CAPS Notes - Lecture 1
Learning and Decision-Making

Leo Klenner, Henry Fung, Cory Combs

Last updated: 10/28/2019

Overview

This course is about how machines - from single algorithms to autonomous agents - make decisions. The course explores how generalists, working together with specialists, can guide decision-making to ensure that the decisions machines make are "good".

What does it mean to make a "good" decision? That's one of the themes we begin to unpack in this lecture.

Lecture one is a primer on algorithmic decision-making:

- What is an algorithm?
- How do algorithms that train on data (i.e. learn) differ from those that don't train on data?

We also explore, at a foundational level, how the decision-making of algorithms compares to that of humans:

- How can we conceptualize the different ways in which algorithms and humans make decisions?
- How can we use an understanding of such differences to ensure that deploying an algorithm results in "good" decisions?

1. Definitions and Background of AI

Definitions

AI

To talk about AI, we first need to define it. Based on [1, 2], we can outline four possible definitions of AI, all of the form "AI is the field that aims to build..." The definitions can be read in two-dimensions. One dimension is whether the goal is to match human performance or ideal rationality. The other dimension is whether the goal is to build systems that reason or systems that act (without reason).

	Human-based	**Ideal Rationality-based**
Reasoning-based	Systems that think like humans	Systems that think rationally
Behavior-based	Systems that act like humans	Systems that act rationally

Table 1: Overview of four definitions of AI

Each definition reflects an approach within AI research and each comes with its own conceptual challenges.

In this course, we follow [2] and define AI as "systems that act rationally".

Adopting this definition brings about a fundamental challenge concerning the type of rationality entailed in "systems that act rationally". If this definition of AI forces us to interpret "rationally" as "perfectly rational", then we run into a host of issues, from technical ones, such as how we can make sense of a system's decisions if they don't appear to be goal-directed, to ethical ones, such as whether each decision of the system must be seen as "good".

To clarify this aspect, we need to understand what different types of rationality we can pursue with AI.

Rationality

Adopting the definitions of [3], we can differentiate between three types of rationality:

- **Perfect rationality**, or the capacity to generate maximally successful behavior given the available information
- **Calculative rationality**, or the capacity to compute, in principle, the perfectly rational decision given the initially available information
- **Bounded optimality**, or the capacity to generate maximally successful behavior given the available information and computational resources

Type	**Information**	**Computation**	**Time**	**Frequency**	**Desirability**
Perfect rationality	Yes	No	Yes	Rarely exists	High
Calculative rationality	Yes	Yes	No	Often exists	Low
Bounded optimality	Yes	Yes	Yes	Often exists	Depends on the bounds

Table 2:Overview of differences between types of rationality

Table 2, above, further differentiates the types of rationality based on five variables:

- constraints on information
- constraints on computation (either compute power available or program specification)
- constraints on time (deadlines for making a decision)
- frequency (how often the type of rationality can be observed for real-world systems)
- desirability (how desirable it is to have a system of this type of rationality)

To understand why calculative rationality might not be desirable, consider a chess program that choses the "right" move but takes 10^5 times too long to make it.

How does this distinction help us adopt the definition of "systems that act rationally"? When we talk about AI, we assume that a system's decisions are subject to bounded rationality.

Agent and Environment

Taking on the definition of bounded rationality, we know that the decisions we analyze are subject to two set of constraints:

- The constraints of the **agent** that makes the decisions, arising from how the agent is specified in terms of the algorithm that it computes and the resources it has to perform this computation.
- The constraints of the **environment** in which the decisions are carried out, arising from aspects like imperfect information or other domain-specific features that might challenge the agent's performance (e.g. through necessitating fast decisions).

There is also a third sets of constraints:

- The constraints of the **agent-environment interaction**, arising from aspects like whether the environment can return the feedback to the agent that the agent needs to make decisions.

We will look more at agent-environment interaction later in the course, particularly in context of reinforcement learning. For now, it is important to realize that, while engineers might know a lot about the specification of the agent, robust decision-making also requires domain experts who can understand the environment in which the agent's decisions take place.

Background

When we talk about AI, we do not necessarily mean algorithms that train on data (i.e. learn). We could also be talking about **rule-based systems**, which make decisions by following sets of rules specified by their developers. These systems "know" without having "learned" through observations (i.e. data).

Systems that are based on **machine learning** techniques are a more recent development within AI. Note that both options are in line with the definition of "systems that act rationally".

The central difference between the two options is that the decision-making of rule-based systems is **deterministic**, whereas the decision-making of learning-based systems is **non-deterministic**. (When we turn to neural networks later in the course, we will see that the *training* of a neural network is non-deterministic, whereas *once trained*, a neural network can be deterministic.)

"Deterministic" here means that given the same input, the system will always return the same output. With deterministic systems, we have a **linear mapping** of inputs to outputs.

```
# Simple example of deterministic decision-making:
# whenever A is true, B is assigned the value 2

if condition A is True:
    then set variable B = 2
```

Note, that this doesn't mean that rule-based systems are necessarily simple. In fact, they can be extremely **complicated** - for example, they might contain multiple branches to account for alternative courses of action. However, unlike machine learning systems, rule-based systems are not **complex**: they are never greater than the sum of their parts.

"Non-deterministic" means that given the same input, the system will *not* always return the same output. With non-deterministic systems, we have a **non-linear mapping** of inputs to outputs.

```
# Simple example of non-deterministic decision-making:
# whenever A is true, B is assinged one of the possible even values between 0
# and n

if condition A is True:
    then set variable B = (choose random even number from range (0, n))
```

The period from 1987- 1994 is often termed an "AI winter", in which the evident limitations of deterministic "if-then" systems caused a slowdown in progress. In the late 1990s, progress picked up, and by the late 2000s massive advancements were achieved due both to the development of new training algorithms and the increasing availability of extremely large datasets on which to train. Non-deterministic systems were at the center of the deep learning revolution. We will discuss some of these algorithms and related developments later on.

The non-deterministic nature of machine learning leads, however, to a number of problems for implementing this technology in practice. Among others, these problems concern **transparency**, **bias**, **predictability**, and **safety**. We will address these problems as key themes throughout the course.

2. Applications and Risks of AI

Applications

AI has applications across many domains, from games to financial markets to strategy. Here, we briefly review applications of AI in financial markets, historically one of the richest domains of AI implementation.

AI in Financial Markets

Here is a short timeline of AI in financial markets:

- 1970s: rule-based algorithms start to execute trades
- 1975: creation of the first index fund, using rules to track the components of financial market indices
- 1990s: creation of exchange-traded funds (ETFs), which use rules to automate specific investment strategies, and of quantitative funds, which drive the use of advanced algorithms in financial markets
- 2010s: increasing focus on machine learning within quantitative funds, leading some funds to switch from hypothesis-driven to data-driven investment approaches

Within this short history, we can differentiate between four different types of agency in financial markets:

Type	Primary function	Primary product
Human	Create strategies	Mutual funds
Rules-based	Execute trades, or mimic human strategies and execute trades	Index funds
Human plus algorithm	Algorithms perform data analysis, humans select trades	Quant funds
Machine learning	Create and execute strategies	Quant funds

Table 3: Overview of differences types of agency in financial markets

Each type of agency comes with its own patterns of decision-making; for example, rule-based automated execution of trades is faster than human execution but bound to human-identified processes, and machine learning strategies may outperform some humans but lead to the creation of strategies that are opaque to humans. We will return to the associated risks later on. First, we explore the impact of algorithms on financial markets.

To understand the impact of AI on financial markets, we can, as a proxy, look at the allocation of U.S. public equity assets based on 2019 data:

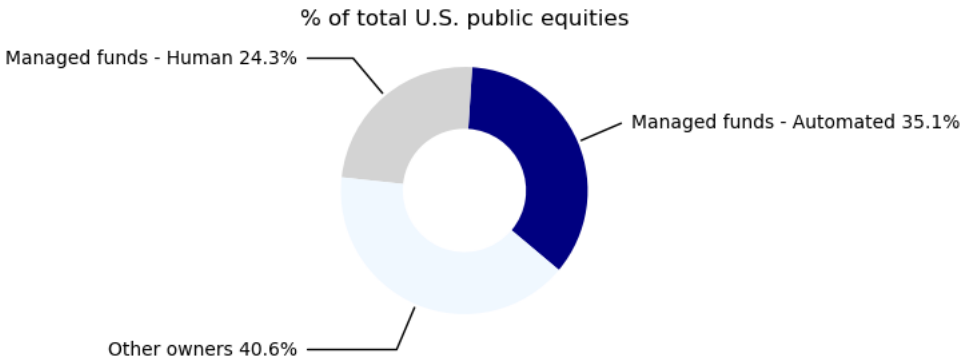

Figure 1: % of U.S. Public Equity Asset Holdings worth USD 31 trillion, adapted from [4]

Figure 1 shows that with 35.1% percent of holdings, automated funds are the dominant management solution within the U.S. public equity market, exceeding the holdings of human-managed funds by more than 10%.

Next, we want to understand the balance between rule-based strategies and machine learning strategies within automated funds:

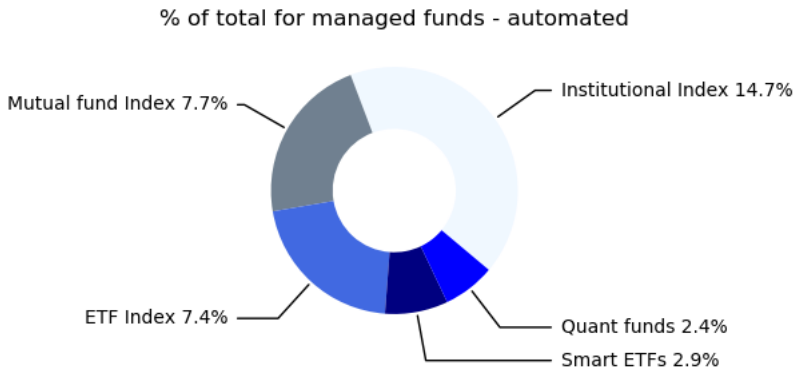

Figure 2: % of U.S. Public Equity Asset Holdings worth USD 31 trillion, adopted from [4]

Figure 2 shows that of the total U.S. public equity asset holdings, only 2.4% are in quant funds, so only a maximum of 2.4% would have exposure to machine learning-based decision-making. Given the interconnectedness of financial markets, this means that risks related to machine learning still matter. However, we also need to consider risks that come from rule-based decision-making.

Risks

The risks associated with AI are manifold. A separate field, called AI safety [5, 6, 7], is dedicated to managing these risks. Here, we look at a small subset of risks, on which we will expand throughout the course.

Machine Learning-Specific Risks

Note that all of these risks are human-facing.

- **Overconfidence and mistrust:** Results discovered through machine learning might be spurious [8], meaning that they are based on an artificial correlation between unrelated covariates, especially in high-dimensional datasets. It can take time for the spuriousness to become apparent, or it could even go undetected if not properly audited; hence, there is a risk of initial human overconfidence when working with relevant algorithms. At the same time, there might be persistent mistrust in such algorithms among key stakeholders even if results turn out to be non-spurious. This is often where transparency and explainability become paramount concerns.
- **Lack of data oversight:** Large datasets are required to train machine learning algorithms, but simply "having big data" isn't enough. There are numerous risks associated with lack of human data oversight. First, even if datasets appear reasonably large enough, their size might not truly be sufficient to completely train an algorithm, which can result in suboptimal performance. This is particularly the case if a dataset contains many observations of one feature or variable but very few of another, e.g. normal credit card acivity vs. suspicious credit card activity. Second, training an algorithm on a dataset that is too large for the algorithm might result in the algorithm overfitting, i.e. learning oversimplified behavior that will not map to new, real-word data. Third, poorly selected datasets can lead to bias and "garbage in, garbage out" scenarios, such as when a system is designed to simply use the "best" correlation among uncorrelated features. Fourth, adversarial examples [9], i.e. data known and selected to distort system outputs, may be injected into the training data leading to corrupted performance of the algorithm. This is a particular challenge in natural language processing, including in chatbots.

- **Lack of continous adaptation:** Environments like financial markets are in constant flux and humans must constantly keep learning to succeed across these changes. Once an algorithm has been trained and deployed, it might not take long before the training dataset has become outdated and the algorithms performance degraded. On the other hand, allowing the algorithm to learn continously throughout its deployment as well brings back the challenges of human oversight over the training data, compounded with the difficulty of management now being real-time rather than in a test environment.
- **Lack of transparency and explainability:** Advanced machine learning algorithms perform complex optimization processes to achive their objectives. These processes are often opaque to humans and don't come with meaningful explanations attached. Although the algorithm might make the "right" decision, the humans who are working alongside the algorithm might not be able to make sense out of this decision, and consequently might remain incapable of developing appropriate responses on their end. This can lead to a joint performance failure.

  A prominent example of a lack of transparency and explainability comes from the world of the strategy game Go. In match 2 against Lee Sedol, the 9 dan-ranked Go player (one of the world's top-ranked), DeepMind's machine learning-based Go-bot AlphaGo made its famous move 37 that baffled professional commentators, forced Lee to leave the room, and secured AlphaGo its second win.

  The live-feed from the commentators [10] provides detailed insights into how humans might struggle to make sense of a decision made by a machine learning algorithm:

  > That's a very suprising move. I thought that was a mistake. I thought it was click miss. Exactly, if we were in online Go, we would call it a 'clicko'.

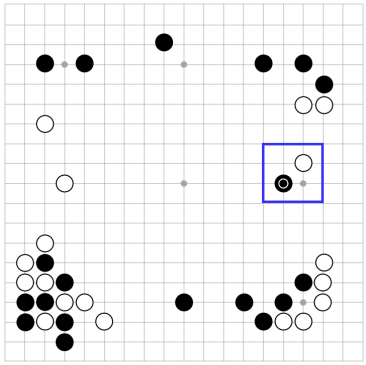

Figure 3: In Match 2 against Lee Sedol (white), AlphaGo (black) makes its game-changing move 37

  First, the commentators are convinced that AlphaGo's move must have been a mistake or technical error. Being unable to make sense out of the decision, they start to doubt their environment, which brings back the challenges of mistrust mentioned earlier.

  Next, the commentators compare the decision to the alternatives they considered and understand.

  > Yeah, it's a very strange move. Something like this [changes black's position on the board] would be a more normal move and then this [moves white's position on the board] is how white would respond.

  Finally, after seeing the ambigious response of another human impacted by this decision, the commentators resign themselves to making a hedged statement and emphasizing that they would need more time and information about how the game will proceed to make a more definite statement.

  > Lee has left the room. He left the room after this move. Just to recover from this move. It's a very suprising move. I don't know whether it's a good or a bad move at this point.

How can we manage these risks? From managing overconfidence to managing a lack of transparency, we have seen that building a response to risks associated with machine-learning takes time. However, in real-time environments, time is often highly constrained.

Ecosystem-Specific Risks

In this section, we briefly look at a risk associated with rule-based decision-making: the fast and automated execution of decisions.

The decision-making of machine learners might take place on top of an infrastructure of fast rule-based execution, eliminating the time humans would need to meaningfully respond to challenges like mitigating the opacity of a move 37. This makes fast rule-based execution, paired with human stakeholders' comparatively slow response time and heterogenous information, an ecosystem-specific risk for machine learning-based decision-making.

- **Complexity and speed**: Software development processes are complex. They often involve multiple versions of a program existing in the same codebase. The goal of these processes is often speed. Engineers build a program that, among other things, works faster compared to other programs. The risks from development complexity and speed of execution came together in a 2012 stock trading disruption at Knight Capital Group that forced the financial services firm to shed 70.60% of its market value of USD 1.5bn over two days for an error that was discovered and fixed *in less than an hour*.

Figure 4: Initial drop in the stock price of Knight Capital Group following a trading disruption on August 1, 2012

A detailed account of this event of how Knight enabled the fatal trading disruption through careless changes to its software is given in [11]. Here, we represent the changes to Knight's case in four steps across Figures 5 and 6. Blue lines represent active connections between the componets of Knight's trading program, grey lines represent inactive connections.

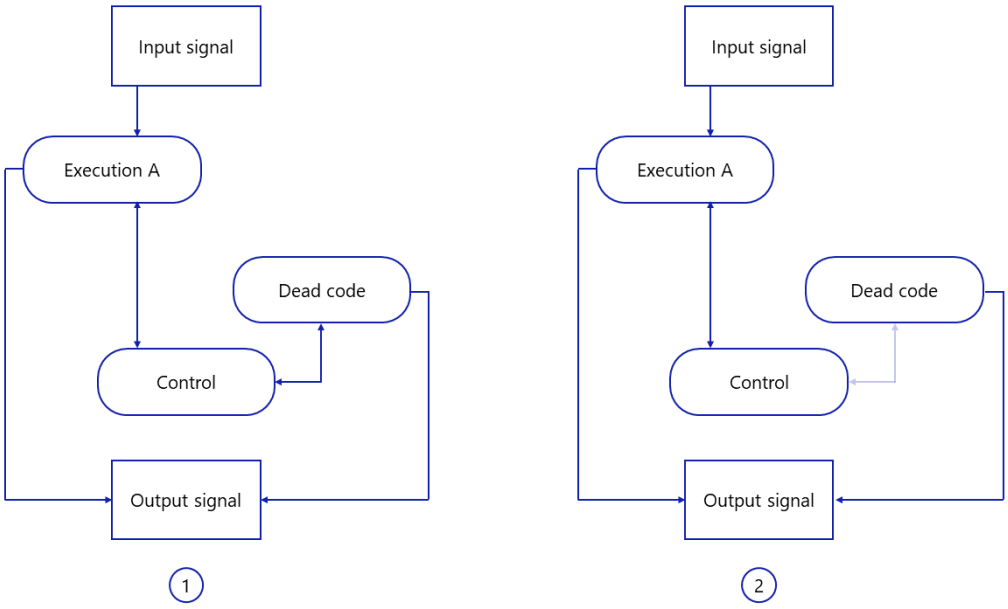

Figure 5: Step 1 and 2 of the changes to Knight's codebase

- **Step 1:** Knight has a functioning trading system that received orders, processes their execution, controlls this execution, and sends the processed orders to markets. Through development bad practice, an old algorithm build for experimental purposes remains part of the codebase of the trading system, although it cannot be activated under the current system specification. The algorithm is designed to *buy high* and *sell low*. Although the algorithm currently sits as "dead code" in the codebase, it retains the functionality to send orders to markets.
- **Step 2:** Knight makes updates to the control component in its codebase. These updates bring improvements but through further developement bad practice detach the control from the experimental algorithm.

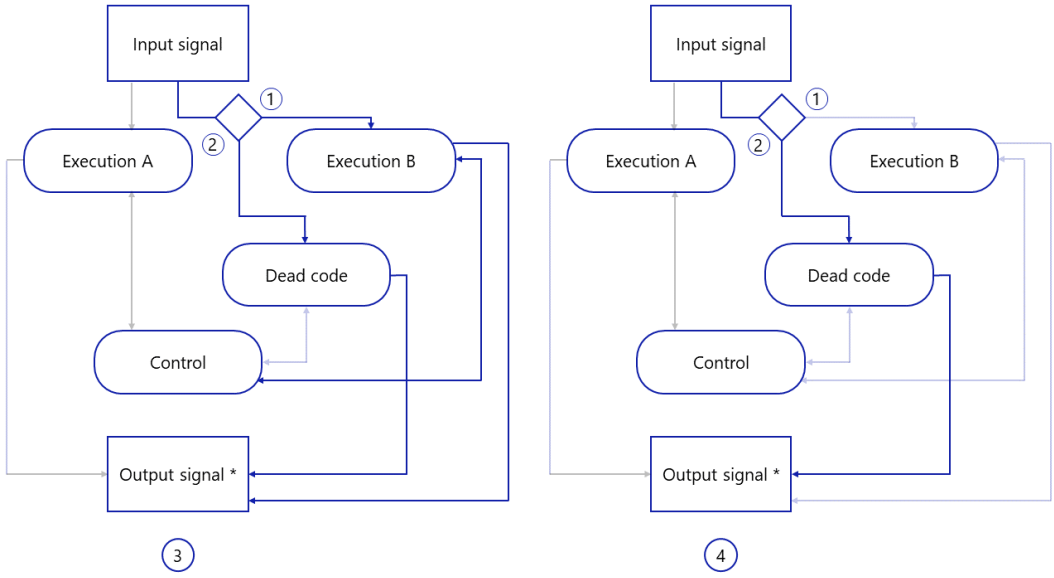

Figure 6: Steps 3 and 4 of the change's to Knight's codebase

- **Step 3:** To make its system compatible with a new market, Knight builds a new execution component for its system. These changes are made under severe time pressure and introduce a critical development flaw. The code for the new execution component repurposes the activation code of the experimental algorithm. As a result, the experimental algorithm can now receive orders. It does not receive these orders because the new execution system is till higher up in the hierachy of the program, so all the input signals are fed into the execution component.

- **Step 4:** The updates to Knight's system go live on eight servers and Knight starts receiving orders. However, under time pressure, one of Knight's developers has forgotten to update one of the eight servers correctly. Hence, this server does not have access to the new execution component. On this server, the input signal, the orders that Knight receives from its clients, are now fed into the experimental algorithm, which starts to *buy high* and *sell low* without being checked by any controls. After less than an hour developers locate the error and shut the server down. However, they made the fatal flaw of reverting to the old code on all eight servers and get the complete system live again. As it turned out, the experimental algorithm was now *activated on all eight servers*. The cumulative losses were irreversible and ended up reducing Knight's stock price by 70%.

The story of Knight capital shows the multiple risks that arise from different patterns of decision-making. In this case, the rapid execution of an algorithm paired with the adaptable but sometimes inconsistent decision-making of human developers. In the next section, we'll look more at human decision-making from the perspective of human situational learning.

As a last note, reading through this section and looking at Figures 5 and 6, you have familiarized yourself with two important concepts, those of an algorithm and a program:

- An **algorithm** is "a group of components talking to each other in sequence", where each component can be a simple instruction like "when A is True, then do B" or a more sophisticated function like "do convex optimization for A"
- A **program** is "a group of components talking to each other in sequence", but one where each component is an algorithm; the parts of a program are algorithms

3. Human Learning and Decision-Making

We discuss the case study for this session "Learning in a Counterinsurgency Team".

Jason Amerine:

- *Learning from instruction:*
  - Amerine mentions the importance of instructors during training

> You get out of these courses and sometimes you have instructors that take what they teach very seriously and other times you don't.

- *Rules and applying them to new environments:*
  - Amerine mentions that the transition from training to deployment is linear

> I found was that every major lesson I have learned throughout my career, whether it was in the Q Course [Army Special Forces Qualification Course] or Ranger School, I mean, everything that I was taught in the school house, I applied over there

> All the major muscle movements during the campaign we really had been taught

- *Exception handling and continous learning:*
  - Amerine mentions one core exception to the linear transition from training to deployment and how it was solved through generating new rules of behavior

> You had certain unique areas, when we talk about the Horse Soldiers [...] I mean that was something you couldn't have foreseen and litreally was an act of God that we had the right officer there who could teach his people how to ride

Mark Nutsch:

- *Rules and applying them to new environments:*
    - Nutsch mentions that the transition from training to deployment is linear but focuses on process over instructors

> But even in that new situation, the guys kept going, 'Hey, we have been here before, remember Special Forces training, remember Robin's Sage at this phase of insurgency, you know as that would progress, remember that.'

- *Adaptation and learning how to learn:*
    - Nutsch describes training as learning how to learn, so the team can adapt fast to changes in the environment and its behavior reflects new information

> But the sergeants and I, coming back as we're talking about this, we did the things you do in training. Each day we would do lessons learned, an internal AAR [After Action Report], whether it was five minutes or fifteen minutes, sit down and go 'Damn, what nearly killed us today? How do we make sure that doesn't happen again? You know, how do we survive the next hour? And how do we win?'

> I would have to say, even in our mission, we were the students

- *Breadth of relevant training:*
    - Nutsch mentions that the relevant training for him extended across his entire life and career

> You relied on that training that you had, the leadership lessons, people, mentors, that talked to you, every aspect of my career up to that point, to include character building events I had as a teenager through high school and college, all of that came to that focal point in my life on that battlefield

- Pattern matching and learning with imperfectly labeled data:
    - Nutsch describes the process of matching patterns based on a different local information system

> They couldn't read a map but they could describe to you passionately 'It's this village, don't you understand? It's this village right over here. It's this guy, he's the one we're after.'

Core themes:

- Application of rules vs adaptation
- Learning from a supervising instructor vs learning from interaction with the environment
- Learning through guidance vs learning through trial-and-error
- Transfer of knowledge across domains vs learning how to learn

Differences between Amerine and Nutsch:

- Amerine emphasizes the value of having knowledge that generalizes across environments
- Nutsch emphasizes the value of learning how to learn in training, so that even without knowledge about a new environment, fast adaptation is possible
- Amerine emphasizes instructor-led learning
- Nutsch emphasizes process-driven learning

4. Types of Machine Learning

In this section, we give an extremely short overview of different types of machine learning, which will expand on in detail in the next sessions.

- **Supervised learning**
    - Supervised learning is learning from labeled data, where the labels are provided to the algorithm by a human and the algorithm learns to apply these labels to new data, e.g. for *prediction*
- **Unsupervised learning**
    - Unsupervised learning is learning from data that does not have labeles for the purpose of learning features of the data that can be used to group or cluster the data, e.g. for *pattern detection*
- **Reinforcement learning**
    - Reinforcement learning is about learning through trial-and-error from interaction with an environment what the best sequence of actions is to achieve a goal, e.g. for *autonomous control*
- **Meta- and transfer learning**
    - Meta- and transfer learning are at the forefront of current AI research and concern how algorithms can learn how to learn and quickly adapt previously learned behavior to new environments

5. Learning and Optimality

If we're searching for one unified expression of how the different types of machine learning work, it would be mathematical optimization. Here, we give a brief overview of optimization in a mathematical sense and then focus on how optimization interacts with policy.

In simple mathematical terms, when we learn a task, our goal is to **minimize the rate of errors**. Or, we can state that our goal is to **maximize the rate of success**.

Both minimizing error and maximizing success sound intuitive but there's some other elements we need to consider to understand optimization.

The main element is that we should understand minimization and maximization in a spacial sense. Optimization is a **search** across space (the space of possible decisions) for the minimum or maximum of this space. For an example of such a **search space**, see Figure 7 below.

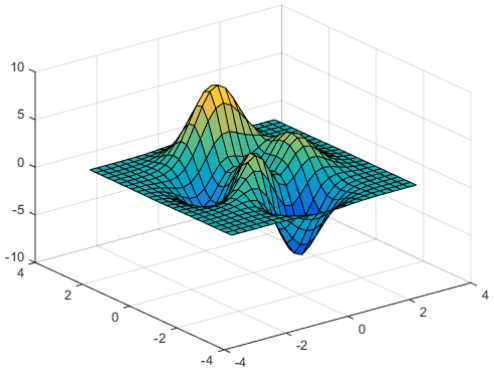

Figure 7: Search space of an optimization problem with local and global maxima

When we start to learn, we start at a random point in this space and we explore the space over time to find its respective mimum or maximum.

In this search, we need to differentiate between **local optima** and **global optima**. Local optima are optimal points that are optimal only for a region of the space. Global optima are optimal points for the entire space. Sometimes as we explore the space, the learner may get stuck in a local optimum. To solve this, we can add **constraints** that reduce the search space and faciliate the learner's search.

When we talk about constrains, we need to differenatiate betwee two types of constraints:

- contraints that change *whether* the algorithm can solve the problem, e.g. how much computational ressources are available
- constraints that change *how* the algortihm solves the problem, e.g. removing certain optimal or non-optimal decisions from the search space

When we say that we "add constraints", we care about constraints that change *how* the problem gets solved.

Note, that these constraints also bring aspects of policy and strategy into the calculus of optimization. The reason for this is simple:

- making the "right" decision requires a consistent definition of what constitutes and error or success for a selected task

Why do we need to be consistent in stating what we mean by error or success?

We need to be consistent in defining what we want to optimize because if we don't guide the learner in a consistent way, it will find all kinds of unexpected solutions within the wide space that it searches on its own. Some of these solutions might run counter to what we wanted the algorithm to learn in the first place.

We can describe the tradeoffs around the target specification for optimization with the following terms, taken from [12]:

- **ideal specification** desribes the "wishes" of the system designers that correspond to the hypothetical description of an ideal system
- **design specification** describes the "blueprint" corresponding to the specificatoin that the designers *actually use* to build the system
- **revealed specification** describes the "behavior" corresponding to the specification of *what actually happens*

To understand the challenges of consistent target specification, consider the following example:

- In a computer game-version of a boat race, multiple bonus packages are placed across the racetrack from start to end. You want to train an algorithm that learns how to win the race. To achieve this objective, you tell the algorithm "maximize bonus points", assuming that this maximization will lead to the fastest traversal of the racetrack from start to end.

Figure 8 shows what happened when researchers implemented the above logic for the game *CoastRunners* [12]. Best viewed as an animation.

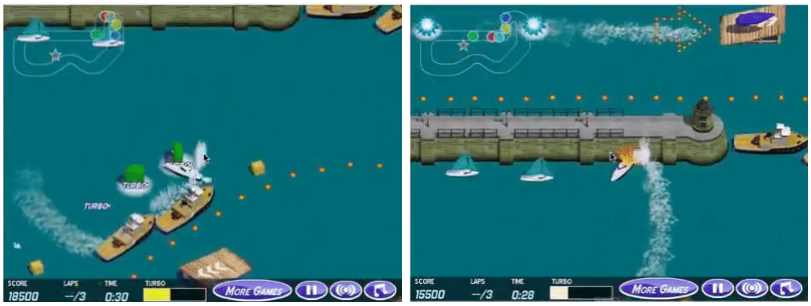

Figure 8: Faulty optimization in the CoastRunners game

Instead of finishing the race, the machine learner found a laguna in which "turbo boosts" are spawned in regular intervals. The learner discovers that circling through these boosts yields a higher bonus score than simply following the racetrack from start to end. To time the respawn of the "turbo boosts", the learner circles through the lagoon, in the process crashing into other boats and parts of the environment. The algorithm never completes the race but has a substantially higher score than the boats that finish the race. Thus, the revealed specification radically differs from the design and ideal specification of the algorithm.

To ensure that the optimization is consistent and aligned with human values, generalist and domain experts can provide invaluable guidance to the specialists who develop the relevant algorithms, especially complex real-world environments.

What does all of this tell us about "good" decisions? We can state that, while all "good" decisions are optimized, not all optimized decisions are "good"

References

[1] Steward Russel and Peter Norvig. 2019. Artificial Intelligence: A Modern Approach. Available at: http://aima.cs.berkeley.edu/

[2] Selmer Bringsjord and Naveen Sundar Govindarajulu. 2018. Artificial Intelligence. Available at: https://plato.stanford.edu/entries/artificial-intelligence/

[3] Stewart Russel. 2015. Rationality and Intelligence: A Brief Update. Available at: https://people.eecs.berkeley.edu/~russell/papers/ptai13-intelligence.pdf

[4] The Economist. 2019. The stockmarket is now run by computers, algorithms and passive managers. Available at: https://www.economist.com/briefing/2019/10/05/the-stockmarket-is-now-run-by-computers-algorithms-and-passive-managers

[5] Jan Leike et al. 2017. AI Safety Gridworlds. Available at: https://arxiv.org/abs/1711.09883

[6] Paul Christiano and Greg Brockman. 2016. Concrete AI Safety Problems. Available at: https://openai.com/blog/concrete-ai-safety-problems/

[7] DeepMind's AI Safety Team, see: https://deepmind.com/safety-and-ethics

[8] Jianquing Fan and Wen-Xin Zhou. 2016. Guarding Against Spurious Discoveries in High Dimensions. Available at: http://www.jmlr.org/papers/volume17/16-068/16-068.pdf

[9] Ian Goodfellow et al. 2017. Attacking Machine Learning with Adversarial Examples. Available at: https://openai.com/blog/adversarial-example-research/

[10] Lee Sedol vs Alpha Go. 2016. Video available at: https://www.youtube.com/watch?v=JNrXgpSEEIE

[11] Bishr Tabbaa. 2018 The Rise and Fall of Knight Capital - Buy High, Sell Low. Rinse and Repeat. Available at: https://medium.com/dataseries/the-rise-and-fall-of-knight-capital-buy-high-sell-low-rinse-and-repeat-ae17fae780f6

[12] Pedro Ortega et al. 2018. Building safe artificial intelligence: specification, robustness and assurance. Available at: https://medium.com/@deepmindsafetyresearch/building-safe-artificial-intelligence-52f5f75058f1