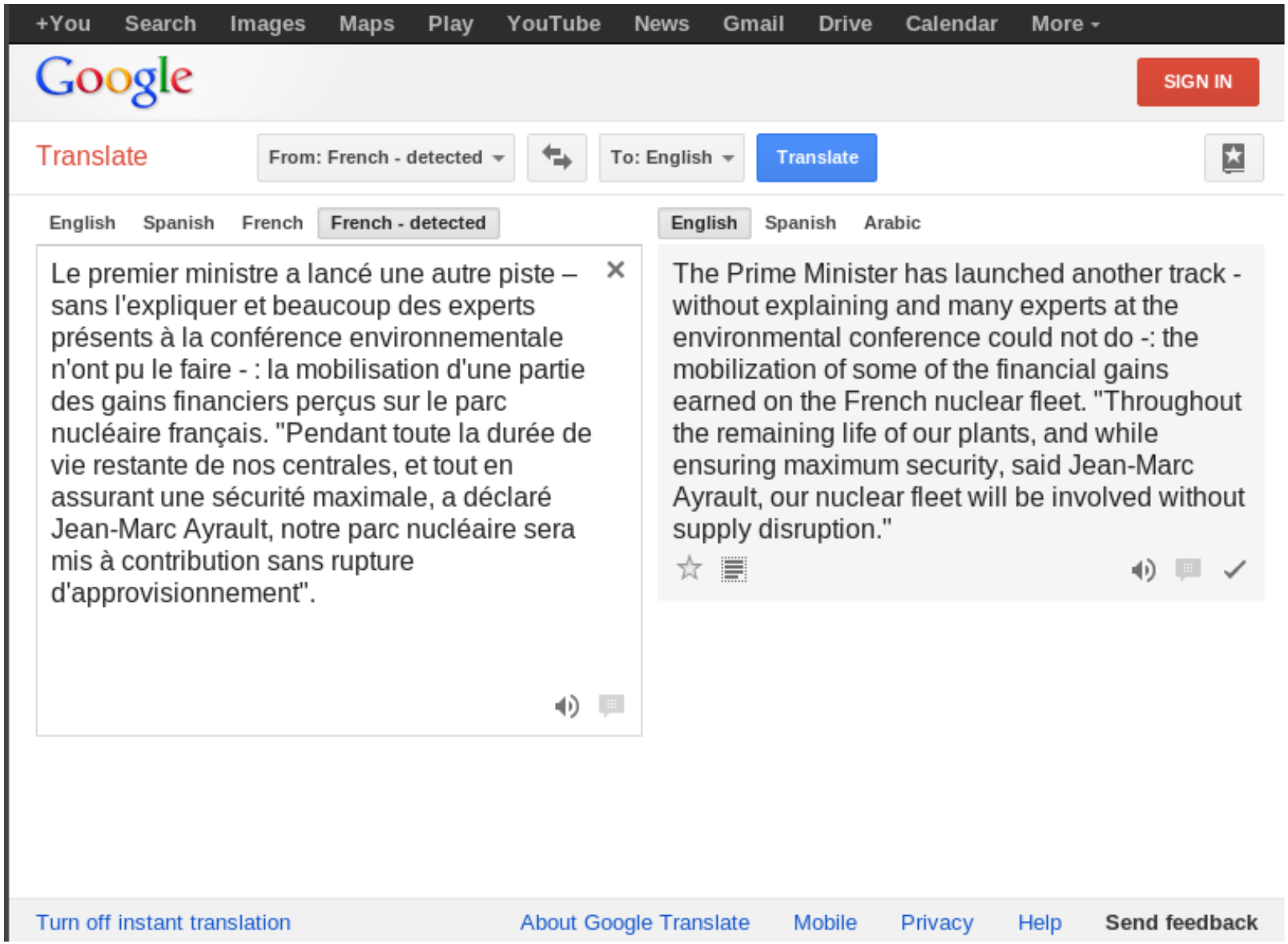
- When you deposit a check at an ATM, handwriting recognition is employed to automatically figure out the deposit amount.

Face detection



- When you take a picture, face detection is employed to identify faces and perform auto-focus or auto-tagging.

Machine translation



- If you want to read a news article in another language, you can turn to machine translation.

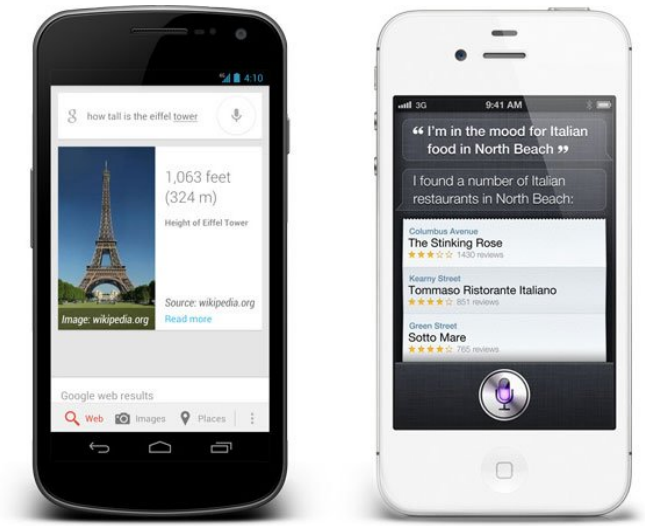
Route planning

- AI also plays an important infrastructural role in applications which one might not immediately think of as AI, but more operations.
- Package delivery companies use search, planning and optimization techniques to deliver millions of packages using thousands of trucks and planes.

Self-driving cars

- In the next decade, we will likely interface with AI in many more ways.
- Already, self-driving cars are hitting the road and have driven thousands of miles autonomously.

Virtual assistants



- With the rise of mobile devices, smart cars and homes, and improvements in speech recognition, we will be able to interact with computers using natural language and gestures. Imagine coming home and saying: "what do I need to buy for tomorrow's picnic and where can I do that now?"
- Currently, Apple's Siri, Google Now, and Microsoft Cortana provide a first stab at this problem, handling mostly simple utterances and actions (e.g., setting an alarm, sending a text, etc.) The technology is still in its infancy, but it is an exciting and a rapidly moving field.

Physical assistants

- There are many more applications of AI if you include the physical world. Here is an example of a towel-folding robot from Pieter Abbeel's group at Berkeley.
- Folding towels is not easy for robots, since towels are fairly unstructured objects: they could be in any location and pose. It turns out that the hardest part of folding a towel is detecting this pose (done by finding corners). Not exactly ready to deploy in people's homes, but a step in the right direction.

Many more applications...

...

Web search

Speech recognition

Handwriting recognition

Machine translation

Information extraction

Document summarization

Question answering

Spelling correction

Image recognition

3D scene reconstruction

Human activity recognition

Autonomous driving

Music information retrieval

Automatic composition

Social network analysis

...

...

Product recommendation

Advertisement placement

Smart-grid energy optimization

Household robotics

Robotic surgery

Robot exploration

Spam filtering

Fraud detection

Fault diagnostics

AI for video games

Character animation

Financial trading

Protein folding

Medical diagnosis

Medical imaging

...

Characteristics of AI

High societal impact (affect billions of people)

Diverse (language, vision, robotics)

Complex (really hard)

- What's in common with all of these examples?
- It's clear that AI applications tend to be very **high impact**.
- They are also incredibly **diverse**, operating in very different domains, and requiring integration with many different modalities (natural language, vision, robotics). Throughout the course, we will see how we can start to tame this diversity with a few fundamental principles and techniques.
- Finally, these applications are also mind-bogglingly **complex** to the point where we shouldn't expect to find solutions that solve these problems perfectly.

Two sources of complexity...

Machine translation: number of possible translations?

Input:

C'est ne pas une pipe.

Output:

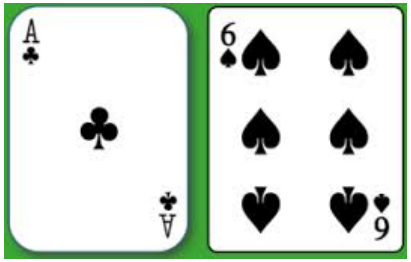
□	□	□	□	□	□
a	a	a	a	a	a
aardvark	aardvark	aardvark	aardvark	aardvark	aardvark
aback	aback	aback	aback	aback	aback
...	...	...	...	...	...

$$(\text{vocabulary size})^{(\text{sentence length})} = 10000^{20}$$

Computational complexity: exponential explosion

- There are two sources of complexity in AI tasks.
- The first, which you, as computer scientists, should be familiar with, is **computational complexity**. We can solve useful problems in polynomial time, but most interesting AI problems — certainly the ones we looked at — are NP-hard. We will be constantly straddling the boundary between polynomial time and exponential time, or in many cases, going from exponential time with a bad exponent to exponential time with a less bad exponent.
- Just as a simple example, in machine translation, we are given an input sentence (say, in Chinese) and need to output an translation (say, in English). Suppose our English vocabulary has size 10000 and we are considering English translations with 20 words. Then the total number of translations is $10000^{20} = 10^{80}$, which is completely ridiculous. One can be more clever and use the input sentence to prune down the number of words from 10000 to 10, but 10^{20} is still quite absurdly large.

这是什么意思?



Even infinite computation isn't enough...need to somehow *know* stuff.

Information complexity: uncertainty is pervasive

- The second source of complexity, which you might not have thought of consciously, is **information complexity**.
- (Note that there are formal ways to characterize information based on Shannon entropy, but we are using the term information rather loosely here.) Suppose I gave you (really, your program) literally infinite computational resources, locked you (or your program) in a room, and asked you to translate a sentence. Or asked you to classify an image with the type of bird (it's a Weka from New Zealand, in case you're wondering). Or if you're playing Blackjack and drew an ace and a 6, and have do say whether to hit or stay.
- In each of these cases, increasing the amount of computation past a certain point simply won't help. In these problems, we simply need the information or knowledge about a foreign language, ornithology, or about what's in the dealer's deck to make optimal decisions. But just like computation, we will be always information-limited and therefore have to simply cope with **uncertainty**.

Resources

Computation (time/memory)

Information (data)

- We can switch vantage points and think about resources to tackle the computational and information complexities.
- In terms of computation, **computers** (fast CPUs, GPUs, lots of memory, storage, network bandwidth) are a resource. In terms of information, **data** is a resource.
- Fortunately, for AI, in the last two decades, the amount of computing power and data has skyrocketed, and this trend coincides with our ability to solve some of the challenging tasks that we discussed earlier.

How do we ~~solve~~ tackle these challenging problems?

How?

Real-world task

- So having stated the motivation for working on AI and the challenges, how should we actually make progress?
- Given a complex real-world task, 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 real-world task and code.

Paradigm

Real-world task 

Modeling

Formal task (model) 

Algorithms

Program 

- A useful paradigm for solving complex tasks is to break them up into two stages. The first stage is modeling, whereby messy real-world tasks are converted into clean formal tasks called **models**. The second stage is algorithms, where we find efficient ways to solve these formal tasks.

Algorithms (example)

Formal task:

- Input: list $L = [x_1, \ldots, x_n]$ and a function $f : X \mapsto \mathbb{R}$
- Output: k highest-scoring elements

Example: $k = 2$

L	A	B	C	D
f	(3)	2	(7)	1

Two algorithms:

- Take the largest, remove it, take the second largest, ...
- Sort L based on f, then take first k elements

- Let's start with something that you're probably familiar with: algorithms. When you study algorithms, you are generally given a well-defined formal task, something specified with mathematical precision, and your goal is to solve the task. A solution either solves the formal task or it doesn't, and in general, there are many possible solutions with different computational trade-offs.
- As an example, suppose you wanted to find the k largest elements in a list of $L = [x_1, \ldots, x_n]$ according to given a scoring function f that maps each element into a real-valued score.
- Solving a formal task involves coming up with increasingly more efficient algorithms for solving the task.

Modeling (example)

Real-world task:

- Input: list of 1000 web pages
- Output: 10 most relevant web pages

Modeling

$L =$ set of web pages

$f(x) = 10 \cdot \text{QueryMatch}(x) + 3 \cdot \text{PageRank}(x)$

Formal task:

- Input: list $L = [x_1, \ldots, x_n]$ and a function $f : X \mapsto \mathbb{R}$
- Output: k highest-scoring elements

- However, real-world tasks are not well-defined. For example, the web ranking task is to output the 10 most pages relevant to a user's query. What does "relevant" mean?
- Our strategy is not to develop algorithms for solving real-world tasks directly, but rather to convert them via a process called **modeling** into formal tasks.
- In our example, we would have to decide on the appropriate scoring function $f(x)$, which could be the number of words in x that are also in the user's query, the PageRank of x which measures how popular x is, or some combination of the two.
- The advantage of modeling is that now we can entertain increasingly sophisticated scoring functions f without bothering about the details of how we're going to compute it.

Modeling and algorithms

- Separate **what** to compute (**modeling**) from **how** to compute it (**algorithms**)

- Advantage: divison of labor

- This class: providing a toolbox of different classes of models

- This modularity between modeling and algorithms is a powerful abstraction that is the basis for successful methods, not only in AI but in many disciplines.
- The advantage is a division of labor: we can think about the real-world task at a higher-level of abstraction (via the model), while people who are good at coming up with clever algorithms have some formal specification to latch on to.
- The purpose of this class is to introduce you to a set of different model types which are useful in a variety of different settings. We will both develop efficient algorithms for these formal tasks, but more importantly, practice the art of modeling: converting real-world tasks into models.

Summary so far

- Applications of AI: high-impact, diverse
- Challenges: computational/information complexity
- Paradigm: modeling + algorithms

Roadmap

Why learn AI?

What topics will you learn?

How will you learn it?

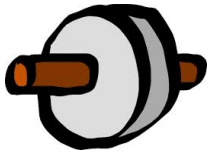
Optimization

Course plan

"Low-level intelligence" "High-level intelligence"

Machine learning

- We now embark on our tour of the topics in this course. The topics correspond to classes of models that we can use to represent real-world tasks. The topics will in a way advance from low-level intelligence to high-level intelligence, evolving from models that simply make a reflex decision to models that are based on logical reasoning.

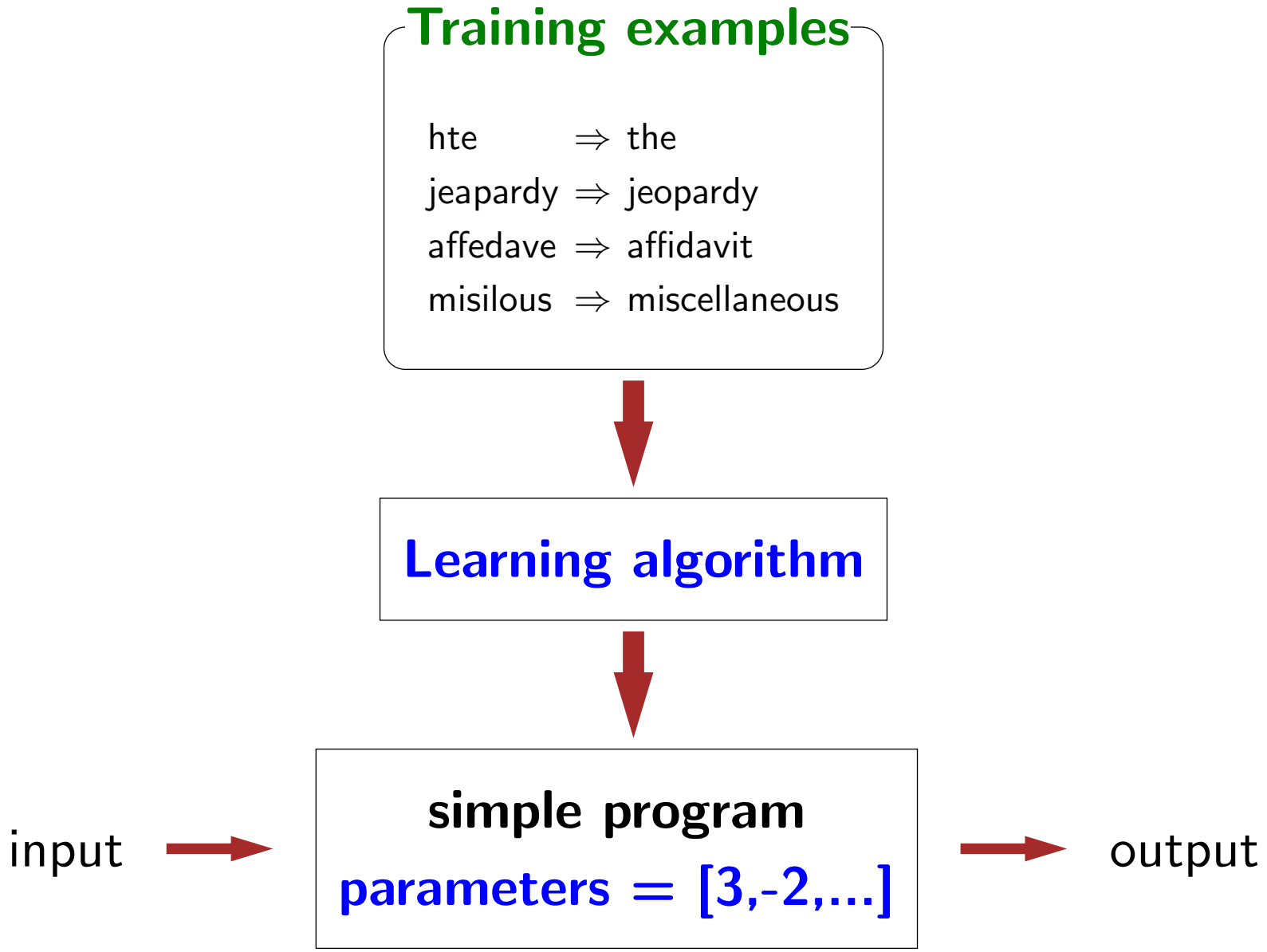


Traditional approach

A spell checker:

Problem: complexity becomes unwieldy

Machine learning approach



- Supporting all of these models is **machine learning**. Abstractly, machine learning allows systems to adapt and improve according to their environment, rather than be built once and remain static. Pragmatically, the idea is to write a simple program with a few knobs (**parameters**) which can be tuned.
- We start with a set of training examples that partially specify the desired system behavior. We then construct a learning algorithm that takes the training examples and sets the parameters of our simple program so that it approximately produces the desired system behavior.

Machine learning

Key idea: generalization

Learning algorithm maximizes accuracy on **training** examples.

But we only care about accuracy on future **test** examples.

How to **generalize** from training to test?

- The main conceptually magical part of learning is that if done properly, the trained program will be able to produce good answers 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 even be formalized using tools from probability and statistical learning theory.

Course plan


Reflex
"Low-level intelligence"
"High-level intelligence"
Machine learning

cs221.stanford.edu/q

Question

Movie review: "Shows moments of promise but ultimately succumbs to cliches and pat storytelling." What's the sentiment?

positive

negative

- To demonstrate reflex-models, let us take the example application of sentiment analysis. For concreteness, consider the real-world task of determining whether a movie review expresses a positive or negative sentiment. Sentiment analysis (opinion mining) has been quite popular in recent years. For example, companies are quite interested in mining Twitter and other social media to find out what people think of their products.

Reflex-based models

Input: x

Output: $f(x)$, a simple function of x

Example: f is set of simple rules

If x contains "cliches", return NEGATIVE.

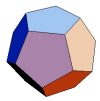
If x contains "promise", return POSITIVE.

...

- Now we start with the simplest class of models, which are reflex-based models. The idea of a reflex-based model is to take an input x and perform a very simple calculation based on x to produce some output $f(x)$. For example, in sentiment classification, x is a review and $f(x)$ is the prediction of whether the review is postive or negative.
- Reflex-based models could consist of a small set of deterministic rules that look at various superficial properties of the input, e.g., what words it contains.

Reflex-based models

Use scores to capture nuances...



Example: f is based on scores

Set score $= 0$.

If x contains "cliches", score -$= 10$.

If x contains "promise" , score $+= 5$.

If score > 0, return POSITIVE.

More generally...

Key idea: linear classifier

$$f(x) = \text{sign}(w_1\phi_1(x) + \cdots + w_d\phi_d(x))$$

- Fancier reflex-based models are based on features and weights (think soft rules): If x has a **feature** (e.g., x contains "cliches"), then the score is updated with the feature's weight (e.g., -10).
- Abstractly, we can think of having d features $\phi_1(x), \ldots, \phi_d(x)$, each of which capture some property of x (in general, $\phi_j(x) \in \mathbb{R}$, but for intuition, you can think of them as 0 or 1). Each feature $\phi_j(x)$ is associated with a weight $w_j \in \mathbb{R}$ which represents how much support $\phi_j(x)$ provides to either a positive or negative classification.
- The final prediction $f(x) = \text{sign}(\sum_{j=1}^{d} w_j \phi_j(x))$ is based on taking the weighted sum over all the features to get a score. If the score is positive, then we output POSITIVE; otherwise, we output NEGATIVE.
- Where do the weights come from? We can use machine learning to set them automatically from data, as we'll see later.

Sentiment detection

[demo]

Course plan

Search problems

Markov decision processes

Adversarial games

Reflex **States**

"Low-level intelligence" "High-level intelligence"

Machine learning

Text reconstruction

Chinese is written without spaces:

这 是 什么 意思 ?

Arabic omits (some) vowels:

مَكْتَبَة

Add vowels and spaces to reconstruct an English phrase:

rtfclntllgnc

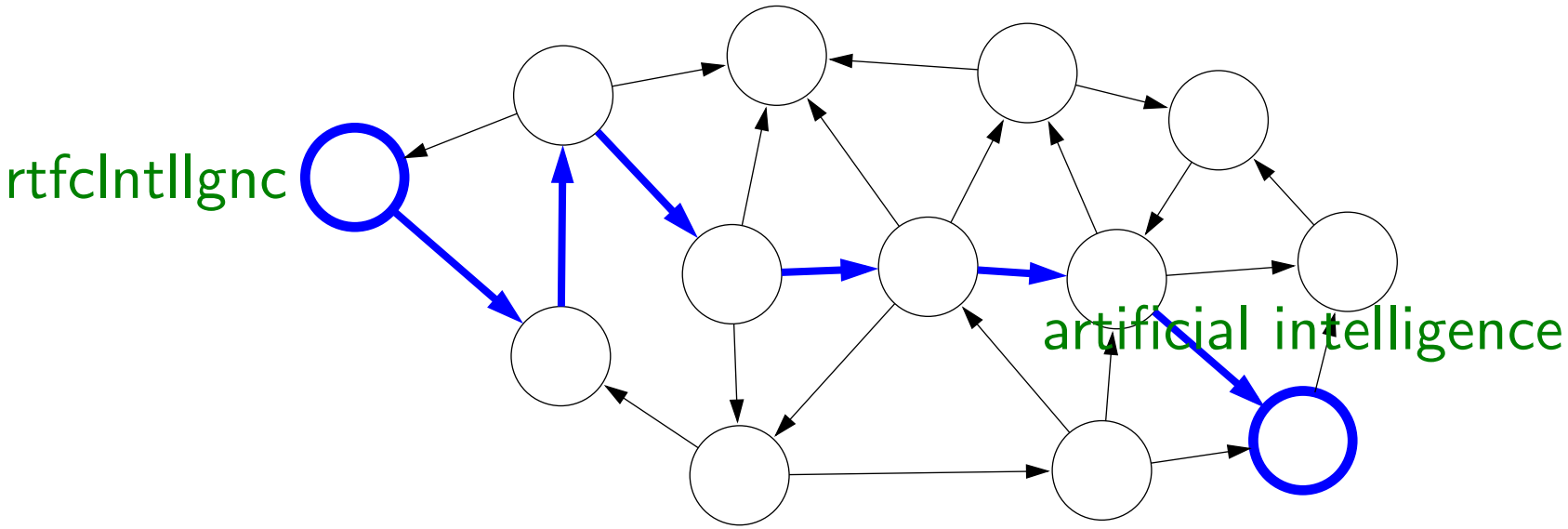
- Reflex-based models are normally used for classification tasks where the output is one of a small set (think multiple choice). However, in many problems (e.g., speech recognition, machine translation), the output is more complicated (e.g., an entire sentence).
- Consider a simpler version of text reconstruction, where we are given some incomplete form of a sentence (such as missing vowels/spaces), and the goal is to recover them. In this class, you will build a system that does this.

Text reconstruction

[demo]

State-based models

Solutions are represented as paths through a graph



- Other examples where reflex-based models are too simple are tasks that require more forethought (i.e., thinking) such as playing chess or planning a big trip. State-based models are one class of models which provide an answer.
- The key idea is, at a high-level, to model real-world tasks as finding (minimum cost) paths through graphs. This reduction is useful because we understand graphs well and have a lot of efficient algorithms for operating on graphs.

State-based models

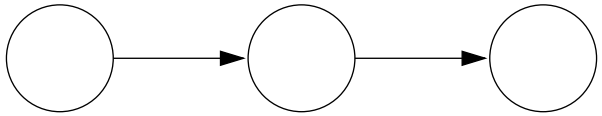
Key idea: state

A **state** captures all the relevant information about the past in order to act optimally in the future.

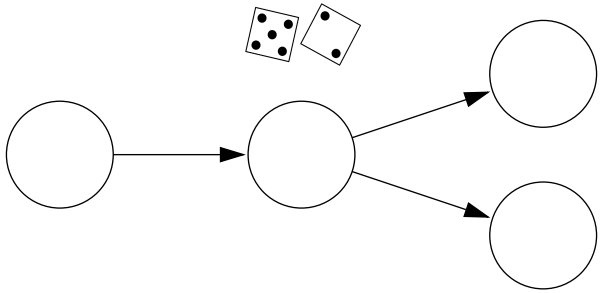
- The simplest case is a **search problem**, where the solution to the original task is a path through the graph from a start to a goal.
- The central concept in using state-based models are **states**, which correspond to the nodes in the graph. Intuitively, a state must contain all the relevant information about the past needed is needed to make optimal choices in the future.
- Consider the task of finding the cheapest way to travel from city A to city B via a sequence of intermediate cities. In this case, the state should contain the current city, but perhaps also the amount of gas or the time of day, depending on the task. We will discuss these design issues in much more detail later in the course.

State-based models

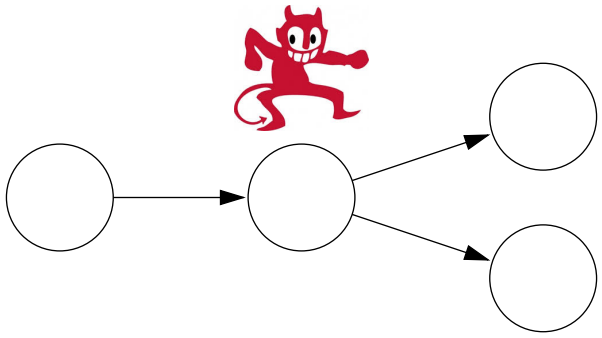
Search problems: you control everything



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

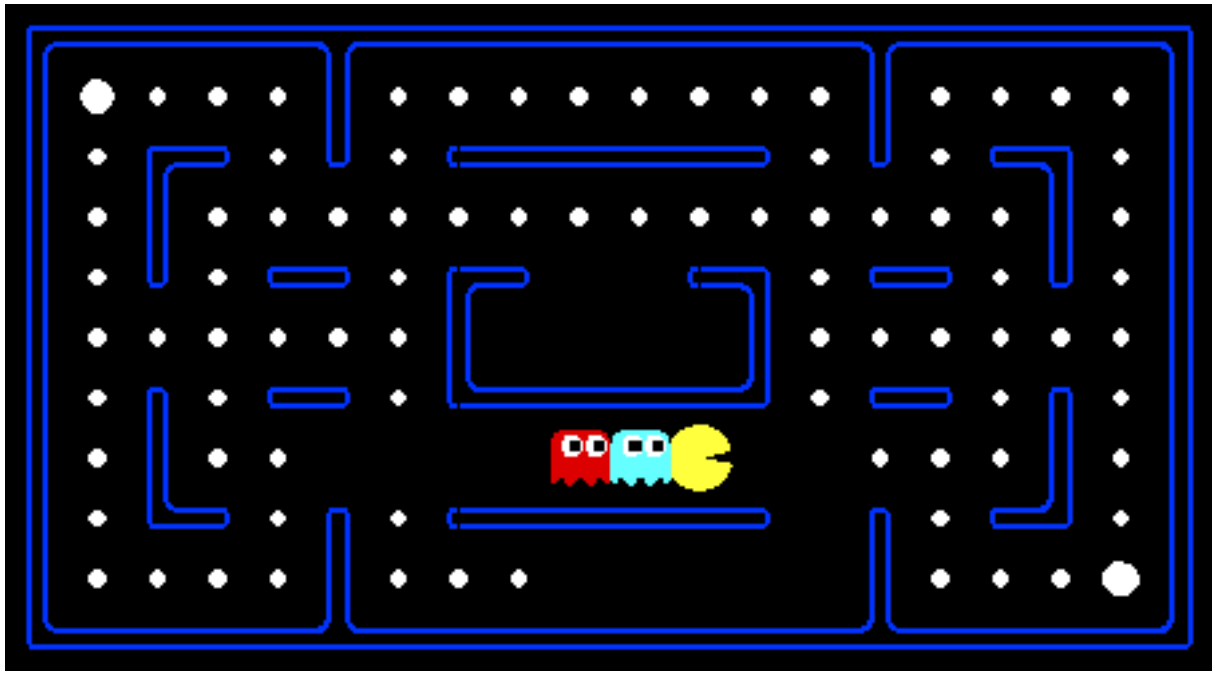


Adversarial games: against opponent (e.g., chess)



- Search problems are adequate models when you are operating in environment that has no uncertainty. However, in many realistic settings, there are other forces at play.
- **Markov decision processes** handle tasks with an element of chance (e.g., Blackjack), where the distribution of randomness is known (reinforcement learning can be employed if it is not).
- **Adversarial games**, as the name suggests, handle tasks where there is an opponent who is working against you (e.g., chess).

Pac-Man



[demo]

cs221.stanford.edu/q

Question

What kind of model is appropriate for playing Pac-Man against ghosts that move into each valid adjacent square with equal probability?

search problem

Markov decision process

adversarial game

Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Bayesian networks

Reflex **States** **Variables**

"Low-level intelligence" "High-level intelligence"

Machine learning

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

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

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

- In state-based models, solutions had a very procedural feel (how to go from A to B). In many applications, the order in which things are done isn't important.

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

Example: Sudoku

Variables: $X_{i,j} \in \{1, \ldots, 9\}$ for $1 \leq i, j \leq 9$

Constraints: Each row of X contains $\{1, \ldots, 9\}$

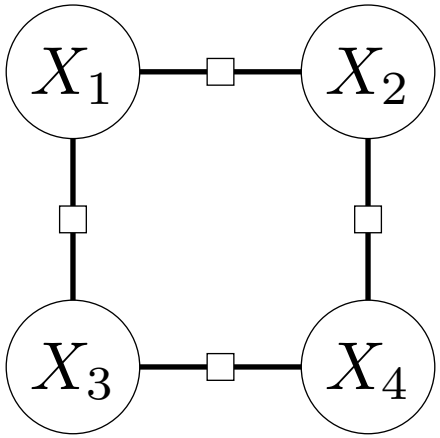
Each column of X contains $\{1, \ldots, 9\}$

Each sub-block of X contains $\{1, \ldots, 9\}$

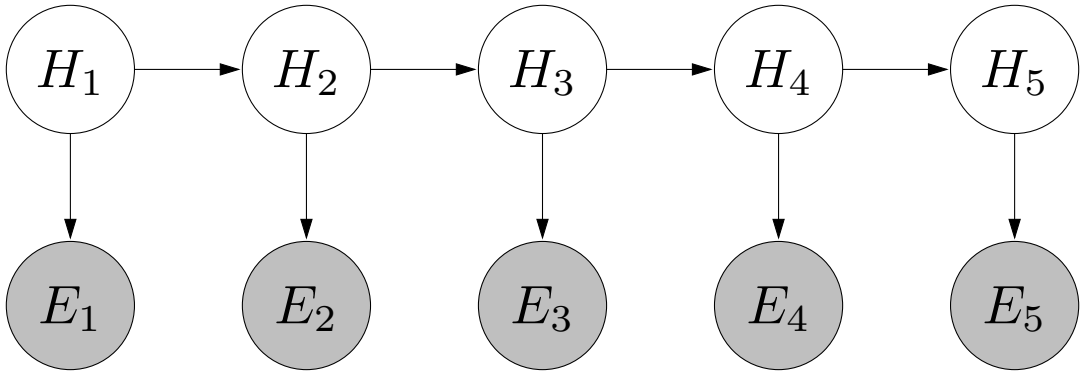
- For example, in Sodoku, it doesn't matter what order the squares are filled in. For these applications, **variable-based models** are more appropriate models. In these models, a solution corresponds to an assignment of values (e.g., digits) to variables (e.g., squares).

Variable-based models

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

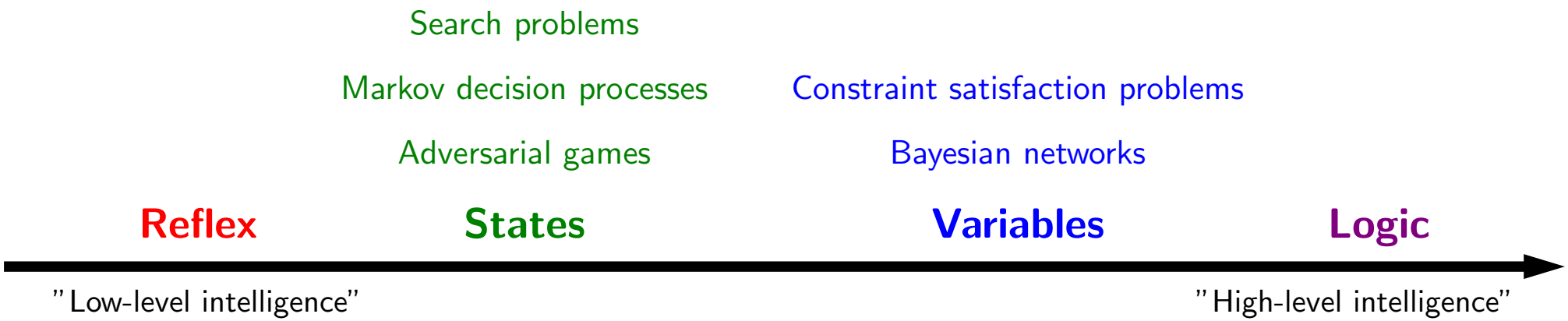


Bayesian networks: soft dependencies (e.g., tracking cars from sensors)



- **Constraint satisfaction problems** are variable-based models where we only have hard constraints. For example, in scheduling, we can't have two people in the same place at the same time.
- **Bayesian networks** are variable-based models where variables are random variables which might be only 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 structure and formally defines a joint probability distribution over all the variables. This is the topic of probabilistic graphical models (CS228).

Course plan



Machine learning

Question

You get extra credit if you write a paper and you solve the problems. You didn't get extra credit, but you did solve the problems. Did you write a paper?

yes

no

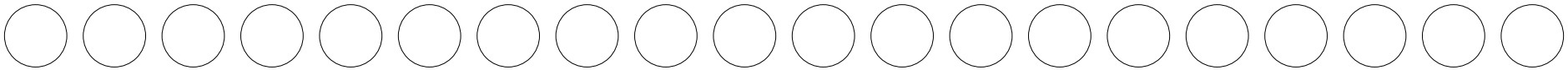
Implicit representation

All students work hard.

John is a student.

Therefore, John works hard.

Variable-based models would explicitly represent all the students — this is inefficient.



Higher-order reasoning

John believes *it will rain*.

Will it rain?

Does John believe *it will not rain* (assuming John is logical)?

Need **expressive power** of logic to represent this...

- Our last stop on the tour is **logic**. Even more so than variable-based models, logic provides a compact language for modeling, which gives us more expressivity.
- It is interesting that historically, logic was one of the first things that AI researchers started with in the 1950s. While logical approaches were in a way quite sophisticated, they failed to scale up to complex real-world tasks. On the other hand, methods based on probability and machine learning do scale up, which is why they presently dominate the AI landscape. However, they have yet been applied successfully to tasks that require really sophisticated reasoning.
- In this course, we will appreciate the two as not contradictory, but simply tackling different aspects of AI — in fact in our schema, logic is a class of models which can be supported by machine learning. An active area of research is to combine the modeling richness of logic with the robustness and agility of machine learning.

Language and logic

Components:

- Natural language parsing
- Knowledge representation
- Logical inference

[demo]

Course plan

Search problems

Markov decision processes

Adversarial games

Constraint satisfaction problems

Bayesian networks

Reflex **States** **Variables** **Logic**

"Low-level intelligence" "High-level intelligence"

Machine learning

Roadmap

Why learn AI?

What topics will you learn?

How will you learn it?

Optimization

Course objectives

Before you take the class, you should...

- Programming (CS 106A, CS 106B, CS 107)
- Discrete math (CS 103)
- Probability (CS 109)

At the end of this course, you should...

- Have a deep understanding of AI methods
- Be able to tackle real-world tasks with the appropriate models and algorithms
- Be more proficient at math and programming

Coursework

- Homeworks (60%)
- Midterm (20%)
- Project (20%)

Homeworks

- 8 homeworks, mix of written and programming problems, centers on an application

Introduction	foundations
Machine learning	sentiment classification
Search	text reconstruction
MDPs	blackjack
Games	Pac-Man
CSPs	course scheduling
Bayesian networks	car tracking
Logic	language and logic

- Some have competitions for extra credit
- When you submit, programming parts will be sanity checked on basic tests; your grade will be based on hidden test cases

Midterm

- Goal: test your ability to use knowledge to solve new problems, not know facts
- All written problems, similar to homeworks
- Closed book except one page of notes
- Covers all material up to and including preceding week
- Tue Nov. 18 from 6pm to 9pm (3 hours)

Project

- Goal: choose any task you care about and apply techniques from class
- Work in groups of up to 3
- Milestones: proposal, progress report, poster session, final report
- Task is completely open, but must follow well-defined steps: task definition, implement baselines/oracles, evaluate on dataset, literature review, error analysis (read website)
- Help: assigned a CA mentor, come to any office hours

Policies

Late days: 8 total late days, max two per assignment

Regrades: come in person to the owner CA of the homework

Collaboration: discuss together, but write up and code yourself; don't look at previous years' solutions

Piazza: ask questions on Piazza, don't email us directly

Piazza: extra credit for students who help answer questions

All details are on the course website

Lecture slides

- Ask questions and provide feedback on individual slides
- Keyboard shortcuts under [Help]
- Lecture notes are available interleaved with slides

Different ways to view the slides:

- One page (press 'p'): make screen large enough for notes to be side-by-side
- Presentation with slide builds (press 'shift-d')
- Text-only outline (press 'shift-o'): for fast loading
- PDF (1pp or 6pp)

Roadmap

Why learn AI?

What topics will you learn?

How will you learn it?

Optimization

Optimization

Models are optimization problems:

$$\min_{x \in C} F(x)$$

Discrete optimization: x is a discrete object

$$\min_{x \in \{\text{abcd,xyz}\}} \text{Length}(x)$$

Algorithmic tool: dynamic programming

Continuous optimization: x is a vector of real numbers

$$\min_{x \in \mathbb{R}} (x - 5)^2$$

Algorithmic tool: gradient descent

- We are now done with the high-level motivation for the class. Let us now dive into some technical details. Recall that modeling converts real-world tasks to models and algorithms solves these models. In this course, we will express our models as **optimization problems**, which provides a mathematical specification of what we want to compute.
- In total generality, optimization problems ask that you find the x that lives in a constraint set C that makes the function $F(x)$ as small as possible.
- There are two types of optimization problems we'll consider: discrete optimization problems and continuous optimization problems. Both are backed by a rich research field and are interesting topics in their own right. For this course, we will use the most basic tools from these topics: **dynamic programming** and **gradient descent**.
- Let us do two practice problems to illustrate each tool. For now, we are assuming that the model (optimization problem) is given and only focus on **algorithms**.

Problem: computing edit distance

Input: two strings, s and t

Output: minimum number of character insertions, deletions, and substitutions it takes to change s into t

Examples:

"cat", "cat"	$\Rightarrow$	0
"cat", "dog"	$\Rightarrow$	3
"cat", "at"	$\Rightarrow$	1
"cat", "cats"	$\Rightarrow$	1
"a cat!", "the cats!"	$\Rightarrow$	4

[live solution]

- Let's consider the formal task of computing the edit distance (or more precisely the Levenshtein distance) between two strings. These measures of dissimilarity have applications in spelling correction, computational biology (applied to DNA sequences).
- As a first step, you should think to break down the problem into subproblems. Observation 1: inserting into s is equivalent to deleting a letter from t (ensures subproblems get smaller). Observation 2: perform edits at the end of strings (might as well start there).
- Consider the last letters of s and t. If these are the same, then we don't need to edit these letters, and we can proceed to the second letters. If they are different, then we have a choice. (i) We can substitute the last letter of s with the last letter of t. (ii) We can delete the last letter of s. (iii) We can insert the last letter of t at the end of s .
- In each of those cases, we can reduce the problem into a smaller problem, but which one? We simply try all of them and take the one that yields the minimum cost!
- We can express this more formally with a mathematical recurrence. These types of recurrences will show up throughout the course, so it's a good idea to be comfortable with them. Before writing down the actual recurrence, the first step is to express the quantity that we wish to compute. In this case: let $d(m, n)$ be the edit distance between the first m letters of s and the first n letters of t. Then we have

$$d(m,n) = \begin{cases} m & \text{if } n = 0 \\ n & \text{if } m = 0 \\ d(m-1, n-1) & \text{if } s\_m = t\_n \\ 1 + \min\{d(m-1,n-1), d(m-1,n), d(m,n-1)\} & \text{otherwise.} \end{cases}$$

- Once you have the recurrence, you can code it up. The straightforward implementation will take exponential time, but you can **memoize** the results to make it $O(n^2)$ time. The end result is the dynamic programming solution: recurrence + memoization.

Problem: finding the least squares line

Input: set of pairs $\{(x_1, y_1), \ldots, (x_n, y_n)\}$

Output: $w \in \mathbb{R}$ that minimizes the squared error

$$F(w) = \sum_{i=1}^{n} (x_i w - y_i)^2$$

Examples:

$$\begin{aligned} \{(2,4)\} &\Rightarrow 2 \\ \{(2,4),(4,2)\} &\Rightarrow ? \end{aligned}$$

[live solution]

- The formal task is this: given a set of n two-dimensional points (x_i, y_i) which defines $F(w)$, compute the w that minimizes $F(w)$.
- A brief detour to explain the modeling that might lead to this formal task. **Linear regression** is an important problem in machine learning, which we will come to later. Here's a motivation for the problem: suppose you're trying to understand how your exam score (y) depends on the number of hours you study (x). Let's posit a linear relationship $y = wx$ (not exactly true in practice, but maybe good enough). Now we get a set of training examples, each of which is a (x_i, y_i) pair. The goal is to find the slope w that best fits the data.
- Back to algorithms for this formal task. We would like an algorithm for optimizing general types of $F(w)$. So let's **abstract away from the details**. Start at a guess of w (say $w = 0$), and then iteratively update w based on the derivative (gradient if w is a vector) of $F(w)$. The algorithm we will use is called **gradient descent**.
- If the derivative $F'(w) < 0$, then increase w; if $F'(w) > 0$, decrease w; otherwise, keep w still. This motivates the following update rule, which we perform over and over again: $w \leftarrow w - \eta F'(w)$, where $\eta > 0$ is a **step size** which controls how aggressively we change w.
- If η is too big, then w might bounce around and not converge. If η is too small, then we w might not move very far to the optimum. Choosing the right value of η is rather tricky and there is no good clean answer. Empirically, one typically just tries a few values and sees which one works best, developing some intuition in the process.
- Now to specialize to our function, we just need to compute the derivative, which is an elementary calculus exercise: $F'(w) = \sum_{i=1}^{n} 2(x_i w - y_i)x_i$.

Summary

- AI applications are high-impact and complex
- Think in terms of modeling + algorithms
- Models: learning + [reflex, states, variables, logic]
- Section this Friday at 3pm: review of foundations
- Homework [foundations]: due next Tuesday 11pm
- Course will be fast-paced and exciting!

cs221.stanford.edu/q

Question

What was the most surprising thing you learned today?