Intelligent Systems Summary (Final)
Greg B
2020

Intro

Hi all. This is a lightning summary of our Int Sys course this year. Each lecture of the course is a subsection. I hope this is useful for people. I have tried to make it so that most of the bullets can be read across lines (as if they were full sentences).

I hope this helps as a handy reference during the exam session. I tried to make them as readable as possible, with the goal being that you could read through these notes and come away with understanding most of the major concepts that were covered in the course.

Notes in Blue are Important. Notes in green are my own additions. Memes are there to fill blank space and make the doc a little more cheery.

After thinking a while, there is limited benefit in doing this for the more mathy-lectures of Multi-Agent RL, so these (Lectures 6,7) are not included.

Lecture 1 & 2: Intelligence vs Autonomy + Agent Oriented Design

Intelligence is quite hard to define. Attempts have been made, some more successful than others:

Defining Intelligence (some):

- Intelligence is the ability to deal with complexity
- Intelligence is Goal-Directed adaptive behavior
- Intelligence is a force, F, that acts to maximize future freedom of action

Remember also that there are different forms of intelligence: One can have social intelligence (the intelligence exhibited by a skilled salesperson), emotional intelligence (a therapist or a mentor), mental intelligence (a mathematician), etc.

The indirect goal of AI is to take these indirect, everyday notions and to build a computational model of them. This noble goal has resulted in a multidisciplinary field which, at its core, concerns itself with the design, analysis and application of computer-based systems which at least *deserve to be said* to be intelligent.

Note also that there is a (larger and more interesting) *philosophical* aspect of intelligence. Intelligence is the thing that makes us who we are. Without a proper grasp of intelligence, we will never know what's *really* going on in the universe. The tragedy of the situation is that we as human beings are introspective enough to know that we have it, and to use it everyday, but we are not introspective enough to actually know how it works or why it's here. If we did, we could just build it directly.

This is a recurring theme that lurks throughout this course that I emphasize in these notes.

The Turing Test:

- Is an attempt at a better definition of intelligence, invented by Alan Turing.
- Turing's definition is an "operational" one (as opposed to a conceptual one): He defined intelligence on the basis of behavior that is indistinguishable from that of a human's.
- Turing's idea was this: A computer program is intelligent if it answers (written) questions asked by a person in such a way that it seems like the answers could only be generated by another human being. If the interviewer is fooled, the test is passed.

Since the Dartmouth summer school on AI, there have been different paradigms that people have followed when trying to build intelligence. They are the following:

Knowledge Based AI (1956 -)
Guiding Model: A human being
Guiding Assumptions:

- The belief that Intelligence *is* (or eventually boils down to) knowledge representation
- A von Neumann computer is a perfect model of the human "cognitive apparat"

Key terms:
Symbol System Hypothesis: The ability to produce and manipulate symbols is a necessary and sufficient condition for intelligence (Essentially, intelligence is a symbol system, with symbols representing the state of all our current knowledge and beliefs)
Top-down design: To build AI, we start with high-level concepts at the knowledge level and break them down into smaller, programmable units.
Major Problem: *Symbol Grounding* - How do we ensure our symbols relate to the real world?

Behavior-based AI (1985 -)
Guiding Model: Human or Animal
Guiding Assumptions:

- Intelligence is built upon elementary behavioral activities (e.g., moving along a wall, grasping an object)
- Behavior-based implies acting in the real world - thus a coupling between the sensory system and the motor system is essential
- There can be no intelligence unless it is grounded in an ability to act in the real world
- This leads to the *Physical Grounding hypothesis*

Physical Grounding Hypothesis: Rooting of symbols in the real world is a necessary condition for intelligence
(No rooting → No meaning → No intelligent behavior)

Connectionism (1950-1965, 1986 -1994, 2012 -)
Guiding model: (human) brain
Guiding Assumptions:

- Processing of information through very simple but many interconnected units (neurons) that interact at a low (in terms of signal-processing) level
- A connectionist system will be massively parallel, distributed, and process information below the level of symbols

Distributed AI (1980 -)
Guiding Assumptions:

- An intelligent system could actually be a group of interacting intelligent beings that interact on the knowledge level

Key issues: Communication, coordination, cooperation, negotiation, organizational structure.

The Agent Concept

- There is no universally accepted definition of an "agent". It is a term that has been applied differently by different people and in different contexts.
- But there is an Emerging Standard View:

"An agent is a (computational) entity that is situated in some environment and that is capable of flexible, autonomous activity -- that is, capable of action and interaction -- in order to meet it's design objectives."

Examples of Agents:

(1)

- A watch (not even a smart one)
  - It interacts with environment
  - It moves autonomously
  - It meets its design objectives

(2)

- Reinforcement Learning
  - Interacts with environment
  - Motivated to move by reward function
  - Learns to maximize reward (in effect meeting its design objective)

(3)

- Alexa / Siri
  - Autonomously offers advice
  - Adapts to user preference

Characterising Intelligence is hard
So many characteristics of agency that are claimed to be essential by different people…one could say…"Rationality" (an intelligent agent must be rational). Okay. Fair enough. But what about...Mobility? Introspectiveness? Benevolence? Must an intelligent agent be benevolent? Must it be kind? Surely it must have desires, or preferences...But what about beliefs, knowledge, intentions, plans? There are so many things one could think of.

How about the reverse direction -- What if we simply defined a minimal definition of intelligence? That way, we can at least say that these criteria MUST be met. This creates a useful lower boundary and in principle, an exclusion criteria.

A Minimally Intelligent Agent

To be considered intelligent, an agent must:

- Be Proactive:
  - It takes the initiative to satisfy it's goals (design objectives)
- Be Reactive:
  - It can at least perceive and respond to its environment, and not just be inert.
- Have Social Ability:
  - It must be capable of interacting with other agents and humans
  - An intelligent agent with social ability could for example cooperate with others, negotiate, and make joint plans. Not to mention the darker side of social ability: Fighting against others, competing against them, protecting ones own interest, etc. These also form part of social ability!

Our first tradeoff (Goal-Directedness vs. Reactivity):

- Consider the act of designing a minimally intelligent agent yourself -- notice the tradeoff you would have to make in your design decisions. It must be proactive (i.e: goal-directed) but it must **also** be reactive. These are two forces which can be in opposition to one another!
- Building a Goal-Directed system can be simple
  - E.g: A functional system that is externally activated
- Building a Reactive system can be simple, too:
  - Design a bug that always flies towards the light
- BUT: Balancing goal-directedness with reactivity can be VERY COMPLEX (!!). This is because you must **react to new situations** while at the same time **focus on getting things done**. Not trivial to do, even for humans!
- This is a fundamental thing that we will return to when discussing commitments and conventions.

Fun quote:

"You can never globally optimize your life… you can only use local regret minimization."

Agent vs. Object:

The traditional view of an object and our view of an agent have at least three distinctions:

- agents embody a stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent;
- agents are capable of flexible (reactive, pro-active, social) behaviour, and the standard object model has nothing to say about such types of behaviour;
- a multi-agent system is **inherently multi-threaded,** in that each agent is assumed to have at least one thread of control.

Types of Agent Environments

- Accessible vs inaccessible.
  - Agent can obtain complete, accurate, up-to-date information about the environment's state
- Deterministic vs non-deterministic.
  - How much uncertainty there is regarding the next state given an action
- Episodic vs non-episodic.
  - Episodic: Performance depends on a number of discrete episodes
- Static vs dynamic
  - Is the Agent's influence on the world, the only way the world can change? Then it's static. The Physical world is highly dynamic.
- Discrete vs continuous
  - Refers to the action space: Chess is discrete, taxi-driving is continuous (Russell and Norvig's example)

Agent Architectures

Logic Based Agents

- Symbolic / Knowledge based AI view
- Intelligent behavior is a result of symbolic representation of the environment, combined with logical deduction (theorem proving)
- *Theory of Agency*: ?
- *Belief database:* Information the agent has about the environment (the facts it has access to)
- Action Determination:

```
1.  function action(Δ, ρ) returns an action
2.  begin
3.      for each a ∈ A do
4.          if Δ ⊢ρ Do(a) then
5.              return a
6.          end-if
7.      end-for
8.      for each a ∈ A do
9.          if Δ ⊬ρ ¬Do(a) then
10.             return a
11.         end-if
12.     end-for
13.     return null
14. end function action
```

Implied action

Allowed action

No action

Action Rules!

$$\varphi(\ldots) \longrightarrow \psi(\ldots)$$

"If you can prove this then do this"

Problems with Logic Based Agents:

- Symbol Grounding: How to couple perception with purely symbolic facts
- Reasoning takes (too much time)
    - Reasoning with temporal information in a time efficient way is DAMN HARD.
- 

- Reasoning requires a sophisticated world model:
  Humans develop a mental model of the world based on what they are able to perceive with their limited senses. The decisions and actions we make are **based on this internal model.**

  To handle the vast amount of information that flows through our daily lives, our brain learns an abstract representation of both spatial and temporal aspects of this information.

  Simulating this in a machine is very, very complicated (see https://arxiv.org/pdf/1803.10122.pdf)

Reactive

- Intelligence emerges from the interaction between simple behaviors and the environment
- Intelligence cannot be separated from acting in the real world: Intelligence cannot be disembodied

Subsumption Architecture:

- Decision making is established through a set of behaviors. But what if multiple hardwired behaviors choose confliction actions? One needs to subsume the other.

Definition of *subsume*

transitive verb

: to include or place within something larger or more comprehensive : encompass as a subordinate or component element

// red, green, and yellow are *subsumed* under the term "color"

Example of Subsumption in real life:
You expect an autonomous car to do the following things for you:

- Get you from A to B
- Avoid accidents
- Not consume too much fuel

Now you are driving from A to B, but there is a bicycle lying in the road, and going around it would increase your fuel consumption. You could

probably drive over it, but most people would prefer to subsume goal 3 (not consume too much fuel) into goal 2 (avoid accidents, damaging the car).

Advantages of this subsumption architecture:

- Simple
- Computationally tractable (this just means, it's physically feasible/practical within the current state-of-the-art of computing)
- Robust against failure

Problems of this subsumption architecture
Belief / Desire / Intention

1. Only uses local info -- there is no model of the environment
2. Thus it can't look far into the distance (this requires prediction, which in turn requires a model of the world)
    a. "In the real world, prediction is the essence of intelligence" - Yann LeCun
3. Emergence can be hard to predict
4. Dynamics of interactions between all these different behaviors can become too complex to design

Belief - Desire - Intention
BDI Agents emphasize practical reasoning:

1. They **deliberate**: What goals am I really trying to achieve here?
2. They use **means-end** reasoning: How will I achieve that goal? Let me work backwards from there

Set of Intentions:

- Lead to actions, by driving reasoning
- Constrains future deliberation (James Bond isn't deliberating too much on anything that doesn't affect a mission, for example)
- Intentions persist until the goal is achieved. OR the goal is believed to be unachievable (or achieving the goal would be pointless)

Notice the tradeoff that exists here!
We need to **accomplish our intentions** through acting directly on them...but **at the same time** we need to be sure that fulfilling the intention is still actually something we want, something that is useful, and something that is actually possible. Otherwise why bother? So we need to also constantly **reconsider our intentions**.

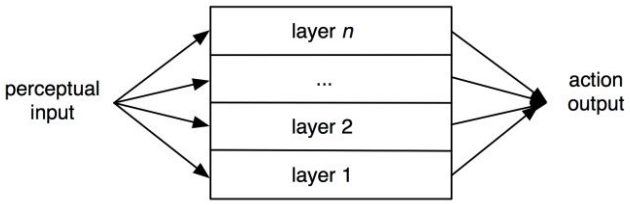
Bold vs. Cautious

Bold agents never stop to reconsider
Cautious agents constantly stop to reconsider.

See final slides of week 1 for BDI Components

Layered Architecture
Our need for both pro-active and reactive behavior naturally leads to multiple interacting layers of hierarchically organized behaviors

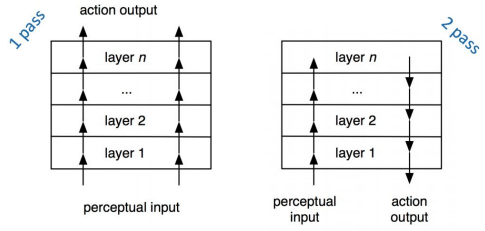
Horizontal Layered Architecture
Number of desired behaviors = Number of layers



Might need a function to mediate the interaction between these layers if their actions contradict (this is comparable to subsumption)

Central control might create bottleneck if complex. If you have n layers, and they can each propose m actions, that leaves you with m^n possible joint actions. Can become very complex very quickly.

Vertically Layered Architectures



Lecture 3: Genetic Algorithms

Definition of Evolutionary Computation

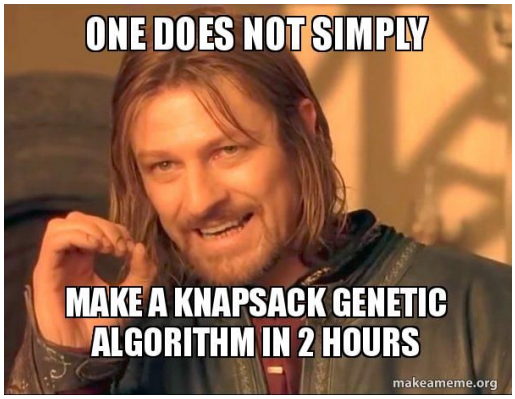
"Evolutionary computation is a way to solve an optimization problem using a population of candidate solutions and the dynamics of genetic evolution. These include selection, reproduction and mutation."

- It is an **any-time** algorithm: That just means it can return a valid solution to a problem even if it is interrupted before it ends. The expectation though, is that if you let it run longer, it should keep finding better and better solutions (until an optimal one is found, or it converges. Hopefully they are the same thing!)
- It is an **iterative** process
- It is **stochastic** (random, cannot be predicted)

The steps of your typical GA (Genetic Algorithm)
[Use this as reference. Keep coming back to it]

```
1:  generate Pop with n chromosomes                          } Construct initial
2:  for ind ∈ Pop do                                         } population
3:      INITIALIZE(ind)
4:  repeat
5:      Pop_mat ← ∅                                          }
6:      repeat                                               } Fill mating pool
7:          ind ← SELECT(Pop)                                } with selected
8:          add ind to Pop_mat                               } individuals
9:      until |Pop_mat| = |Pop|                              }
10:     Pop ← ∅                                              }
11:     repeat                                               }
12:         randomly choose and remove two                   }
              individuals p1, p2 from Pop_mat                }
13:         with probability Pr_c do                         } Generate new
              {c1, c2} ← CROSSOVER(p1, p2)                   } population from
14:         otherwise: c1 ← p1; c2 ← p2                      } mating pool
15:         for child ∈ {c1, c2} do                          }
16:             for each gene g in child do                  }
17:                 with probability Pr_m do MUTATE(child, g)}
18:             add child to Pop                             }
19:     until |Pop_mat| = 0                                  }
20: until TERMINATE(Pop)
21: report best individual in Pop
```

Algorithm 1: The standard genetic algorithm.



Key terms:

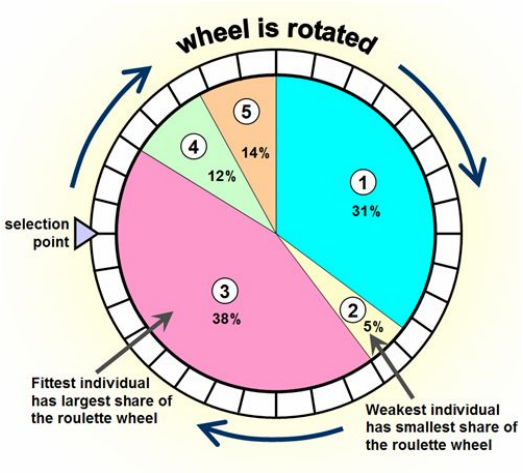
- *Candidate Solutions:* A randomly generated set of possible solutions to your optimization problem.
  - Needs to be diverse (to feasibly branch into different directions)
  - Needs to be large (A requirement for it being diverse and because mutation is a low-probability operator)
  - This population acts as the basis for better solutions and evolves through the generations

- *Genotype:* The form of the candidate solutions that encode actual solutions to the problem.
  - The most simple form would be a string of bits <1,0,1,1,0,...,0,0,1>
  - More complex forms would be trees, graphs, vectors of real numbers

- *Phenotype*: The actual solution for the problem that can be tested for performance.
  - If you aren't just looking for some optimal bit-string, then going from Genotype → Phenotype requires non-trivial translation.

- *Fitness:* The criterion to be optimized
  - Evaluating fitness is not easy and can be a fairly complex thing

- *Roulette Wheel* selection:



- *Boltzmann Tuning*: We can use a temperature-parameter T to tune our selection pressure.
    - High Values of T: more randomness
    - Low Values of T: more deterministic according to fitness rank

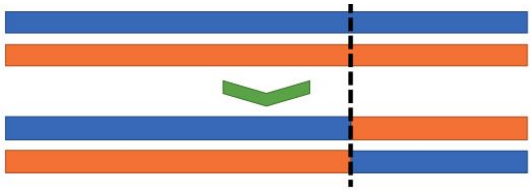
- *Tournament Selection*: Select a fixed number of individuals from the population and host a "tournament": The fittest two members of that group get to reproduce (get added to the mating pool). The rest are discarded. Keep running tournaments until you've gone through the entire population.

    - Smaller Tournaments REDUCE selection pressure. (You are competing with less people so you have a better chance to be one of the two selected)
    - Smaller Tournaments lead to MORE diversity in population. But keeping all this diversity means we need to do more rounds of selection. Thus it's slower to converge (To see this, imagine if we just held one massive tournament -- we would converge instantly, but wouldn't be doing much evolving).

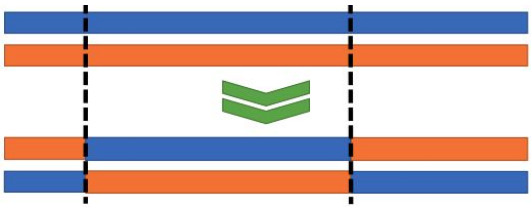
- Tuning Tournament Selection: We can make our outcomes stochastic:

  For Tournament of size **k:**
  Choose best individual with prob **p**
  Choose 2nd individual with prob **p*(1-p)**
  Choose 3rd individual with prob **p* ((1-p)^2)**
- *Reproduction*: Two parents share their genetic code and generate offspring

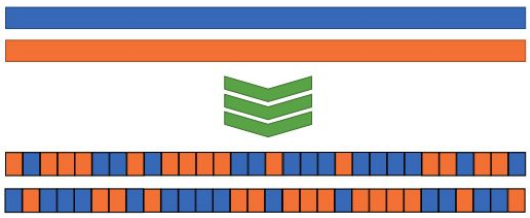
Single Point Crossover
(the blue and orange flips)



Double Point Crossover



Uniform Crossover



- *Mutation*: Random changes in the genotype
    - This is a low probability operator
    - The purpose of mutation is to search the region of solutions that are not covered by the initial population

- *Bit flipping mutation:* Random bit flipped (or **whole genome gets flipped randomly (??)** (This one is a bit weird to me. Seems like a nice way to destroy all your progress!)
- *Real value mutation*: Randomly created values are added to variables with a low probability

Key Questions: Why do some good solutions survive and get combined into even better solutions?

Schema Theory:

- What is a Schema?
- Consider the following binary genotypes:
- 0111, 1111, 1101, 0101
- All of these instances belong to the following schema: *1*1
- More generally, we can consider fixed length schemata(s) of the form:

$$s \in \{0, 1, *\}^n$$

The **order** of s: the number of fixed positions (the number of 0's and 1's)
The **defining length** of s is the *longest distance between fixed positions*.

The probability of any such schema of surviving mutation is

$$(1 - p_m)^{o(s)}$$

where p_m is the mutation probability of a single bit

The probability of a schemata surviving crossover is:

$$(1 - p_c \frac{d(s)}{n-1})$$

where p_c is the crossover probability

The **Schema Theorem** states the following:
Schemata with higher order have a lower probability of surviving mutation

Schemata of longer defining lengths have a lower probability of survival after crossover.

Thus the full Schema Theorem states that:

"Of those schemata with **above average fitness**, those with **short defining lengths** and **low order** are more likely to survive and increase their proportion through successive generations."

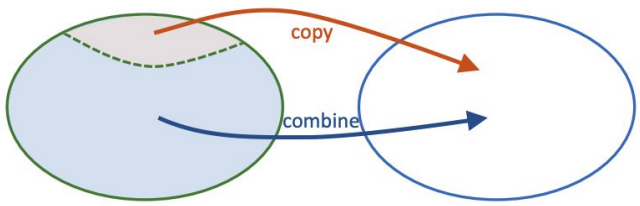
The Building Blocks Theorem:
Genetic Algorithms perform adaption by identifying and recombining so called *building blocks* which are relatively high in fitness to build entire solutions.

Wondering what these building blocks might look like? The schema theorem tells us directly: These building blocks take the form of **schemata with low order and short defining length.**

Your friend is designing a genotype. How would you use the schema theorem to help them when designing it?

- Tell them to put related genes close together so that they can **form schemas** with **low description length**
- Tell them to make each gene as independent from one another as possible
- Further, when designing a crossover operator, design it so that it has a low probability of destroying schemas!

Elitism in Genetic Algorithms:
Sometimes, we know that some solutions are good and we don't want to risk losing them when performing selection. We can accomplish this by simply copy pasting the best k-individuals to the next generation population.



Advanced Technique: Separable Natural Evolution

- I don't understand this. http://people.idsia.ch/~tom/publications/bbob-snes.pdf

Avoiding Premature Convergence

GA's can converge prematurely. One way this can happen is that a **niche solution** can draw a lot of the population towards it (but it is suboptimal).

- Too many individuals become similar
- Crossover won't result in any more exploration of the search space
- Only mutation can get the population out of this situation. It is as if we have descended into a valley. How to avoid this?

Introduce a Niche Penalty

We can avoid overproduction of similar individuals using a niche penalty:

- Any group of individuals of sufficient similarity (niche radius) can have a penalty added to their fitness evaluation.

Another way to avoid premature convergence

- monitor premature convergence and simply replace part of the population with randomly generated individuals. But when to do this? This is a design question, dependant on the task and the associated search landscape

Challenge: Building Blocks Theorem might not scale to seriously complex problems

- The building block theorem tells us that partial solutions should combine into the full solution.
    - This implies the problem is divisible into smaller subproblems which can be solved independently and combined.
    - But this is **simply not the case** for many intrinsically complex problems.

Fitness Function Requirements

- A fitness function needs to be able to separate partial solutions from non-solutions
    - There needs to be a spread of fitness values

When do GA's really not work?

GA's can't solve tasks with a binary fitness value. (It's solved or not solved, and nothing in between)

- However, if a candidate solution can be **tried many times** with *non deterministic outcomes*, then using the **percentage of success** could be a suitable fitness evaluation.

See slides on Neuro-Evolution.
See also https://arxiv.org/abs/1712.06567. They show that gradient-free GA's can perform well on hard Deep-RL problems.

Lecture 4: Expert Systems

What's an Expert System?

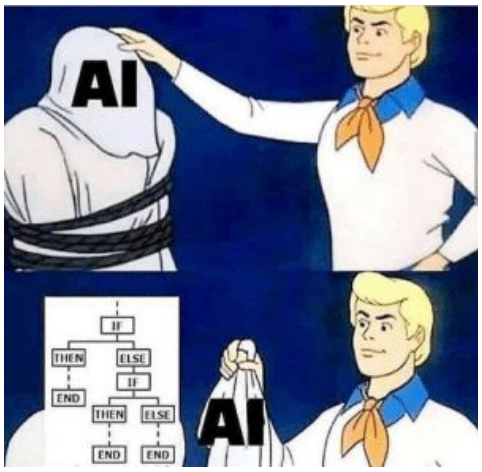
- In plain English, an expert system is a computer system that can make decisions that normally only experts can make.

In more detail:
"An expert system is a computer system that uses knowledge and inference to solve problems in specialized subject domains that are difficult enough to require human expertise."

Examples of early expert systems:

(1)

- DENDRAL
  - A program for explaining empirical data
  - Used a bunch of "if-then" rules to generate a branching tree to analyze mass spectrometry data to search for extraterrestial life



(2)

- MYCIN
  - A rule-based computer program for advising doctors about therapy selection
  - MYCIN would:
    - Identify bacteria of an infection
    - Recommend antibiotic
    - Explain it's decision
  - MYCIN was cool because it could give a confidence interval over it's prediction
  - But it didn't use Bayesian reasoning around uncertainty.

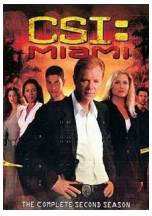
What's the difference between an expert system and an intelligent system?

- An expert system is only software. It's not embedded in the real world. So you could say it's a subcategory of intelligent systems.

What's the difference between an expert system and a decision support system?

- An expert system combines knowledge with reasoning, and **makes the decision for you** by using the information you give it.
- A decision support system helps you process data (data analytics dashboard, etc), and **helps you make your own decision**

Where can we use Expert Systems?

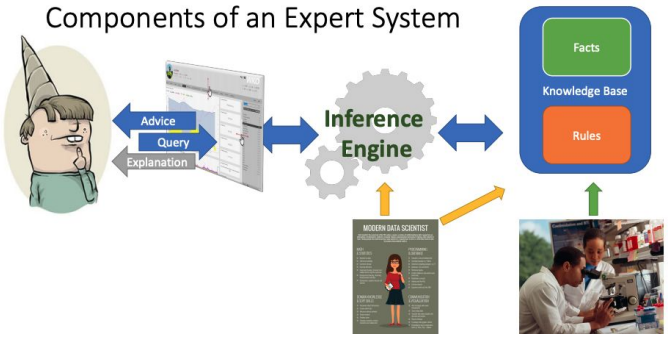
- Most obvious is medical diagnosis
- Biological classification
- Crime investigation (like sketching criminal descriptions, not like CSI: Miami)
- 

Expert Systems are popular:

- They can be checked for correctness
- Compare this to machine learning, where many decisions are black-box

Components of an Expert System:

- User makes query, gets sent to an inference engine which interacts with the knowledge base (composed of rules and facts determined in the knowledge engineering process)
- User receives back advice and perhaps an explanation about why that particular advice was given



Summary

- Expert systems emulate human expertise in specific application domains
- Consist of a user interface, inference engine and knowledge base
- Explanation facilities of Expert Systems can be very important
- MANY formalisms for representing knowledge and reasoning mechanisms for deriving conclusions
- Mainly two types: probabilistic and non-probabilistic

Inference Methods that may be used:

- Bayes rule and naive bayes
- Graphical models
- Factor Graphs
- Markov Network
- Bayesian Network

Advantages of Expert Systems

- Consistency. It will make the same decision given the same data
- Memory. It won't forget a rule or fact...humans forget stuff all the time.
- Logic. No sentimentality that clouds its decision making.
- Infinitely reproducible. Doesn't get tired, can be copy-pasted to other places

Disadvantages of Expert Systems

- Common sense is a problem. It's hard to program common sense
- No creativity (it can't find new solutions to new problems)
- Hard to maintain. Knowledge base, with potentially 1000s of facts, needs to be updated manually (this is the real killer)

Lecture 5: Multi-Agent Coordination

So far, we've seen

- Agent architectures (the form of agents)
- Generating behavior (through evolutionary computation, through reactive behavior, and through reasoning systems (expert systems)
- But what about, once you have agents, getting them to coordinate with each other?
- Imagine an Amazon Warehouse, for example!
- Fun fact. There are more than 200 000 robots working in Amazon fulfilment centers.

So, what is "Coordination", really?

A few different definitions.

- Coordination is the act of managing dependencies between activities
- Coordination is **any decision** that uses information that concerns the existence, decisions, or decision making strategies of others.

We'd like to characterise cooperative systems.

Cooperative systems can be characterised in terms of their environment, the nature of the entities doing the cooperating, or the type of cooperation itself.

- Type of Environment:
  - Diverse/Not Diverse
  - Predictability
  - Dynamics of change
- Cooperating entities
  - How many are there?
  - How homogenous are they?
  - What are their goals? Are they shared goals? Or does each have their own goal?
  - Think about this in terms of human beings. In one sense, we all share the same goal. But we also individually all have different goals. Sometimes these align and sometimes they don't. Focusing on goals is clearly a first-class modeling abstraction.
- Type of Cooperation:
  - Frequency: one-time cooperation?
  - Patterns of cooperation:
    - Recurrent task?
    - Hierarchical Structure?
  - Nice example of a co-operative system where interacting entities are tightly coupled by a strongly shared goal: the Bee movie! link: bee

Why coordinate?

Why actually coordinate in the first place? Why not just have central control from one entity?

Principle of Bounded Rationality:

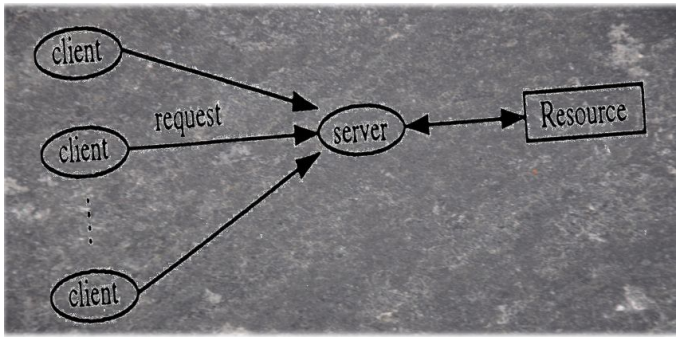
- The human mind's processing capacity is limited
- The amount of information that can be processed by an individual is limited
- The amount of control a single individual can wield is limited

The Basic Models and Mechanisms for Coordination

This is the "meat" of this lecture. Make sure you know these.

Standard Client Server

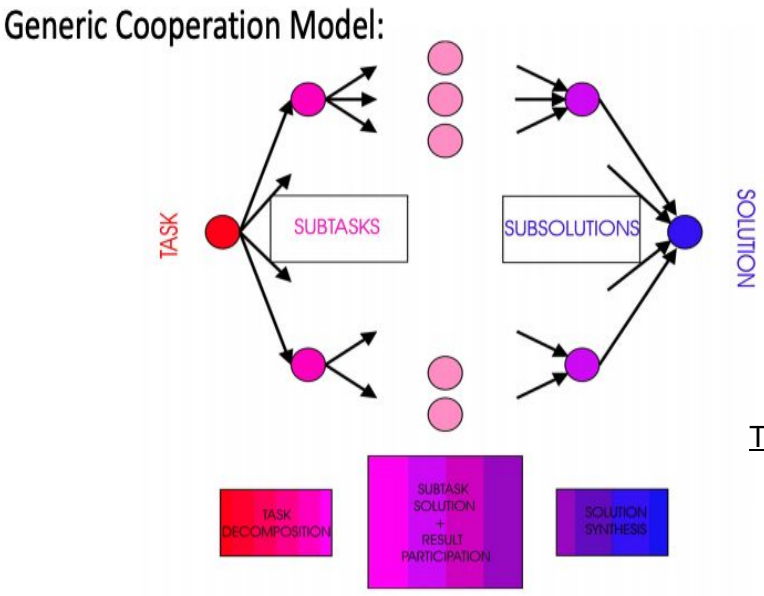
- Client
  - sends requests for services
- Server
  - receives requests from clients
- The service
  - The actual subject of the request



- Pros of Client - Server
    - Simple control structure
    - simple to synchronize
- Cons of Client - Server
    - The server can be a bottleneck
    - Poor fault tolerance. Oh, your server is down? Have fun, nothing works now.

Task and Results Sharing

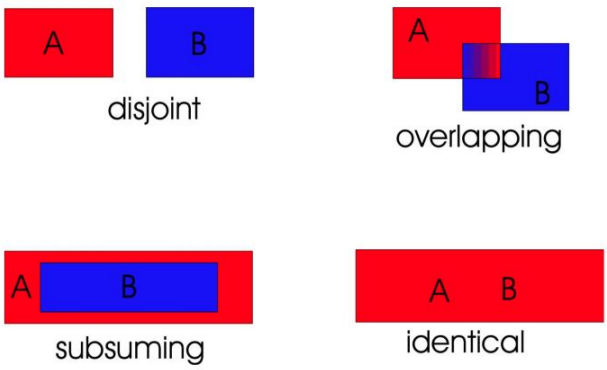
Perhaps we could make a cooperation model based on sharing of tasks and sharing of results!



- Take a task: Split it up into subtasks. This is called Task Decomposition
- Solve each subtask
- Combine all solved subtasks (subsolutions) into the full solution

Sub-task relations

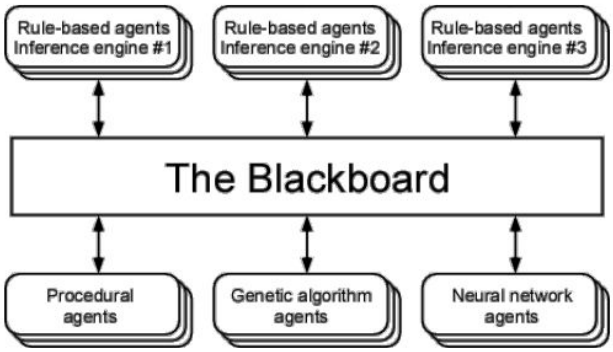
Tasks can be intimately related to one another



- Pros of Task and Results Sharing
    - Each sub-problem can be solved without full knowledge of how to solve the whole problem
    - Each sub-problem requires less resources
    - Different subtasks can be solved simultaneously (Parallelism)
    - Failure to solve one subtask doesn't affect ability to solve others (Robustness)

- Key challenges of Task and Results Sharing
    - ***The connection problem:*** Who exactly managers the coordination of all these subtasks?
    - What does each entity need to know?
    - What should the level of task decomposition be?
    - What combination strategy do you use when combining subtasks?

The Blackboard System



- We have a blackboard: A common memory with read/write control
- Every participant (agent) reads from and writes to this blackboard
- Participants decide on the what and when

When to use the blackboard system:

- Suited for open applications
- Suitable for data driven applications

Contract Net Protocol

Contract Net is a very simple negotiation protocol

A Network of **nodes** (Cooperating units) that act as *managers or contractors*

- **Managers**: Announces tasks to be done
- **Contractors:** Bids for the right to carry out the task
- The contractor with the best bid is **selected by the announcing manager**.
- Nodes can switch roles (from manager → contractor or contractor → manager)

The Task Announcements in Contract Net need to fulfill certain criteria

- Must specify **who is eligible**
- Must abstract the task (describe it in short)
- Must specify what the bids should look like
- Must have an expiration date

Contract Net is a kind of middle-ground between Client-Server and Blackboard.

Design Considerations of Contract Net

- What tasks should be announced?
- Who should receive the announcements? Everyone or a preselected few?
- What incentivizes the contractors to actually bid?
- In the event of multiple similar bids, how do you select which one receives the contract?
- In the event of multiple announcements, how do the contractors choose which one to bid for?

Functionally Accurate Cooperation (FA/C)

- FA/C is a *guideline for cooperation* when the individuals' local knowledge is **incomplete, uncertain and inconsistent**

> "If available information is not perfect, do not longer aim at building a system in which only completely accurate information is exchanged among cooperating entities. Instead, in response to this lack of perfection make sure that the involved parties exchange functionally correct information (tentative partial results) and that they cooperate in refining these information."

Key terms:

functionally correct: the information is *acceptable and reasonable* from the agent's (local) point of view.

cooperation: the agents cooperate to transform their locally correct knowledge into globally correct understand, in a process of iterative refinement.

To do this, they need to be able to:

- measure and evaluate functional correctness on its own received information
- detect inconsistencies between its own tentative results (which are incomplete) and the results received from others
- The agent needs to integrate into its local database those portions of partial results which are consistent with its own results
- revise and extend its own partial results on the basis of the newly integrated data

Advanced Models and Mechanisms for Coordination

Auctions

The valuation of goods in an auction is different for different auctions:

- **Private-value auction:** each bidder's value of the good depend only on her/his own preferences (e.g. art auctions)
- **Common-value auction:** each bidder's value entirely depends on other agents' values (e.g. fresh food auctions)
- **Correlated-value auction:** bidder's value depends partly on own preferences and partly on others' values

Further, we can classify risk attitudes in the people participating in the auction:

- **risk-averse bidder**: bidder who would prefer to get the goods even if they paid slightly more for it than her/his private value

- **risk-averse auctioneer**: auctioneer who prefers to sell the good even if at a lower price than he could achieve under different circumstances. Because by waiting for a new auction, you are taking a risk (what if the next auction is worse? What then?)

- **risk-neutral:** neither risk-averse nor prepared to take a risk. Not sure what this would mean in practice 🤔

Type of Auction	**Rules**
English Auction (first-price, open-cry)	Bidder is free to raise his bid Auction ends when no bidder is willing to raise anymore
First-price, sealed bid	Each bidder submits one bid without knowing the bid of the others
Vickrey (second-price sealed-bid)	Each bidder submits one bid without knowing the others. Highest bidder wins, but only needs to pay the price of the 2nd highest bid
Dutch Descending	The auctioneer starts at a high price and continuously lowers it. Auction ends when one bidder takes at the current price

Which Auction is best?

This question can't be answered because you need to specify who it would **be best for**, the auctioneer or the bidder? We can only talk of expected outcomes.

Expected Outcomes of Auctions (Auction Theory)

English/Dutch/Vickrey/First-price-sealed-bid produce the same expected revenue to the auctioneer in private value auctions where the values are independently distributed and bidders are risk-neural

Among risk averse bidders, Dutch and First-price-sealed-bid give higher expected revenue to the auctioneer than Vickrey or English

(because bidder can 'insure' himself by bidding more than would be offered by a risk-neural bidder)

*Risk-averse auctioneers achieve higher **expected** revenue via Vickrey or English than via Dutch or First-price-sealed-bid*

Issues with Auctioning

You can have bidder collusion, where bidders act with knowledge of their partners bids (a Dutch auction where bidders collude to wait until the price is quite low maybe)

Voting Systems

Type of Voting System	Rules	Problem
Plurality Protocol	Everyone votes simultaneously on a selection of candidates One of the candidates (with most votes) wins.	A random alternative can split the majority If you vote Independent in the US, you are removing a vote from the normal Democrat-Republican 2-party system.
Binary Protocol	Vote on the candidates in pairs: A or B? okay now A or C? The surviving alternative is the winner	The agenda is now very very important. A good candidate who enters in at the end is more likely to win than the one who starts.
Borda Protocol	Each voter rates the candidates in a long list |A| candidates: Top of list gets |A| points 2nd gets |A| -1 etc	Adding and removing irrelevant alternatives

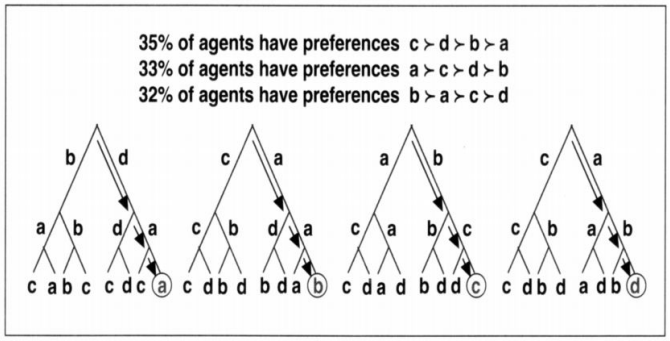
Influence of irrelevant alternatives in the Borda Protocol:

If we remove candidate D, a different candidate wins.

Agent	Preferences
1	$a \succ b \succ c \succ d$
2	$b \succ c \succ d \succ a$
3	$c \succ d \succ a \succ b$
4	$a \succ b \succ c \succ d$
5	$b \succ c \succ d \succ a$
6	$c \succ d \succ a \succ b$
7	$a \succ b \succ c \succ d$
Borda count	*c* wins with 20, *b* has 19, *a* has 18, *d* loses with 13
Borda count with *d* removed	*a* wins with 15, *b* has 14, *c* loses with 13

Influence of the Agenda on the Binary Protocol:

The final candidate changes based on how the agenda is set



Commitments and Conventions:

As social beings, we can make many different types of commitments. We can make:

- Psychological commitment (commitments we make to ourselves)
- Social commitment (to others, performing actions or preventing circumstances)
- Joint commitment (ones we make with others, to undertake joint activities)
- Precommitment (decision to get involved / commit to something sometime in the future)
- Leveled commitment (can cancel, under penalty)

Illustrations:

Conventions

A convention is a description of circumstances under which an agent should (or is allowed to) *reconsider its commitments*

Why do we need conventions?

This goes back to our earlier discussions on tradeoffs.

The fact is that life can often get in the way:

- Between making a commitment and carrying it out, the world may change.
- The commitment could become unattainable
- But we need our agents to be able to respond to changes. Thus conventions are a first-class abstraction.

The tradeoff when designing commitments and conventions:

- If conventions are too strict, the agent's have no flexibility to change course
- If the conventions are too relaxed, the agent's can abandon their commitments too often and become unreliable

Social Convention: description of how to behave with respect to other community members when commitments alter

Centrality Hypothesis of Commitments and Conventions (Jennings, '93)

"*All coordination mechanisms can ultimately be reduced to joint commitments and their associated social conventions."*

Multiagent Coordination then becomes a result of the interplay between:

Commitments + Conventions
+ Social Conventions
+ Local Agent Reasoning

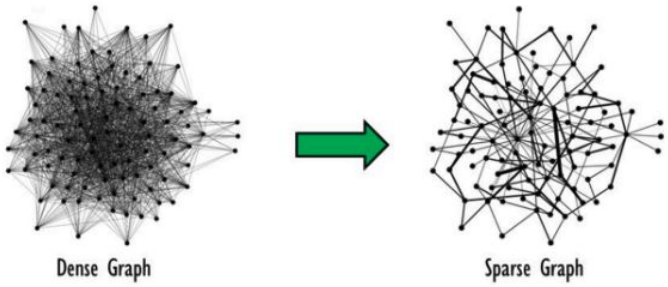
Lecture 6 and 7: Multi-Agent Reinforcement Learning

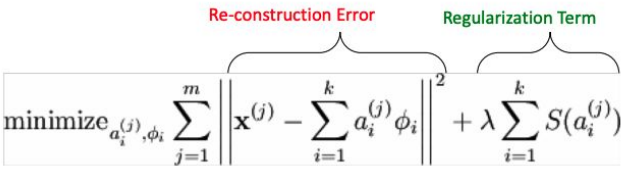
I think it's best to print these slides out.

Lecture 8: Deep Learning

Sparse Coding (https://blog.metaflow.fr/sparse-coding-a-simple-exploration-152a3c900a7c)

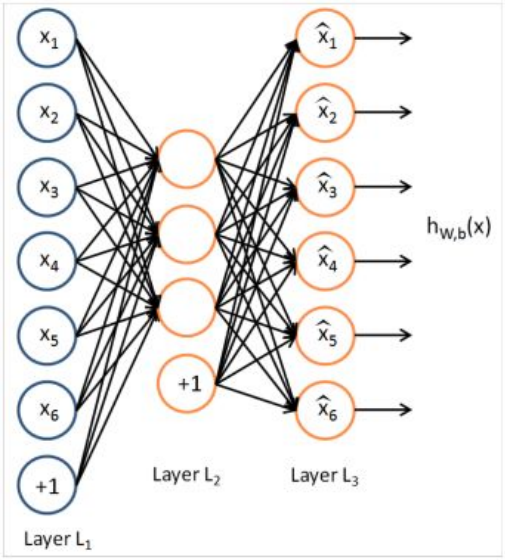
- Is the study of algorithms which aim to learn a useful **sparse** representation of any given data.
- The algorithm only needs input data to learn the sparse representation. This is very useful since you can apply it directly to any kind of data, it is called unsupervised learning
- It will automatically find the representation without losing any information (As if one could automatically reveal the intrinsic atoms of one's data).





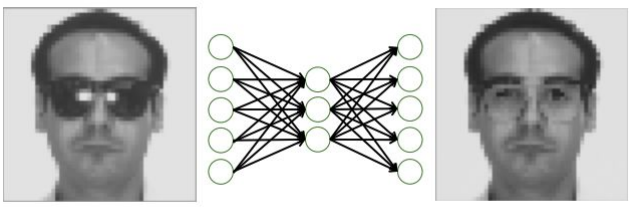
Autoencoder (See Deep Learning Book - Bengio, Courville, Goodfellow)

- An auto encoder is a neural network that is trained to attempt to copy its input to its output.
- Internally, it has a hidden layer **h** that describes a code used to represent the input.
- The network may be viewed as consisting of two parts:
- an encoder function h=f(x)
- and a decoder that produces a reconstruction r=g(h)
- If an autoencoder succeeds in simply learning to setg(f(x)) =x everywhere, then it is not especially useful.
- Instead, autoencoders are **designed to be unable to learn to copy perfectly**.
- Usually they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data. Because the model is **forced to prioritize** which aspects of the input should be copied, it often learns useful properties of the data.

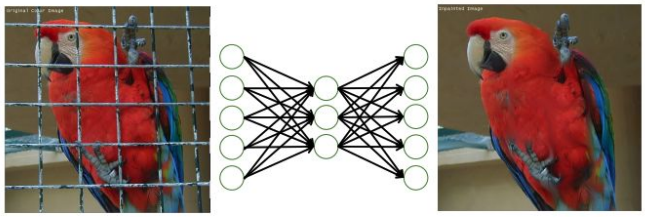


In this autoencoder above, the hidden nodes are less numerous than the input nodes. Thus they have to be informative, and are forced to learn a compact representation of the data.

What has happened here?



- The autoencoder was trained on images of normal faces, and therefore does not know how to represent sunglasses.

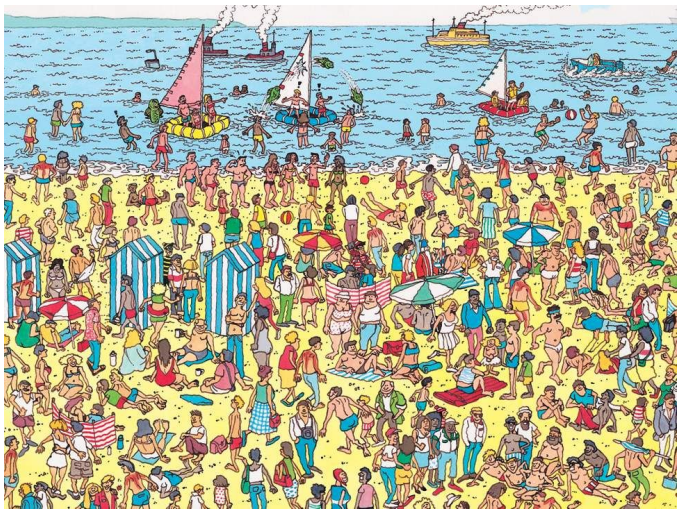


- Similarly, the autoencoder never learned to represent a fence in front of a bird, so it "inpaints" (fills in the blanks) of the bird and the fence is not in the final image.

Convolution for Object Detection (https://d2l.ai/chapter_convolutional-neural-networks/why-conv.html)

Imagine that you want to detect an object in an image. It seems reasonable that whatever method we use to recognize objects should not be overly concerned with the precise location of the object in the image. Pigs usually do not fly and planes usually do not swim. Nonetheless, we should still recognize a pig were one to appear at the top of the image.

We can draw some inspiration here from the children's game 'Where's Waldo':



The game consists of a number of chaotic scenes bursting with activity. Waldo shows up somewhere in each, typically lurking in some unlikely location. The reader's goal is to locate him. Despite his characteristic outfit, this can be surprisingly difficult, due to the large number of confounders.

However, **what Waldo looks like does not depend upon where Waldo is located.** This is they key. We can detect Waldo by sweeping the image with a "Waldo detector" that could **assign a score to each patch**, indicating the likelihood that the patch contains Waldo. We aim to learn a model so that wherever the "waldoness" is highest, we should find a peak in the hidden layer activations.

CNNs systematize and exploit this idea. Essentially, the convolutional layer picks windows of a given size and weighs their intensities according to a **filter**. It runs this across the image and then gradually collapses it's representation (using max-pooling operations) until we can make global statements about the objects in the image.

Convolutional Filters

$$\begin{bmatrix} p_{0,0} & p_{0,1} & p_{0,2} & \cdots & p_{0,n} \\ p_{1,0} & p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{m,0} & p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{bmatrix} \text{ Image}$$

$$\text{Filter } \begin{bmatrix} f_{0,0} & f_{0,1} & f_{0,2} \\ f_{1,0} & f_{1,1} & f_{1,2} \\ f_{2,0} & f_{2,1} & f_{2,2} \end{bmatrix}$$

$$np_{2,3} = f_{0,0}p_{1,2} + f_{0,1}p_{1,3} + \ldots + f_{2,2}p_{3,4}$$

Please watch this 1m45s video:
https://www.youtube.com/watch?v=f0t-OCG79-U
At 00:59 you witness the ReLU activation (reds are negatives and only greens (positives) are left
At 01:01 you witness Max Pooling (the size of the image shrinks)
At 01:40 you see how the final evaluation works (softmax)

Also note! Detecting objects in the real world is still harder than we can imagine due to the number of edge cases! Here is Director of Self Driving at

Tesla talking about how difficult even detecting stop signs are (due to the long tails of the distribution). It's honestly just insane, the fact that we can do this totally effortlessly boggles my mind.

02m01s video: https://twitter.com/hardmaru/status/1252768723976335360?lang=en. This video actually made me reconsider whether it's worth it (in the self driving case) to keep exploring the long tail...maybe in the future all stop signs should come with an RFID chip that can be recognized by a self driving car so that we don't need to handle all this. It's just too much.

This is a fairly deep rabbithole but I think the main thing to know is that:

- Neural networks can work at different levels of abstraction (Hierarchical learning)
    - There is a natural progression from low level to high level structure

  1st layer: Edges.
  2nd layer: Object parts.
  3rd layer: Objects

AlphaGo:
12 minute talk on how AlphaGo works if you want and then read over slides.
https://www.youtube.com/watch?v=MgowR4pq3e8&t=541s

My main takeaways:

- AlphaGo is a very elegant and powerful combination of different techniques (Deep RL + MCTS) into 1 system
- RL can actually learn very complex and nice stuff --provided you are in **friendly** environments

Lecture 10: Pathways to Human Level AI

What about different paths to AI:

Path 1: Use Genetic Algorithms
Evolution gave rise to intelligent systems. Could we emulate this process on a computer and get similar results?

Arguments for using genetic algorithms to get to AGI:

- Computational power is just rising and rising and we can expect it to keep doing so
- Blind, undirected evolution gave rise to intelligence one. That's proof it could work.
- Surely then, a well-engineered evolutionary process can get there much faster?

Arguments Against using Genetic Algorithms to get to AGI

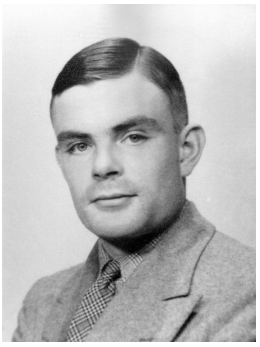
- There is a massive level of **selection/survivorship** bias in the above argument. The fact that we exist doesn't make our existence likely...It could, for all we know, be the only case in the whole universe! We could be extremely, extremely lucky to be here. (I.e: Evolution is a very narrow path)
- Natural intelligence and artificial intelligence might not be the same thing. A plane can fly not because it is working on the same principle as a bird. They really are quite different things.
- Computational power might not actually rise high enough -- could be an almost unreachable number

Path 2: Emulate a whole brain

- Step 1: Take a particular person's brain
- Step 2: Scan it in detail
- Step 3: Construct a software model of it that's faithful to the original
- Step 4: Run on appropriate hardware where it will behave in essentially the same way as the original brain
- Step 5: ???
- Step 6: Profit
- Side note: Could potentially allow digital backup of our consciousness / ability to repair brain damage at will / immortality

The Church Turing Thesis:

- The Church Turing Thesis is simply the idea that a Turing Machine can compute the same functions as any other Turing Machines.
- A machine is called Turing Complete if it



can emulate a Turing Machine. Thus all Turing Machines are Turing Complete.

The Physical Church-Turing Thesis

- The idea that every function which is physically computable...can be computed by a Turing machine (Turing machines are universal)
- Note that the function might be infeasible, but at the very least, if a Turing Machine had infinite time and infinite memory, it could do it.

Levels of Detail Needed to Emulate a Whole Brain
Obviously, there is a difference between your brain and someone else's brain. So we can talk about what exactly we are emulating here:

- Brain Emulation:
    - Software that models the states and functional dynamics of **a brain** (any brain -- the generalized brain)
- Mind Emulation:
    - A brain emulator (so includes above) that is detailed enough to produce

the effects of a mind (The brain you emulate will have an inner life)

- Person Emulation:
    - Mind Emulation (so includes above) that is so detailed, it's actually emulating the mind of a particular individual (The brain you make has the personality and memories of Charlie Chaplin. In a sense, actually IS Charlie Chaplin)

literally

<u>Two Assumptions underlying Whole Brain Emulation</u>

- **The non-organicism assumption:**
    - Total understanding of the brain is not needed, just understanding the component parts and their functional interactions.
    - A simple example: Compilers do not understand software and merely perform syntactic operations that transform human source code into machine executable code.
    - Thus a Whole Brain Emulation pipeline might (without any understanding) mechanically convert a physical system (a brain) into a software system (a simulation)
- **Scale Separation**:
    - There exists a cut-off point beyond which we don't actually need to emulate in order to get the functionality we want
    - An example: Modeling the movement of planets, doesn't require a weather model of Jupiter (or even further, of the Amazon rainforest)

<u>Other pathways</u>

Turing (1950) said: *"Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain."*

This has influenced the idea of **Seed AI,** machine learning without human intervention, in which an AI improves itself by recursively rewriting its own source code. A process known as **recursive self improvement**.

--End--