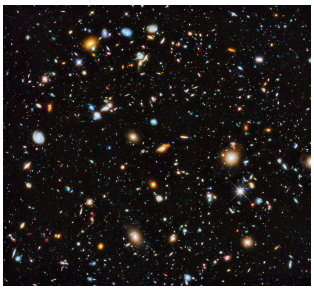
Lecture 1: Overview




Chelsea Finn


Nima Anari


Amelie Byun
(Course Coordinator)


Cindy Jiang
(Head CA)


Andrew Tan


Bryan Kim


Dilip Arumugam


Haoshen Hong


Jennie Chen


Jerry Qu


Marcus Pålsson


Prerna Dhareshwar


Rohan Sampath


Will Deaderick

Our Priorities and Principles

- Sad that we are not together on campus.
- It will not be the same.
- We prioritize supporting each other, as humans.
- We will prioritize clear communication, accessible content and resources.
- We will remain flexible and adjust to the situation

- It will not be the same as a live lecture, but we are going to do what we can to create a good experience.
- We recognize that many of you may be going through really challenging situations right now.
- Our priority is to support each other, as humans, and we ask you to do the same.
- Because we know that everyone is going through different situations, we will prioritize clear communication, accessibility, and remain flexible.

The Plan for Lectures

- Live zoom lectures, recording posted on canvas afterward
- Chat for class-wide communication, Q&A for questions.
- TAs will monitor Q&A, moderate, answer questions.
- Unanswered questions will be answered on Piazza after class.
- Ice breaker (in chat): Why do you want to learn about AI in 3 words?

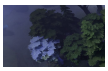
- Recording this so that folks in other timezones can see it.
- We encourage you to come to lecture if possible
- TAs will stick around after class to answer further questions.

Announcements

- Section: this Thursday, overview of foundations
- Homework foundations is out, due next Tuesday 11pm
- Gradescope code will be posted on Piazza

Microsoft creates AI that can read a document and answer questions about it as well as a person

January 15, 2018 | Allison Linn



Microsoft researchers achieve new conversational speech recognition milestone

August 20, 2017 | By Xuedong

June 24, 2014

DeepFace: Closing Performance in Fa

Conference on Computer Vision and Pattern Recognition (CVPR)

By: Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, Lior Wolf

Abstract

In modern face recognition, the conventional pipeline co
classify. We revisit both the alignment step and the repr
modeling in order to apply a piecewise affine transforma
layer deep neural network. This deep network involves m
locally connected layers without weight sharing, rather t
trained it on the largest facial dataset to-date, an identity
belonging to more than 4,000 identities.

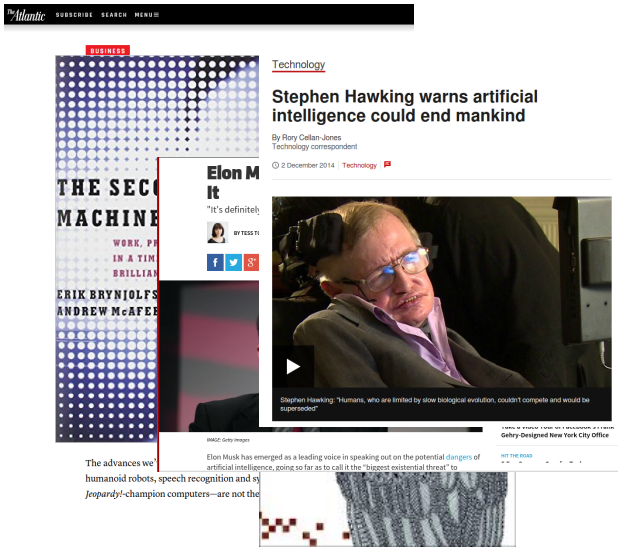
If you think AI will never replace radiologists—you may want to think again

May 14, 2018 | Michael Walter | Artificial Intelligence

It's one of the most frequently discussed questions in radiology today: What kind of long-term impact will artificial intelligence (AI) have on radiologists?

Robert Schier, MD, a radiologist for RadNet, shared his own thoughts on the topic in a new commentary published by the *Journal of the American College of Radiology*—and he's not quite as optimistic as some of his colleagues throughout the industry.

- It is generally not hard to motivate AI these days. There have been some substantial success stories. A lot of the triumphs have been in **games**, such as Jeopardy! (IBM Watson, 2011), Go (DeepMind's AlphaGo, 2016), Dota 2 (OpenAI, 2019), Poker (CMU and Facebook, 2019).
- On non-game tasks, we also have systems that achieve strong performance on reading comprehension, speech recognition, face recognition, and medical imaging **benchmarks**.
- Unlike games, however, where the game is the full problem, good performance on a benchmark does not necessarily translate to good performance on the actual task in the wild. Just because you ace an exam doesn't necessarily mean you have perfect understanding or know how to apply that knowledge to real problems.
- So, while promising, not all of these results translate to real-world applications



- From the non-scientific community, we also see speculation about the future: that it will bring about sweeping societal change due to automation, resulting in massive job loss, not unlike the industrial revolution, or that AI could even surpass human-level intelligence and seek to take control.
- While these are extreme views, there is no doubt that AI is and will continue to be transformational. We still don't know exactly what that transformation will look like.

1956

Birth of AI

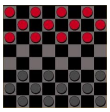
1956: Workshop at Dartmouth College; attendees: John McCarthy, Marvin Minsky, Claude Shannon, etc.

Aim for **general principles**:

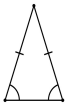
Every aspect of learning or any other feature of intelligence can be so precisely described that a machine can be made to simulate it.

- How did we get here? The name **artifical intelligence** goes back to a summer in 1956. John McCarthy, who was then at MIT but later founded the Stanford AI lab, organized a workshop at Dartmouth College with the leading thinkers of the time, and set out a very bold proposal...to build a system that could do it **all**.

Birth of AI, early successes



Checkers (1952): Samuel's program learned weights and played at strong amateur level



Problem solving (1955): Newell & Simon's Logic Theorist: prove theorems in Principia Mathematica using search + heuristics; later, General Problem Solver (GPS)

- While they did not solve it all, there were a lot of **interesting programs** that were created: programs that could play checkers at a strong amateur level, programs that could prove theorems.
- For one theorem Newell and Simon's Logical Theorist actually found a proof that was more elegant than what a human came up with. They actually tried to publish a paper on it but it got rejected because it was not a new theorem; perhaps they failed to realize that the third author was a computer program.
- From the beginning, people like John McCarthy sought **generality**, thinking of how commonsense reasoning could be encoded in logic. Newell and Simon's General Problem Solver promised to solve any problem (which could be suitably encoded in logic).

Overwhelming optimism...

Machines will be capable, within twenty years, of doing any work a man can do. —Herbert Simon

Within 10 years the problems of artificial intelligence will be substantially solved. —Marvin Minsky

I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines. —Claude Shannon

- It was a time of high optimism, with all the leaders of the field, all impressive thinkers, predicting that AI would be "solved" in a matter of years.

...underwhelming results

Example: machine translation

The spirit is willing but the flesh is weak.

↓

(Russian)

↓

The vodka is good but the meat is rotten.

1966: ALPAC report cut off government funding for MT, first AI winter

- Despite some successes, certain tasks such as machine translation were complete failures, which lead to the cutting of funding and the first AI winter.

Implications of early era

Problems:

- **Limited computation**: search space grew exponentially, outpacing hardware ($100! \approx 10^{157} > 10^{80}$)
- **Limited information**: complexity of AI problems (number of words, objects, concepts in the world)

Contributions:

- Lisp, garbage collection, time-sharing (John McCarthy)
- Key paradigm: separate **modeling** and **inference**

- What went wrong? It turns out that the real world is very complex and most AI problems require a lot of **compute** and **data**.
- The hardware at the time was simply too limited both compared to the human brain and computers available now. Also, casting problems as general logical reasoning meant that the approaches fell prey to the exponential search space, which no possible amount of compute could really fix.
- Even if you had infinite compute, AI would not be solved. There are simply too many words, objects, and concepts in the world, and this information has to be somehow encoded in the AI system.
- Though AI was not solved, a few generally useful technologies came out of the effort, such as Lisp (still the world's most advanced programming language in a sense).
- One particularly powerful paradigm is the separation between what you want to compute (modeling) and how to compute it (inference).

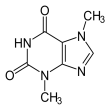
Knowledge-based systems (70-80s)

Expert systems: elicit specific domain knowledge from experts in form of rules:

```
if [premises] then [conclusion]
```


- In the seventies and eighties, AI researchers looked to knowledge as a way to combat both the limited computation and information problems. If we could only figure out a way to encode prior knowledge in these systems, then they would have the necessary information and also have to do less compute.

Knowledge-based systems (70-80s)



DENDRAL: infer molecular structure from mass spectrometry

MYCIN: diagnose blood infections, recommend antibiotics

XCON: convert customer orders into parts specification; save DEC $\$40$ million a year by 1986

- Instead of the solve-it-all optimism from the 1950s, researchers focused on building narrow practical systems in targeted domains. These became known as **expert systems**.

Knowledge-based systems

Contributions:

- First **real application** that impacted industry
- Knowledge helped curb the exponential growth

Problems:

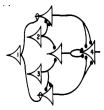
- Knowledge is not deterministic rules, need to model **uncertainty**
- Requires considerable **manual effort** to create rules, hard to maintain

1987: Collapse of Lisp machines and second AI winter

- This was the first time AI had a measurable impact on industry. However, the technology ran into limitations and failed to scale up to more complex problems. Due to plenty of overpromising and underdelivering, the field collapsed again.
- We know that this is not the end of the AI story, but actually it is not the beginning. There is another thread for which we need to go back to 1943.

1943

Artificial neural networks

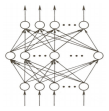


1943: introduced artificial neural networks, connect neural circuitry and logic (McCulloch/Pitts)

1969: Perceptrons book showed that linear models could not solve XOR, killed neural nets research (Minsky/Papert)

- Much of AI's history was dominated by the logical tradition, but there was another smaller camp, grounded in neural networks inspired by the brain.
- (Artificial) neural networks were introduced by a famous paper by McCulloch and Pitts, who devised a simple mathematical model and showed how it could be be used to compute arbitrary logical functions.
- Much of the early work was on understanding the mathematical properties of these networks, since computers were too weak to do anything interesting.
- In 1969, a book was published that explored many mathematical properties of Perceptrons (linear models) and showed that they could not solve some simple problems such as XOR. Even though this result says nothing about the capabilities of deeper networks, the book is largely credited with the demise of neural networks research, and the continued rise of logical AI.

Training networks



1986: popularization of backpropagation for training multi-layer networks (Rumelhardt, Hinton, Williams)

1989: applied convolutional neural networks to recognizing handwritten digits for USPS (LeCun)

- In the 1980s, there was a renewed interest in neural networks. Backpropagation was rediscovered and popularized as a way to actually train deep neural networks, and Yann LeCun built a system based on convolutional neural networks to recognize handwritten digits. This was one of the first successful uses of neural networks, which was then deployed by the USPS to recognize zip codes.

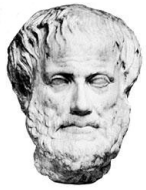
Deep learning

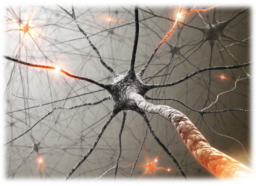
AlexNet (2012): huge gains in object recognition; transformed computer vision community overnight

AlphaGo (2016): deep reinforcement learning, defeat world champion Lee Sedol

- The real break for neural networks came in the 2010s. With the rise of compute (notably GPUs) and large datasets such as ImageNet (2009), the time was ripe for the world to take note of neural networks.
- AlexNet was a pivotal system that showed the promise of deep convolutional networks on ImageNet, the benchmark created by the computer vision community who was at the time still skeptical of deep learning. Many other success stories in speech recognition and machine translation followed.

Two intellectual traditions





- AI has always swung back and forth between the two
- Deep philosphical differences, but deeper connections (McCulloch/Pitts, AlphaGo)?

- Reflecting back on the past of AI, there have been two intellectual traditions that have dominated the scene: one rooted in logic and one rooted in neuroscience (at least initially). This debate is paralleled in cognitive science with connectionism and computationalism.
- While there are deep philosophical differences, perhaps there are deeper connections.
- For example, McCulloch and Pitts' work from 1943 can be viewed as the root of deep learning, but that paper is mostly about how to implement logical operations.
- The game of Go (and indeed, many games) can be perfectly characterized by a set of simple logic rules. At the same time, the most successful systems (AlphaGo) do not tackle the problem directly using logic, but appeal to the fuzzier world of artificial neural networks.

A melting pot

- Bayes rule (Bayes, 1763) from **probability**
- Least squares regression (Gauss, 1795) from **astronomy**
- First-order logic (Frege, 1893) from **logic**
- Maximum likelihood (Fisher, 1922) from **statistics**
- Artificial neural networks (McCulloch/Pitts, 1943) from **neuroscience**
- Minimax games (von Neumann, 1944) from **economics**
- Stochastic gradient descent (Robbins/Monro, 1951) from **optimization**
- Uniform cost search (Dijkstra, 1956) from **algorithms**
- Value iteration (Bellman, 1957) from **control theory**

- Of course, any story is incomplete.
- In fact, for much of the 1990s and 2000s, neural networks were not popular in the machine learning community, and the field was dominated more by techniques such as Support Vector Machines (SVMs) inspired by statistical theory.
- The fuller picture is that the modern world of AI is more like New York City—it is a melting pot that has drawn from many different fields ranging from statistics, algorithms, economics, etc.
- And often it is the new connections between these fields that are made and their application to important real-world problems that makes working on AI so rewarding.

Roadmap

A brief history

Two views

Course overview

Course logistics

Optimization

Two views of AI

AI agents: how can we create intelligence?

AI tools: how can we benefit society?

- There are two ways to look at AI philosophically.
- The first is the science and engineering of building "intelligent" agents. The inspiration of what constitutes intelligence comes from the types of capabilities that humans possess: the ability to perceive a very complex world and make enough sense of it to be able to manipulate it.
- The second views AI as a set of tools. We are simply trying to solve problems in the world, and techniques developed by the AI community happen to be useful for that, but these problems are not ones that humans necessarily do well on natively.
- However, both views boil down to many of the same day-to-day activities (e.g., collecting data and optimizing a training objective), the philosophical differences do change the way AI researchers approach and talk about their work. Moreover, the conflation of these two views can generate a lot of confusion.

AI agents...

An intelligent agent

Perception Robotics Language

Knowledge Reasoning Learning

- The starting point for the agent-based view is ourselves.
- As humans, we have to be able to perceive the world (computer vision), perform actions in it (robotics), and communicate with other agents (language).
- We also have knowledge about the world (from procedural knowledge like how to ride a bike, to declarative knowledge like remembering the capital of France), and using this knowledge we can draw inferences and make decisions (reasoning).
- Finally, we learn and adapt over time. We are born with none of the skills that we possess as adults, but rather the capacity to acquire them. Indeed machine learning has become the primary driver of many of the AI applications we see today.

Are we there yet?

Machines: narrow tasks, millions of examples

Humans: diverse tasks, very few examples

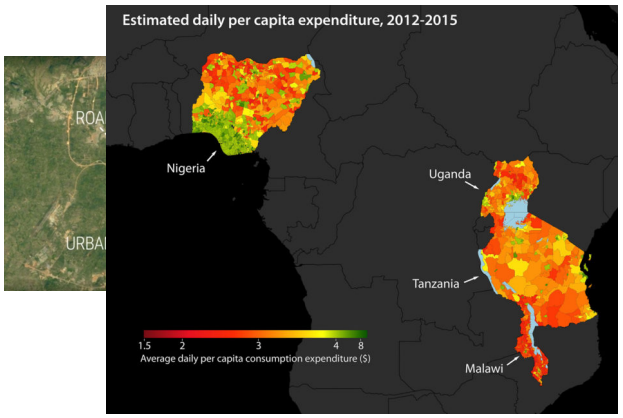
- The AI agents view is an inspiring quest to undercover the mysteries of intelligence and tackle the tasks that humans are good at. While there has been a lot of progress, we still have a long way to go along some dimensions: for example, the ability to learn quickly from few examples or the ability to perform commonsense reasoning.
- There is still a huge gap between the regimes that humans and machines operate in. For example, AlphaGo learned from 19.6 million games, but can only do one thing: play Go. Humans on the other hand learn from a much wider set of experiences, and can do many things.

AI tools...

- The other view of AI is less about re-creating the capabilities that humans have, and more about how to benefit humans.
- Even the current level of technology is already being deployed widely in practice, and many of these settings are often not particularly human-like (targeted advertising, news or product recommendation, web search, supply chain management, etc.)

[Jean et al. 2016]

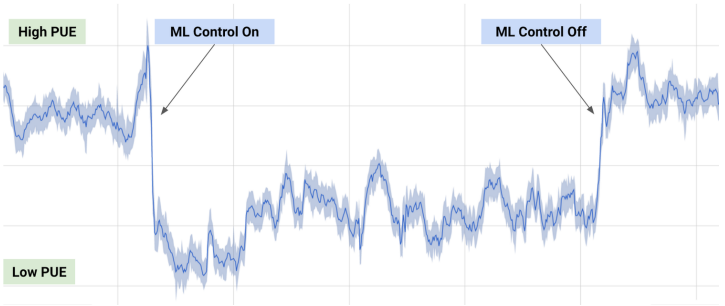
Predicting poverty



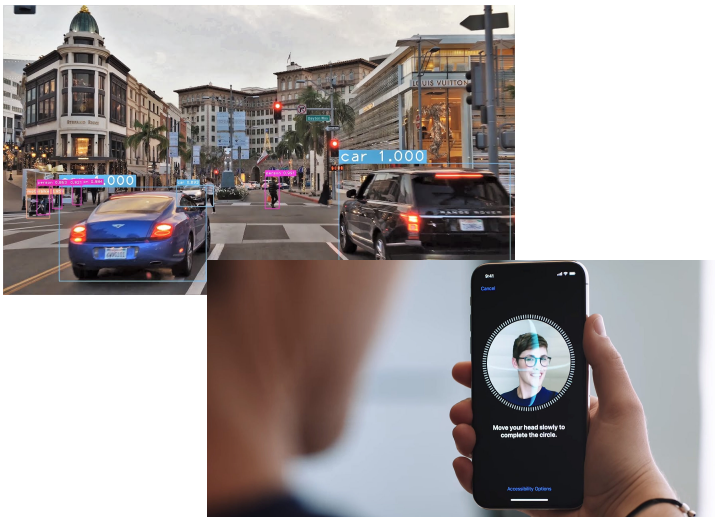
- Computer vision techniques, used to recognize objects, can also be used to tackle social problems. Poverty is a huge problem, and even identifying the areas of need is difficult due to the difficulty in getting reliable survey data. Recent work has shown that one can take satellite images (which are readily available) and predict various poverty indicators.

[DeepMind]

Saving energy by cooling datacenters



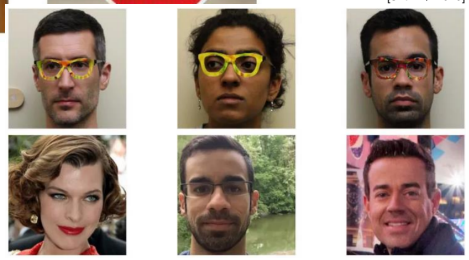
- Machine learning can also be used to optimize the energy efficiency of datacenters which, given the hunger for compute these days, makes a big difference. Some recent work from DeepMind shows how to significantly reduce Google's energy footprint by using machine learning to predict the power usage effectiveness from sensor measurements such as pump speeds, and use that to drive recommendations.



Security

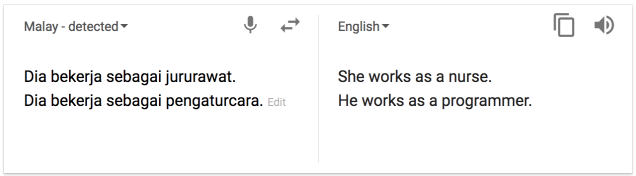
[Evtimov+ 2017]

[Sharif+ 2016]



- Other applications such as self-driving cars and authentication have high-stakes, where errors could be much more damaging than getting the wrong movie recommendation. These applications present a set of security concerns.
- One can generate so-called **adversarial examples**, where by putting stickers on a stop sign, one can trick a computer vision system into mis-classifying it as a speed limit sign. You can also purchase special glasses that fool a system into thinking that you're a celebrity.

Bias in machine translation



society ⇒ data ⇒ predictions

- A more subtle case is the issue of bias. One might naively think that since machine learning algorithms are based on mathematical principles, they are somehow objective. However, machine learning predictions come from the training data, and the training data comes from society, so any biases in society are reflected in the data and propagated to predictions. The issue of bias is a real concern when machine learning is used to decide whether an individual should receive a loan or get a job.
- Unfortunately, the problem of fairness and bias is as much of a philosophical one as it is a technical one. There is no obvious "right thing to do", and it has even been shown mathematically that it is impossible for a classifier to satisfy three reasonable fairness criteria (Kleinberg et al., 2016).

Fairness in criminal risk assessment

- Northpointe: COMPAS predicts criminal risk score (1-10)
- ProPublica: given that an individual did not reoffend, blacks 2x likely to be (wrongly) classified 5 or above
- Northpointe: given a risk score of 7, 60% of whites reoffended, 60% of blacks reoffended

Summary so far

- AI agents: achieving human-level intelligence, still very far (e.g., generalize from few examples)

- AI tools: need to think carefully about real-world consequences (e.g., security, biases)

Roadmap

A brief history

Two views

Course overview

Course logistics

Optimization

How do we solve AI tasks?

- How should we actually solve AI tasks? The real world is complicated. At the end of the day, we need to write some code (and possibly build some hardware, too). But there is a huge chasm.

Paradigm

Modeling

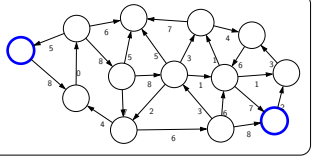
Inference **Learning**

- In this class, we will adopt the **modeling-inference-learning** paradigm to help us navigate the solution space. In reality the lines are blurry, but this paradigm serves as an ideal and a useful guiding principle.

Paradigm: modeling

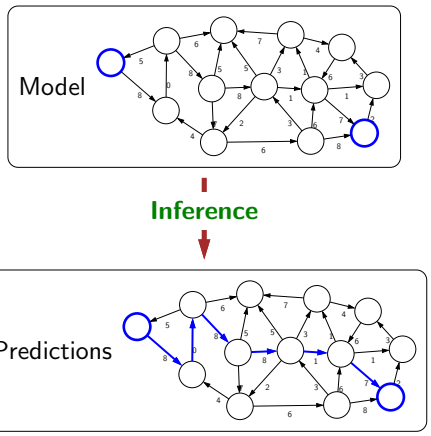
Real world 

Modeling

Model 

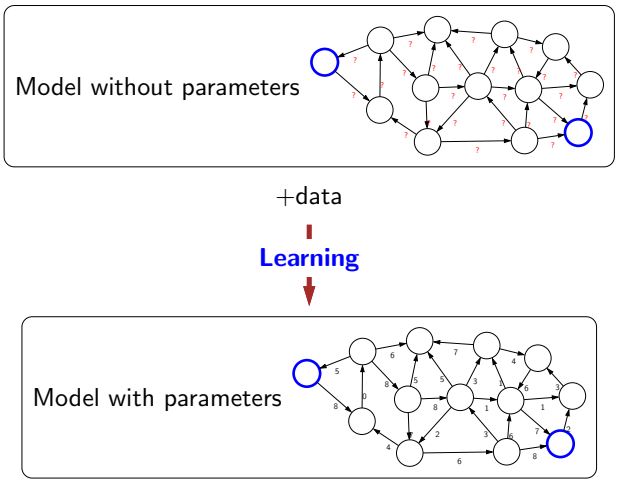
- The first pillar is modeling. Modeling takes messy real world problems and packages them into neat formal mathematical objects called **models**, which can be subject to rigorous analysis and can be operated on by computers. However, modeling is lossy: not all of the richness of the real world can 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 Sodoku, where the real world is identical to the model.)
- As an example, suppose we're trying to have an AI that can navigate through a busy city. We might formulate this 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.

Paradigm: inference



- 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



- 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 right type of data, we can run a machine learning algorithm to tune the parameters of the model.
- Note that learning here is not tied to a particular approach (e.g., neural networks), but more of a philosophy. This general paradigm will allow us to bridge the gap between logic-based AI and statistical AI.

Course plan

Machine learning

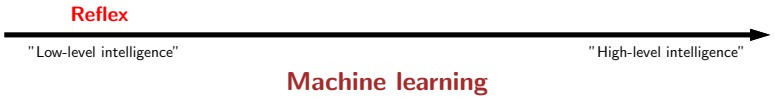
- We now embark on our tour of the topics in this course. The topics correspond to types of models that we can use to represent real-world tasks. The topics will in a sense advance from low-level intelligence to high-level intelligence, evolving from models that simply make a reflex decision to models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

Machine learning

- The main driver of recent successes in AI
- Move from "code" to "data" to manage the information complexity
- 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 systems engineering perspective, machine learning allows us to shift the information complexity of the model from code to data, which is much easier to obtain (either naturally occurring or via crowdsourcing).
- 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 even be formalized using tools from probability and statistical learning theory.

Course plan

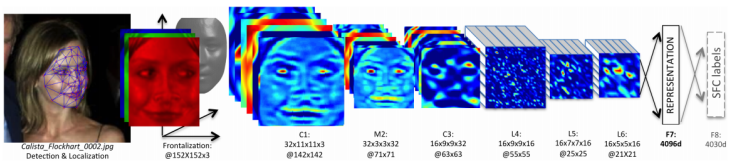


Machine learning

What is this animal?

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 trivial in these models because it is just running the fixed computations, which makes these models appealing.

Course plan

Search problems

Markov decision processes

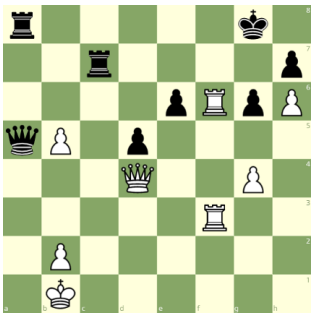
Adversarial games

Reflex **States**

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

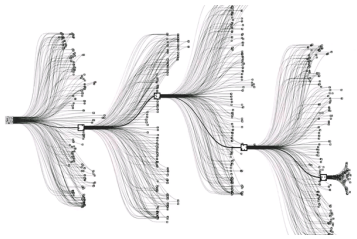
Machine learning

State-based models



White to move

State-based models



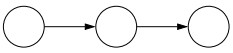
Applications:

- Games: Chess, Go, Pac-Man, Starcraft, etc.
- Robotics: motion planning
- Natural language generation: machine translation, image captioning

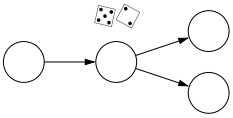
- Reflex-based models are too simple for tasks that require more forethought (e.g., in playing chess or planning a big trip). State-based models overcome this limitation.
- The key idea is, at a high-level, to model the **state** of a world and transitions between states which are triggered by actions. 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

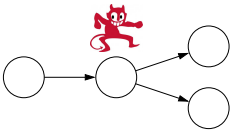
Search problems: you control everything



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

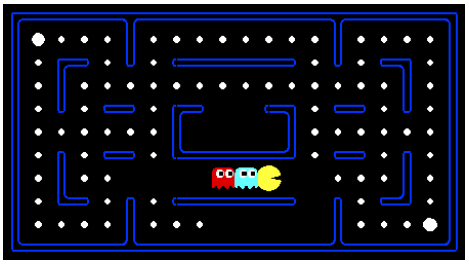


Adversarial games: against opponent (e.g., chess)



- Search problems are adequate models when 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 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]

Course plan

Search problems

Markov decision processes — Constraint satisfaction problems

Adversarial games — 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

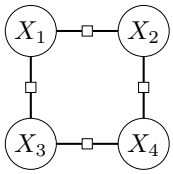
Goal: put digits in blank squares so each row, column, and 3x3 sub-block has digits 1–9

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

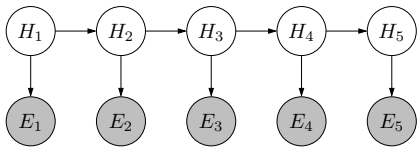
- In state-based models, solutions are procedural: they specify step by step instructions on how to go from A to B. In many applications, the order in which things are done isn't important.

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 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 structure and it formally defines a joint probability distribution over all of the variables. This topic is studied thoroughly in probabilistic graphical models (CS228).

Course plan

Search problems

Markov decision processes | Constraint satisfaction problems

Adversarial games | Bayesian networks

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

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

Machine learning

Motivation: virtual assistant

Tell information →

← **Ask** questions

Use natural language!

[demo]

Need to:

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

- 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 did not work well on complex real-world tasks with noise and uncertainty. On the other hand, methods based on probability and machine learning naturally handle noise and uncertainty, which is why they presently dominate the AI landscape. However, they are yet to be 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 course plan, logic is a class of models which can be supported by machine learning. An active area of research is to combine the richness of logic with the robustness and agility of machine learning.

- One motivation for logic is a virtual assistant. At an abstract level, one fundamental thing a good personal assistant should be able to do is to take in information from people and be able to answer questions that require drawing inferences from these facts.
- In some sense, telling the system information is like machine learning, but it feels like a very different form of learning than seeing 10M images and their labels or 10M sentences and their translations. The type of information we get here is both more heterogenous, more abstract, and the expectation is that we process it more deeply (we don't want to tell our personal assistant 100 times that we prefer morning meetings).
- And how do we interact with our personal assistants? Let's use natural language, the very tool that was built for communication!

Course plan

Search problems

Markov decision processes | Constraint satisfaction problems

Adversarial games | Bayesian networks

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

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

Machine learning

Roadmap

A brief history

Two views

Course overview

Course logistics

Optimization

Course objectives

Before you take the class, you should know...

- Programming (CS 106A, CS 106B, CS 107)
- Discrete math, mathematical rigor (CS 103)
- Probability (CS 109)
- Algorithms (CS 161)

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

- Be able to tackle real-world tasks with the appropriate techniques
- Be more proficient at math and programming

Coursework

- Homeworks (40%)
- Midterm Exam (30%)
- **Optional** Project (30%)
- If you opt out of project: homework (60%), midterm (40%)

Grading

- Satisfactory / No Credit

Homeworks

- 7 homeworks, mix of written and programming problems, each 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

- Pac-Man competition for extra credit
- When you submit, programming parts will be run on all test cases, but only get feedback on a subset
- Check the outputs of your submitted code!

Midterm Exam

- Goal: test your ability to use knowledge to solve new problems, not know facts
- All written problems (look at past exam problems for style)
- Covers all material up to and including May 20th (week 7)
- Time-limited, open-universe (but no collaboration or plagiarism)
- Tuesday, June 2nd (3 hours, within 24 hr window)

Optional Project

- Goal: choose any task you care about and apply techniques from class
- Work in groups of up to 4; find a group early (your responsibility), grading agnostic to group size!
- Milestones: proposal, progress report, poster, 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

Gradescope: submit all assignments there

Late days: 7 total late days, max two per assignment (not for final project report, poster)

Piazza: ask questions on Piazza, do not email us directly except for OAE letters

Piazza: extra credit for students who help answer questions

All details are on the course website

THE HONOR CODE

- Do collaborate and discuss together, but write up and code independently.
- Do not look at anyone else's writeup or code.
- Do not show anyone else your writeup or code or post it online (e.g., GitHub).
- When debugging, only look at input-output behavior.
- We will run MOSS periodically to detect plagarism.

Roadmap

A brief history

Two views

Course overview

Course logistics

Optimization

Optimization

Discrete optimization: find the best discrete object

$$\min_{p \in \text{Paths}} \text{Cost}(p)$$

Algorithmic tool: dynamic programming

Continuous optimization: find the best vector of real numbers

$$\min_{\mathbf{w} \in \mathbb{R}^d} \text{TrainingError}(\mathbf{w})$$

Algorithmic tool: gradient descent

- We are now done with the high-level motivation for the class. Let us now dive into some technical details. Let us focus on the inference and the learning aspect of the **modeling-inference-learning** paradigm.
- We will approach inference and learning from an **optimization** perspective, which allows us to decouple the mathematical specification of **what** we want to compute from the algorithms for **how** to compute it.
- 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 (mostly for inference) and continuous optimization problems (mostly for learning). 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

[semi-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 and computational biology (applied to DNA sequences).
- As a first step, you should think of breaking 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 letter 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-to-last letters. If they are different, then we have three choices. (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 quadratic time (in this case, $O(nm)$). 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:

$\{(2,4)\}$	$\Rightarrow$	2
$\{(2,4),(4,2)\}$	$\Rightarrow$	?

[semi-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 $F(w)$.
- **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** that 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 w might not move very far to the optimum. Choosing the right value of η can be rather tricky. Theory can give rough guidance, but this is outside the scope of this class. Empirically, we will just try a few values and see which one works best. This will help us develop 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

- History: roots from logic, neuroscience, statistics—melting pot!
- Modeling [reflex, states, variables, logic] + inference + learning paradigm
- AI has high societal impact, how to steer it positively?