Grading Scheme:

- 2 Quizzes - 15% + 15% = 30%
- Lab & Assignments - 20%
- Mid Sem Exam - 25 %
- End Sem Exam - 25 %

Professor:

- Vinayak Abrol
- B-409, R&D Block
- Office Hours: Wednesday, 11:00 am - 12:30 pm

Teaching Assistants:

- Mahak Sharma
- Kirti Lakra
- Mohnish Basarkar
- Roshan Jangid

Lecture 2

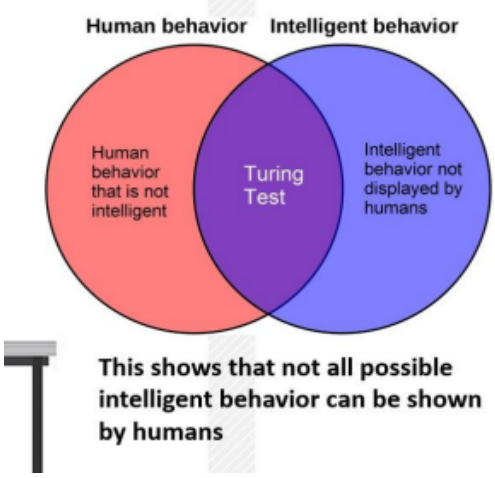
- **What is AI:**
  - **Unnatural - Manmade Intelligence** for machines
  - Simulations
- **Desirable behaviours of AI:**
  - Has **prior knowledge**, builds upon it, **uses it**
  - Understanding, Reasoning, Learning
  - Ability to solve problems
- **Purpose of AI:**
  - Making the computer perform **more complex tasks**
  - Use computers for **decision making**
  - Higher abstraction of programming
- **Progression of AI:**
  - **Narrow AI** - applying to specific predetermined tasks
  - **General AI** - applying to multiple areas and autonomously solving problems that were not the initial intention of the designed structure
  - **Artificial Super-Intelligence** - applying to any area involving scientific creativity, social skills, general wisdom
  - Heuristics and rules - defined sets that govern the workings of the AI
  - Linear models/Decision Trees
  - Deep Models/Ensembles
  - Meta and reinforcement learning

==========

Lecture 3

- **Linear Model** - mapping from input to output is a linear function
- **Non-Linear / Deep Model** - Exponential, Waves, Mixtures, usually more accurate
- The structure could have sequential as well as parallel arrangements
- Examples:
  - Google's project SunRoof
  - Game: AlphaGo defeats Game: Go Champion
  - CMU's Libratus defeats human poker players
  - LipNet from Oxford achieves over 93% success rate in reading people's lips
- **Real AI**
  - Initially, one defines rules, and the AI keeps on building on those to develop logic, rationale, etc
  - General AI is obviously harder to develop as compared to special-purpose (narrow/non-trivial) AI - linked to a specific task
  - The inspiration behind building an AI is its coherence with the human brain - which is not fully understood yet, but there are realizations of certain unique functions and features that a human exhibits which are clearly attributable to the workings of the brain

- **Focus of AI**
  - Intended to **think and act like a human** (not in all cases) - develop a rationale
  - The focus of AI overlooks the emotional aspect of the system - eg: “Is the system conscious?”
  - Most AI systems follow **“Act Rationally”** - as humans are more rational in complex domains
  - **Establish a balance** - humans may be good at some tasks, while machines may be better at others; we don’t want to give one the complete control
- **Worries about AI**
  - **Technological unemployment** - robots taking over manual jobs
  - **Autonomous systems** - examples:
    - An autonomous car gets into an accident; who is to blame? There does not exist any legal code against machines as of yet
    - Autonomous weapon systems pose threats to countries that don't have access to them; they may malfunction and start a conflict anytime
- **AI Prehistory -** Built around other fields of study:
  - **Philosophy -** logic, reasoning, foundations of learning
  - **Math -** formal representation of proofs, algorithms
  - **Economics -** utility, decision theory
  - **Neuroscience -** physical substrate for mental activity
  - **Psychology -** perception, motor control
  - **Computer Engineering -** build fast machines
  - **Control Theory -** build systems that maximize objective function over time
  - **Linguistics -** knowledge representation, grammar: you hear something in English, it subconsciously gets translated to the language you’re most familiar with; you can respond back in English within microseconds
- **Turing Test**
  - A human questioner asks some respondents some questions, and based on the rationality of their answers, they decide whether the respondent is a machine or a human
  - A human is affected by their previous situation, while a machine may not be as influenced by emotional damage
  - Not all possible intelligent behavior can be shown by humans



- **Birth of AI as a separate field**
  - ‘50 was centered around the work of Turing - cryptanalysis, and related fields
  - ‘51 was mostly game bots for chess, draughts, etc. - it was something seen very commonly among people, the idea was to develop a thing (AI) into something more generalized
  - ‘55 was reasoning and application of proper logic - ex: Mathematica, MatLab
    - “Logic Theorist” was created

        - Proved 38/52 theorems in Principia Mathematica
    - ‘56 saw the establishment of AI as a defined field at the Dartmouth conference
    - “General problem solver” by Nerwell and Simon
    - Natural Language Processing
    - Programming languages like Lisp and Prolog invented
- **Challenges**
    - Limited computer power: not enough memory or speed to accomplish anything truly useful
    - Combinatorial explosion - some problems can only be solved in exponential time
    - Common knowledge and reasoning - vision or NLP requires massive info about the world
    - Promises of AI researchers failed to become reality
    - Conscious bias toward humans - if I am apprehensive about being able to solve a problem, I may not be able to work to my full potential towards it
    - Cut off of funding due to lack of progress
- **Boom Phase (‘80-’87)**
    - XCON was one of the first encouraging factors for people to delve into AI
    - Japan aggressively funded AI and made significant advancements
    - This motivated the US & UK to go for the same

==========

Lecture 4

- **After the Boom Phase (‘87-’93)**
    - Lisp machine collapsed as Apple and IBM started gaining speed and power
    - Desktop computers start superseding (more powerful and efficient) expensive heavy machinery like XCON
    - Funding for AI was cut down due to the limitations of expert systems and expectations for Japan’s 5G Project not being met
- **Areas of ML**
    - Deduction, reasoning, problem-solving
        - Theorem provers, puzzle solvers, board game bots
    - Knowledge representation systems, like expert systems
    - Automated planning and scheduling
    - NLP:
        - Language understanding
        - Speech understanding and language generation
        - Machine translation and text mining
    - Motion and manipulation - robotics
    - Social and Business intelligence, like customer behavior modeling
- Followed by application of P&S to AI in general which ultimately led to the onset of ML, knowledge-based systems, robotics, etc

    - Core algorithms do not change but the application of modern systems to these algorithms opens more possibilities
    - Pattern recognition: identifying and mapping patterns in datasets without human intervention
    - Machine perception: computer vision + speech recognition
    - Robotics
    - NLP: linguistics, text-to-speech, identify languages and communicate in real-time, recognize scripts/speech in real-time to identify keywords in a phrase and react accordingly
- Geoffrey Hinton
- **Unsupervised Learning**
    - any form of learning model which does not have prior information about the pattern it identifies, process it executes, etc.
    - The computation model is able to see and identify patterns/specific features that give reason to “cluster” these datasets
- **Areas where Human Intelligence is supreme**
    - Reading: humans can establish links between different pieces of text, rather than plainly reading the text and only inferring its local meaning
    - Translation: humans could not know the script, but still identify the conveyance through some visual cues, etc
    - Art & music: eg - Stockfish beating Vishy Anand; machines may copy the style of previous artwork and apply it to produce new paintings, but they can’t produce their own unique designs
    - Adapting/self evaluation/creativity
    - Emotional intelligence: eg - humor; machines have a relatively inferior outlook on the emotional aspect
    - Transfer learning: humans can apply learnings from one part of the world which they took in as insights towards solving problems in other domains
    - Explaining our reasoning: humans tend to be able to explain their thought processes towards the development of their reasoning
    - Knowing what it’s like to be human
- **Modern AI -** AI is successful broadly due to:
    - Incredible computational power
    - Greater emphasis on solving sub-problems
    - Newer mathematical models and statistical research
- **Definition of AI**
    - Sense - Plan - Act
    - AI is the science and engineering of making intelligent machines that can perform tasks that require intelligence when done by humans
    - Study of “intelligent agents” - devices that act in accordance with the maximum possibility of success; these actions are influenced by the environment around them
    - System’s ability to interpret and learn from data, thus applying these learnings to achieve its goals and tasks through constant adaptation
- **Characteristics of AI systems**

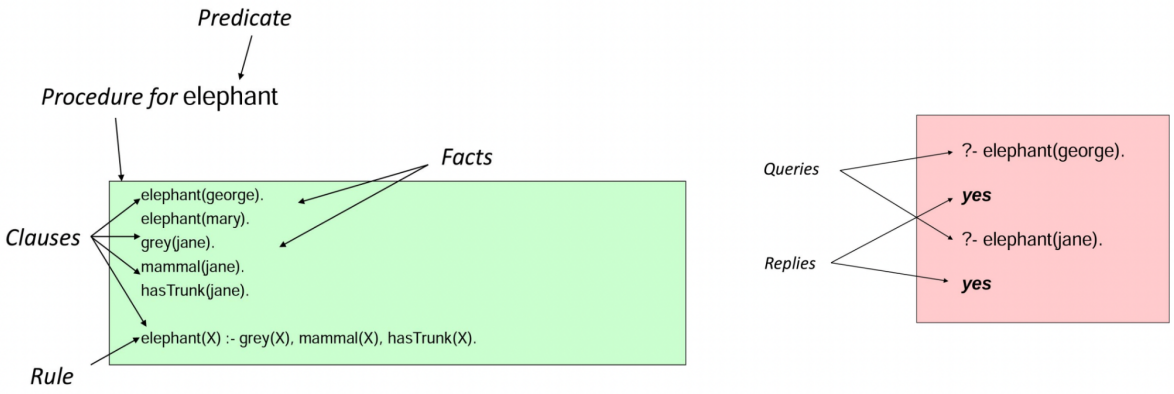
    - Knowledge
        - Explicit - stated by the system - defined/written rules
        - Implicit - conclusions inferred from hard facts
        - Tacit - understood
    - Reasoning
        - Explicit - given through hard facts (rules) to the system; something called “balance”, ex: **A** ⇒ **B** and **B** ⇒ **C** then **A** ⇒ **C**
        - Implicit - situation-based
    - Learning
        - Explicit - pattern or rule; ex: if rainy and you get drenched then you can get flu and it can be dangerous
            - Generalization & Specialization
        - Implicit - learning that occurs inside the network
            - Generalization & Specialization

==========

Lecture 5

- **Steps to design a system** - Decision Making **(for example Stream Selection)**
    - **Sequential Approach:** marks > interest > aptitude **or** interest > aptitude > marks
    - **Non-Sequential:** weighted average score of the three components
    - **Probabilistic Model:** Based on the weighted average, look at sample populations and predict the match of the appropriate stream as a percentage
    - **Next level:** predict the future i.e. decisions are influenced by future prospects
- Types of knowledge:
    - Explicit (documented) & Implicit (applied)
    - Tacit (understood - literal meaning)
- Explicit Knowledge Representation (factors to be taken into account when building an AI system)
    - **Logic -** machine language, programming language, etc.
    - **Rules -** lay down rules according to which the system behaves
    - **Cases -** what are the different cases in the case of this problem
    - **Frames -** frame of reference; framing of a problem (in a business context)
    - **Models -** what possible framework can the system follow
    - **Ontology -** grouping of data (red balls, blue balls), tree representation/diagrams
- PROLOG - declarative programming (**PRO**GRAMMING **LOG**IC)
    - Avoid describing what the program is doing & rather describe how to do it as a sequence of primitives in that language
    - Search through the space of solutions
    - All prolog data structures are called **terms** like:
        - Constants - an atom or a number
        - Variables

        - Compound terms
    - **Program = Facts + Rules + Queries**
    - Contains clauses, rules, queries, replies, facts, predicate
    - The program consists of procedures which in turn consist of clauses - each clause is a fact (a rule) and is executed by posing queries



- **Binding and Unification**
    - Two terms unify if substitutions can be made to make them identical
    - If no such terms exist, they cannot be unified
    - Unification algorithm
        - Constants unify if they are identical
        - Variables unify with any term (constants and other variables)
        - Compound terms unify if their attributes and components unify
    - In essence, **Unification == Pattern Matching** and **Binding == Assignment**
- Operators for comparing terms
    - `X = Y` — X unifies with Y (performs logical pattern matching)
    - `X == Y` — X and Y are identical
        - `[a, b] == [a, b]` succeeds
        - `[a, b] == [a, X]` fails
    - `X \== Y` — X and Y are not identical
        - `[a, b] \== [a, b]` fails
        - `[a, b] \== [a, X]` succeeds, without binding b to X
    - `=n=` — arithmetically not equal
    - `=:=` — arithmetically equal (compares evaluated arithmetic expressions)
    - `<, >, =<, >=`
- Lists
    - Defined as $[x_1, x_2, \ldots]$
    - Nested lists also allowed
    - Operations: **Concatenate, Reverse, Insert, Append, Length**
        - `[1, 2, 3, 4, 5] = [Head | Tail]. [P, Q | E]`
            - Result: `P = 1, Q = 2, E = [3, 4, 5]`
        - `[quod, licet, jovi, non, licet, bovi] = [_, X | _].`
            - `X = licet.` → `True`

==========

Lecture 6

- **Logic** - truth-preserving system of inference
  - **Truth-Preserving -** if initial statements are true, inferred statements will be true
  - **System -** A set of mechanisms and transformations, based on syntax
  - **Inference -** Process of deriving/inferring new statements from old statements
- **Propositional Logic**
  - A proposition is either True or False
  - Generalized statements cannot be facts or propositions
  - Relation to Boolean, but there are extensions - Fuzzy Logic (something in between), Quantum Logic (superposition and collapse)
  - Symbolic AI contains methodologies related to reasoning over logical propositions
  - **Classes of sentences**
    - **Declarative -** contains facts or information - can be translated into propositions
    - **Interrogative -** questions
    - **Imperative -** orders or instructions
    - **Exclamatory -** aimed at evoking emotions but contain no factual content
  - **Notation**
    - Small Letters - generic propositions
    - Capital Letters - specific statements or sets
  - **Basic Elements - Operators**
    - ¬ Negation Operator (NOT)
    - **Λ** Cojunction Operator (AND)
    - **V** Disjunction Operator (OR)
    - → Material Conditional (IF … THEN)
- **FOPL - First Order (Predicate) Logic**
  - Method of converting natural language into a computable format
  - FOPL has variables
  - Assumes that the world contains:
    - Objects - people, students, courses
    - Relations - registered-in, grades-in
    - Functions - avg, min, max
  - **Propositional Logic ⊆ FOPL**
  - **Basic Elements**
    - **Constants**
      - `'Ram', 'Vinay', 32`
      - `'True' & 'False'` are reserved as Truth Symbols
    - **Predicates**
      - Define relations between constants, variables
      - Examples:
        - **`father(Hari, Rajiv):`** Hari is the father of Rajiv

        - **`father(Hari, X):`** X refers to any child of Hari
    - **Functions**
        - Operators on constant and variable data
        - **`sqrt(x), min(L), max(L)`**
        - Example:
            - **`mother(Ajay)`** → Sheela
            - **`green(Apple)`** → True
    - **Variables**
        - x, y, p
    - **Connectives:**
        - Logical Operators: **¬, Λ, V,** →
    - **Quantifiers:**
        - Set Operators: ∀, ∃
        - **∃ `green(apple):`** "Some apples are green"
        - **∀ `apple green(apple):`** "All apples are green"
    - **Arity (# of arguments that predicates hold)**
        - Nullary, Unary, Binary, Ternary
        - **True, ¬x, x Λ y, (p→q) V (¬p→r)**

==========

Lecture 7

- **Compound Propositional Statements**
    - Joint statements - use connectives (**¬, Λ, V,** →)
        - Multiple variables combined using multiple symbols
        - Values of such statements are evaluated using individual truth tables
- **Conditional and Biconditional Statements**
    - Conditional Connective: If A then B (**A→B**) == **¬A V B**
    - Biconditioinal Connective: A if and only if B == **A iff B** == (**A↔B**) == **A⊙B**
- **Tautology, Contradiction & Contingency**
    - Tautology **(**⊤**)**: always True
    - Contradiction **(**⊥**)**: always False
    - Contingency: maybe True, maybe False - in between
    - Identity: evaluates to the variable itself (**p V F; p Λ T**)
    - Satisfiable: ∃ at least one combination of inputs that makes the statement true

- **Equivalences**

double negation	$\neg\neg\alpha$	$\equiv$	α
associativity $\vee$	$\alpha \vee (\beta \vee \gamma)$	$\equiv$	$(\alpha \vee \beta) \vee \gamma$
associativity $\wedge$	$\alpha \wedge (\beta \wedge \gamma)$	$\equiv$	$(\alpha \wedge \beta) \wedge \gamma$
commutativity $\vee$	$\alpha \vee \beta$	$\equiv$	$\beta \vee \alpha$
commutativity $\wedge$	$\alpha \wedge \beta$	$\equiv$	$\beta \wedge \alpha$
distributivity $\wedge$ $\vee$	$\alpha \wedge (\beta \vee \gamma)$	$\equiv$	$(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$
distributivity $\vee$ $\wedge$	$\alpha \vee (\beta \wedge \gamma)$	$\equiv$	$(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
idempotence $\vee$	$\alpha \vee \alpha$	$\equiv$	α
idempotence $\wedge$	$\alpha \wedge \alpha$	$\equiv$	α
absorption 1	$\alpha \wedge (\alpha \vee \beta)$	$\equiv$	α
absorption 2	$\alpha \vee (\alpha \wedge \beta)$	$\equiv$	α
De Morgan's law 1	$\neg(\alpha \vee \beta)$	$\equiv$	$(\neg\alpha) \wedge (\neg\beta)$
De Morgan's law 2	$\neg(\alpha \wedge \beta)$	$\equiv$	$(\neg\alpha) \vee (\neg\beta)$
contraposition	$\alpha \rightarrow \beta$	$\equiv$	$\neg\beta \rightarrow \neg\alpha$
negation $\top$	$\neg\top$	$\equiv$	$\bot$
negation $\bot$	$\neg\bot$	$\equiv$	$\top$
constant $\bot$	$\alpha \wedge \neg\alpha$	$\equiv$	$\bot$
constant $\top$	$\alpha \vee \neg\alpha$	$\equiv$	$\top$

elimination $\top \vee$	$\top \vee \alpha$	$\equiv$	$\top$
elimination $\top \wedge$	$\top \wedge \alpha$	$\equiv$	α
elimination $\bot \vee$	$\bot \vee \alpha$	$\equiv$	α
elimination $\bot \wedge$	$\bot \wedge \alpha$	$\equiv$	$\bot$
elimination $\bot \rightarrow$	$\bot \rightarrow \alpha$	$\equiv$	$\top$
elimination $\bot \rightarrow$	$\alpha \rightarrow \bot$	$\equiv$	$\neg\alpha$
elimination $\top \rightarrow$	$\top \rightarrow \alpha$	$\equiv$	α
elimination $\top \rightarrow$	$\alpha \rightarrow \top$	$\equiv$	$\top$
commutativity $\leftrightarrow$	$\alpha \leftrightarrow \beta$	$\equiv$	$\beta \leftrightarrow \alpha$
elimination $\top \leftrightarrow$	$\top \leftrightarrow \alpha$	$\equiv$	α
elimination $\bot \leftrightarrow$	$\bot \leftrightarrow \alpha$	$\equiv$	$\neg\alpha$

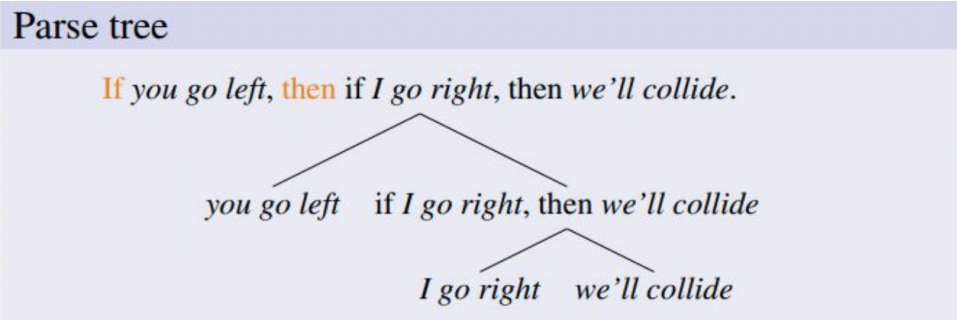
- **Rules of inference in Logic**
  - **Modus Ponens -** method of **affirming (A→B, A ⇒ B)**
    - If A then B
    - Example: It is bright today, therefore I will wear my sunglasses
  - **Modus Tollens -** method of **denying (A→B, ¬B ⇒ ¬A)**
    - If not B then not A
    - Example: I will not wear my sunglasses, therefore it is not bright today.
  - **Modus Tollendo Ponens - Disjunctive Syllogism (AVB, ¬A ⇒ B)**
    - if not A and (A or B), then B
    - Example: I choose soup or salad. I do not choose salad. Therefore, I choose soup.
  - **Hypothetical Syllogism (A→B, B→C, ⇒ A→C)**
    - If A then B and If B then C, then if A then C
    - Example: If I don't wake up then I can't go to work. If I can't go to work then I won't get paid. Therefore, If I don't wake up, I won't get paid.
- **Precedence**
  - **() > ¬ > Λ > V > (→, ↔)**
  - For operators of equal precedence, go left to right
- **Resolution**
  - Breaking down complex propositions into simpler ones

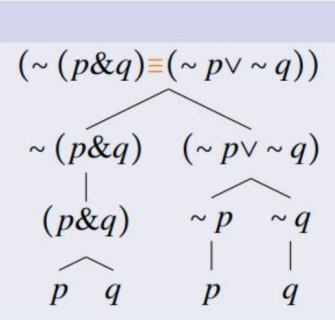
==========

Lecture 8

- **Atomic/Complex sentences**
  - Atomic: simple declarative sentence - cannot be reduced to simpler sentences - either true or false
  - Complex:

        - connectives (such as “of, and, or”) join atomic sentences to form complex sentences
    - Difference between atomic/complex and proposition/compound sentences
        - Propositions/Compounds are in propositional logic
        - Atomic/Complex are in first-order logic



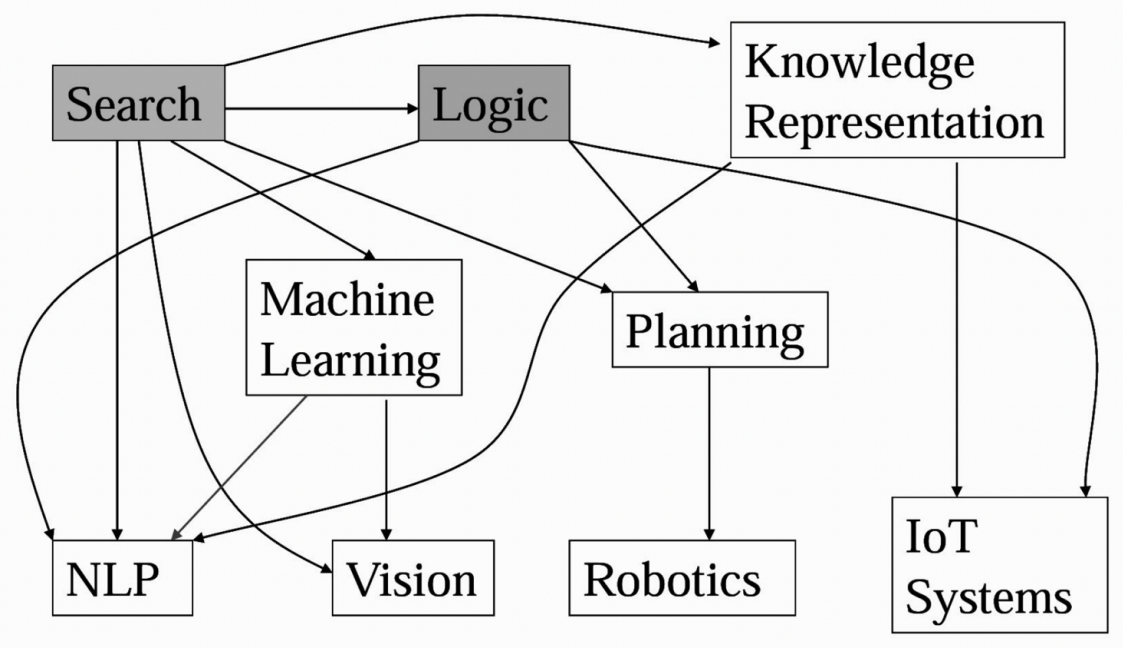


- **Models of FOL**
    - Domain/Universe: non-empty, finite set of elements
    - Interpretation: functions, constants, variables, & predicates on domain elements
    - Example:
        - **"∀ x P(x)"** means all objects in the domain show the property P
- **Quantification**
    - Universal Quantification:
        - A relation that holds for all elements of the domain
        - Example: Everyone who studies at IIITD is smart.
            - **∀ x (studies_at(x, IIITD) → smart(x))**
        - Main connective: →
    - Existential Quantification:
        - A relation that holds for some elements of the domain
        - Example: Some numbers are prime.
            - **∃ x (number(x) Λ prime(x))**
        - Main connective: **Λ**
    - Only Quantifiers of the same type commute, for example:
        - **∀y ∀x == ∀x ∀y**
        - **∃x ∀y Likes(x, y) != ∀y ∃x Likes(x, y)**
    - Duality
        - **∀ (relation) == ¬∃ ¬(relation)**
        - **∃ (relation) == ¬∀ ¬(relation)**
    - Interconnections:
        - **∀ x P(x) == $P(x_1)$ Λ $P(x_2)$ Λ ...**
        - **∃ x P(x) == $P(x_1)$ V $P(x_2)$ V ...**

∀	¬∀	∀¬	¬∀¬
¬∃¬	∃¬	¬∃	∃
every	*not every*	*no*	*some*
everybody	*not everybody*	*nobody*	*somebody*
everything	*not everything*	*nothing*	*something*
everywhere	*not everywhere*	*nowhere*	*somewhere*
always	*not always*	*never*	*sometimes*
anyhow	*not anyhow*	*no way*	*somehow*

==========

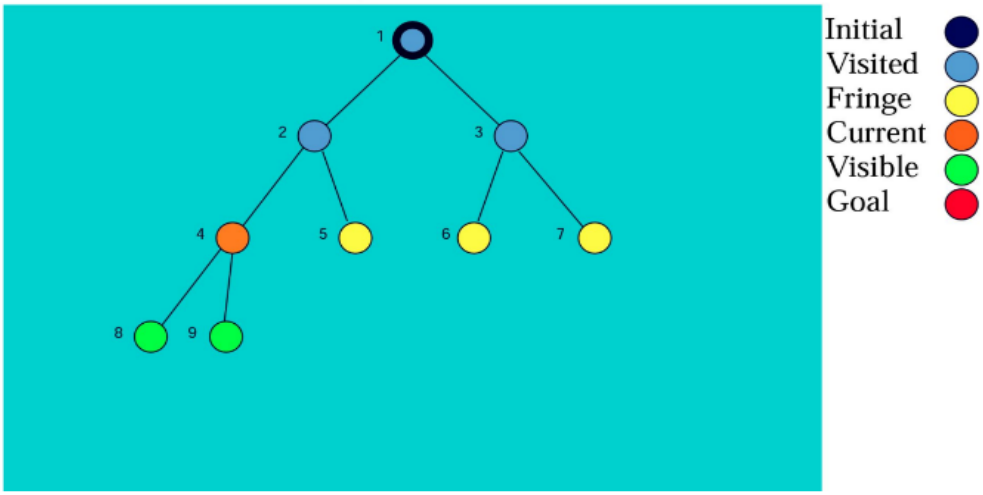
Lecture 9



- **Search Algorithm:** high-level, pattern recognition, requires a lot more coding than logic
- Why do Vision and Robotics not require Logic?:
  - Logic is not required when doing a very specialized task
  - Low-level highly specialized ML Algorithms
- **Search - problem-solving strategy**
  - Many problems can be viewed as **[starting→ goal], i.e. search for a path**
  - There is an underlying state-space that defines the problem and its solutions
  - The state-space can be traversed by applying a successor function/operator to proceed from one state to another
  - Information about the problem or its domain is used to improve the search:
    - Experience from previous instances of the problem

        - Strategies expressed as rules/heuristics
        - Solutions from a simpler version of the problem
        - Add constraints on the problem
    - Search algorithms are the **basis for optimization and planning methods**
    - Optimization: used when a lot of uncertain variables are involved. Example: Weather prediction
    - Planning: used when only a few variables are uncertain, and most of them are predictable.
- **Parts of Search**
    - **Initial state:** the initial state from where the search begins
    - **Goal state:** end the search here
    - **State-Space:** the set of all attainable states
    - **Actions:** set of all possible steps - includes operators and functions
    - **Path:** sequence of steps that lead from one state to another
    - **Goal test:** A test function that checks whether the goal state is reached or not
    - **Solutions:** paths that lead from Initial State to Goal State
    - **Costs:**
        - **Path cost:** sum of the costs of individual actions of a path - analogous to # of operations, not ignoring the constants
        - **Search cost:** path cost of the solution (time & memory)
        - **Total cost:** search cost + all path costs = overall cost for finding the solution
    - **Solution to a search is a PLAN** - a sequence of actions that transforms Start to Goal state
- Assuming the solution exists, Data → Program → Solution
- If the solution is not valid/correct, the program is wrong
- **Types of search**
    - **Uninformed/Blind-** no prior knowledge about anything except the problem
        - BFS: Breadth-first Search
        - DFS: Depth-first Search
        - UCS: Uniform-cost Search
    - **Informed/Heuristic -** some knowledge about the area of the problem/solution
        - Best-first Search
        - Graph Search
        - Greedy Search
        - A* Search
- **Search terminology**
    - **Types of Nodes:**
        - **Initial node:** the node where the search begins
        - **Goal node:** the desired node to reach
        - **Visited nodes:** the nodes which have already been searched
        - **Visible nodes:** the nodes which can be reached from the current node
        - **Fringe/Frontier:**
            - Set of nodes not yet visited
            - Newly generated nodes are added to the fringe

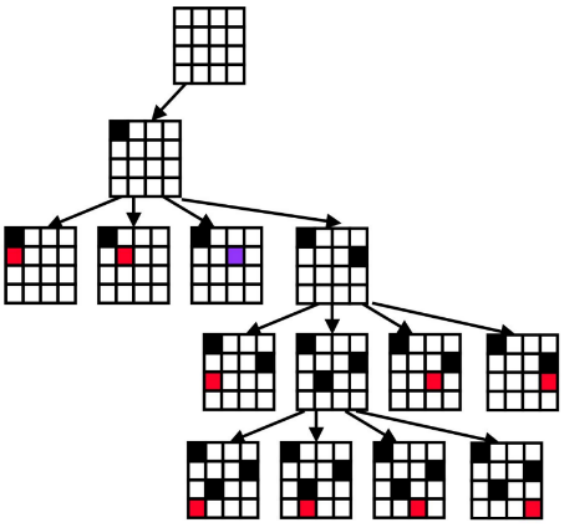
        - **Current node:** the current position of the search
    - **Search space:** usually a graph, on which search is performed
    - **Search tree:**
        - Generated as the search space is traversed; specifies possible paths within the search space
        - Size of search trees gets large for pretty simple problems
    - **Expansion of nodes:** nodes are expanded by the successor function to generate the next set of child nodes
    - **Search strategy:**
        - Determines how to select the next nodes to expand
        - Can be achieved by ordering nodes in the fringe, for example using a Stack or a Queue to store the “best” node w.r.t. some measure



Parameters	Informed Search	Uninformed Search
Utilizing Knowledge	It uses knowledge during the process of searching.	It does not require using any knowledge during the process of searching.
Speed	Finding the solution is quicker.	Finding the solution is much slower comparatively.
Completion	It can be both complete and incomplete.	It is always bound to be complete.
Consumption of Time	Due to a quicker search, it consumes much less time.	Due to slow searches, it consumes comparatively more time.
Cost Incurred	The expenses are much lower.	The expenses are comparatively higher.
Suggestion/ Direction	The AI gets suggestions regarding how and where to find a solution to any problem.	The AI does not get any suggestions regarding what solution to find and where to find it. Whatever knowledge it gets is out of the information provided.
Efficiency	It costs less and generates quicker results. Thus, it is comparatively more efficient.	It costs more and generates slower results. Thus, it is comparatively less efficient.
Length of Implementation	Implementation is shorter using AI.	The implementation is lengthier using AI.
Examples	A few examples include Graph Search and Greedy Search.	A few examples include Breadth-First Search or BFS and Depth-First Search or DFS.

- Example of a searching problem:
  - Put N Queens on an N×N chessboard such that no Queen attacks the other
  - Incremental formulation:
    - **State:** any arrangement of Queens on the board
    - **Initial state:** empty board
    - **Goal state:** can be many, that satisfy the constraints
    - **Successor function:** add a Queen to any square
    - **Goal test:** all queens on the board with no Queen being attacked
    - **Path cost:** irrelevant
  - Complete-states formulation:
    - **State:** any arrangement of Queens on the board
    - **Initial state:** all Queens on the board (on random squares)
    - **Goal state:** can be many, that satisfy the constraints
    - **Successor function:** move a Queen to a different square

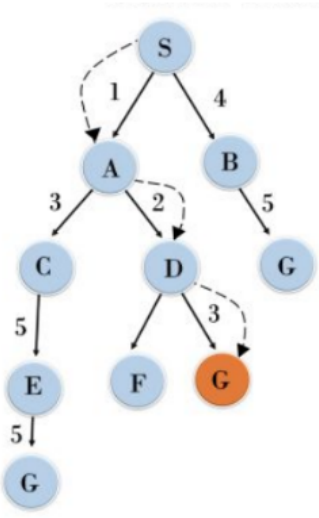
        - **Goal test:** no Queens being attacked
        - **Path cost:** irrelevant
- Good strategies reduce # of possible solution sequences considerably
- Some strategies **trim the search space,** others **add constraints - more efficient**
    - Example: place Queens on “un-attacked” squares only
    - Example: move an attacked Queen to an “un-attacked” square, if possible
    - May not lead to a solution depending on the initial state
- The search ends either when the goal state is reached or when the search space is exhausted



==========

Lecture 10

- **Breadth-first Search**
    - Explore **all possible branches** at the same time
    - Move from level to level, and **process all nodes at each level** before moving on to the next level
    - All nodes visible from the current node are explored first
    - Newly generated nodes are added to a **search queue**
    - **Complete -** explores all possible nodes in the entire tree
    - **Optimal** if all paths are the same cost
    - **Memory-intensive**, as the fringe grows rapidly
- **Depth-first Search**
    - Explore **each node’s successors** first
    - Go inside only **one branch at a time**, and go back to an earlier state to process the other branches **only when you get stuck**
    - Newly generated nodes are added to a **search stack**
    - **Not good** if the goal is on **another (short) branch**
    - **Neither complete nor optimal** - may get stuck in an infinite loop if it starts to explore the wrong branch first, for example, the right branch in a binary tree
    - Uses **less memory**, and has a smaller fringe
- **Uniform-cost Search**
    - Uses the **lowest cumulative cost** to find a path from [source → destination] `= g(n)`
    - Nodes are expanded starting from the root, and according to the minimum cumulative cost
    - **Allowed to backtrack** to a lower cumulative sum in another branch



    - Implemented using **priority queues**
    - **Optimal** - as, in every step, the lowest-cost path is chosen
    - Equivalent to BFS if all path costs are the same
- **Best-first Search**
    - Keep track of **all states generated so far**
    - Pick the **"best" one to explore** from them
    - Not committed to exploring one branch - you can jump between different branches
    - In between BFS and DFS
    - Relies on an evaluation function that rates how useful it would be to expand a certain node - **estimated cost of [node → goal] = `h(n)`**
    - Node with the **lowest heuristic value is expanded first**
    - Node with the lowest value **might not give the optimal path** to the goal

==========

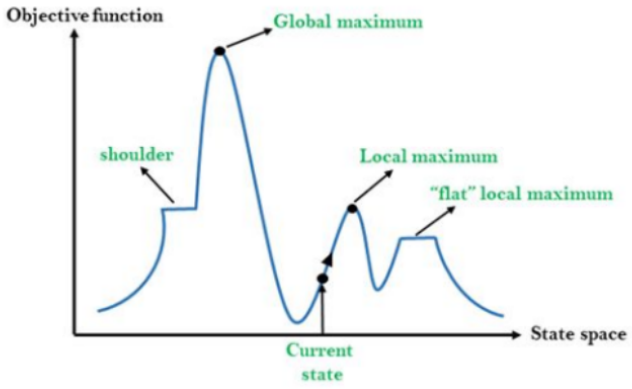
Lecture 11

- **Greedy Search**
    - Algorithm makes the **optimal choice at each step** in an attempt to find the overall optimal solution
    - Does not consider the entire data - **may fail to find the global optimum**
    - Difference from Best-first: here, we **cannot revise our decision**; a path/node once selected, is final
- **A* Search**
    - Similar to UCS
    - It is a Best-first Search, but not a Greedy Search
    - To decide which branch to continue exploring, it adds to the cumulative cost an estimate of how close we are to the goal
        - **A* value = UCS value + distance from goal**
    - Provides a solution **very rapidly**; one of the most used algorithms
    - From the current node, calculate the possible cost to the goal
        - **`f(n) = g(n) + h(n)` = path cost + estimated cost to the goal**
        - **`f(n)`** = evaluator function to select which node to expand
        - **`g(n)`** = cost from initial state to current state **`(n)`**
        - **`h(n)`** = estimated cost from current state **`(n)`** to the goal state
    - The node with the lowest **`f(n)`** is expanded first
    - **Idea: avoid expanding paths that are already expensive**
    - The value of **`f()`** never decreases along any path - all heuristic functions are monotonous
    - **Heuristics** must be **admissible** and **consistent**
        - **Admissible:** $\forall$ **`n h(n) <= h*(n)`** (Optimal Cost to reach goal from n)
        - **Consistent: `cost(A→C) <= cost(A→B) + cost(B→C)`**
        - Consistent ⇒ Admissible

- Best-first: `f(n) = h(n)`
- Uniform-cost: `f(n) = g(n)`

- **Hill-climbing Search**
  - A **heuristic search algorithm** based on **optimization objective**
  - Uses **less memory than A*** - does not store previous path costs in memory
  - Process:
    - Randomly pick a point and look at its neighboring points
    - If you find a point better than the current one, move in its direction
    - Repeat until you reach a point where there is no better one

```
Algorithm hillClimbing() {
    node ← start
    newNode ← head(sort(generateChildNodes(node)))
     while (h(newNode) < h(node)) do {
        node ← newNode
        newNode ← head(sort(generateChildNodes(node)))
    }
    return newNode;
}
```



  - BFS evaluates all nodes and picks the best one, while Hill-climbing moves to a branch as soon as it sees that it is **"better" than the current node** - does not explore the rest
  - Similar to a smarter DFS - as you reach a new branch, pick the node which is most likely to reach the goal
  - **Faster & more memory efficient, but may not be optimal**
  - **Issue:** we may get stuck at a local optimum, plateau, or ridge
  - **Types of Hill-climbing:**
    - **Simple:** Consider just one node
    - **Steepest-Ascent:** Consider all nodes
    - **Stochastic:** Consider one at random
  - **Solutions:**
    - **Local Maximum:** Backtracking

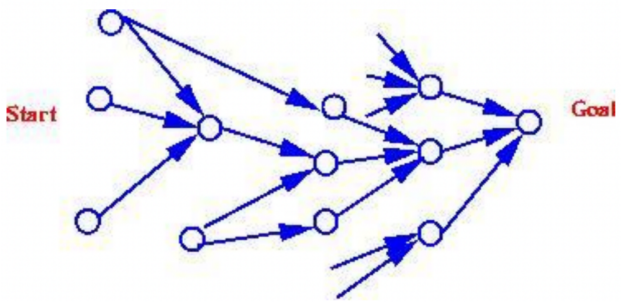
        - **Plateau:** Take larger steps
        - **Ridges:** Simulated annealing

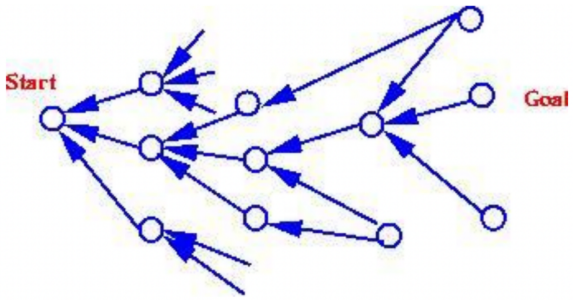
==========

Lecture 12

- **Knowledge Representation**
    - How human knowledge can be encoded into a form that can be handled by computer algorithms
    - Representations should be:
        - **Expressive**: should cater to a large audience
        - **Consistent:** should not be ambiguous
        - **Complete:** should be complete wrt a domain (at least over the variables you have control over)
        - **Extensible:** the framework should be such that it can be modified to take into account more variables / extended to be more general
    - Types:
        - Procedural Knowledge: **HOW to do?**
        - Declarative Knowledge: **WHAT to do?**
    - Knowledge: **explicit** (represented notation) + **implicit** (inference)
    - No universally accepted way how to gather, organize and store knowledge via data structures
    - Representation formats:
        - **Symbol:** distance_left = 4.0; color_right = "blue";
        - **Function:** distance(left, 4.0); color(right, "blue");
- **Rule-Based Systems (RBS)**
    - Knowledge is stored as rules (small pieces of knowledge)
    - **Expert System:** RBS where rules come from human experts in a domain
    - Easy to create - knowledge is different from AI reasoning processes
    - Fast, flexible & easy to expand
    - Rules can be combined or chained together to infer conclusions / derive solutions
    - Rule formats:
        - `IF <antecedents> THEN <consequents>`
        - `IF <conditions> THEN <actions>`
        - `WHEN <events> IF <conditions> DO <actions>`
    - Components:
        - **Working memory -** small allocation of memory where only appropriate rules can be copied and/or interpreted
        - **Rulebase -** the set of rules, stored to facilitate efficient access of antecedents
        - **Interpreter -** processing engine which carries out reasoning on the rules and derives an answer
        - **Explanation facility -** for Expert Systems only
- Procedural Rules

    - Rules that produce the next set of facts, sub-goals and have actions associated
    - Difference between Rules and IF conditions:
        - IF conditions are used to define rules
        - Rules lead to some reasoning (have a logically derived conclusion)
- Reasoning:
    - **Deductive inference rule / Forward Chaining**
        - Conclude **"B" from "A" and "A→B"**, for example:
            - A: It is raining
            - A→B: If it is raining, the street is wet
            - B (Conclusion): The street is wet
        - Goal-driven reasoning **[Start → Goal]**
        - Makes sense if it is the arrival of a new fact that triggers problem solving
    - **Abductive inference rule / Backward Chaining**
        - Conclude **"A" from "B" and "A→B"**, for example:
            - B: The street is wet
            - A→B: If it is raining, the street is wet
            - A (Conclusion): It is raining
        - Rule-based reasoning **[Goal → Start]**
        - Makes sense if it is a query to which a response is required triggers problem solving
    - Choose a direction with a **lower branching factor**

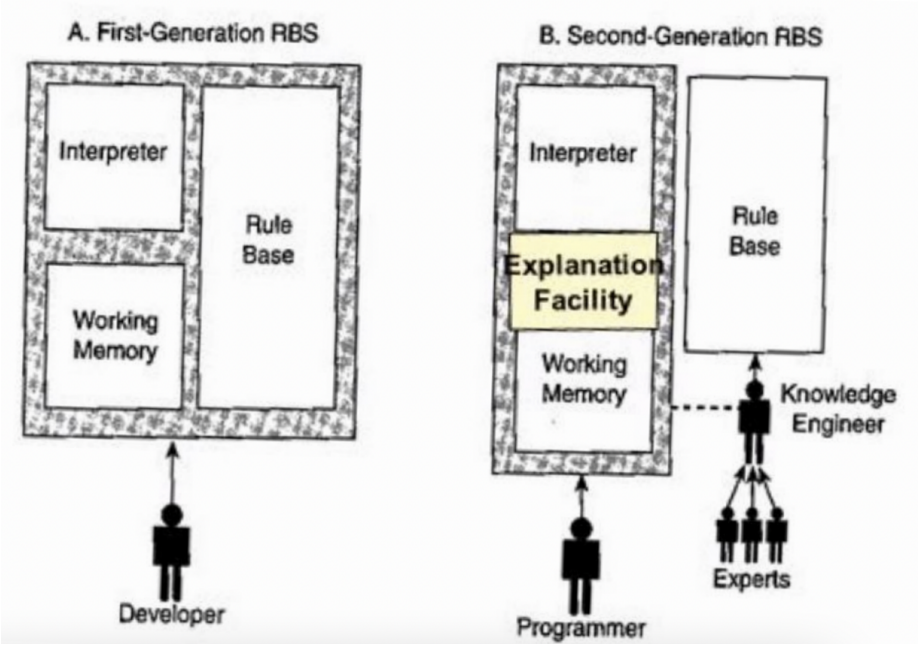




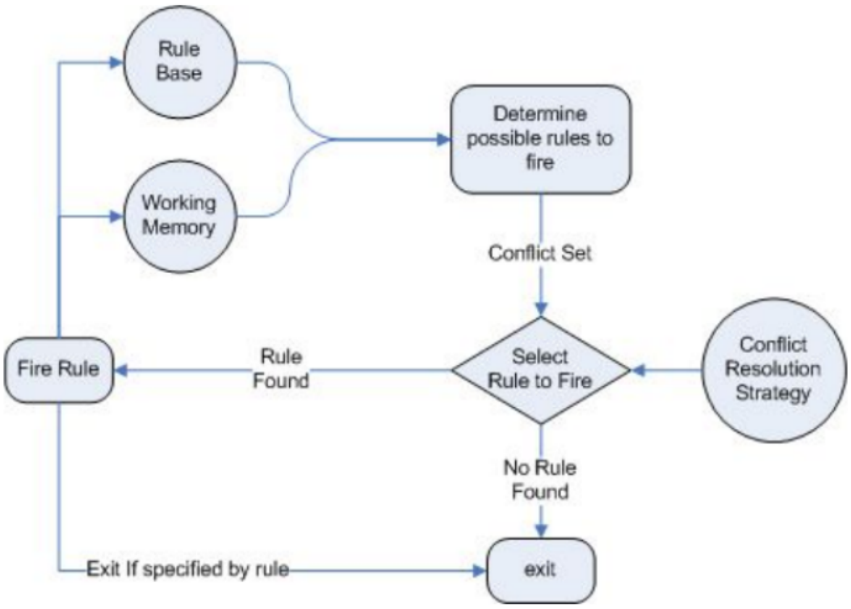
==========

Lecture 13

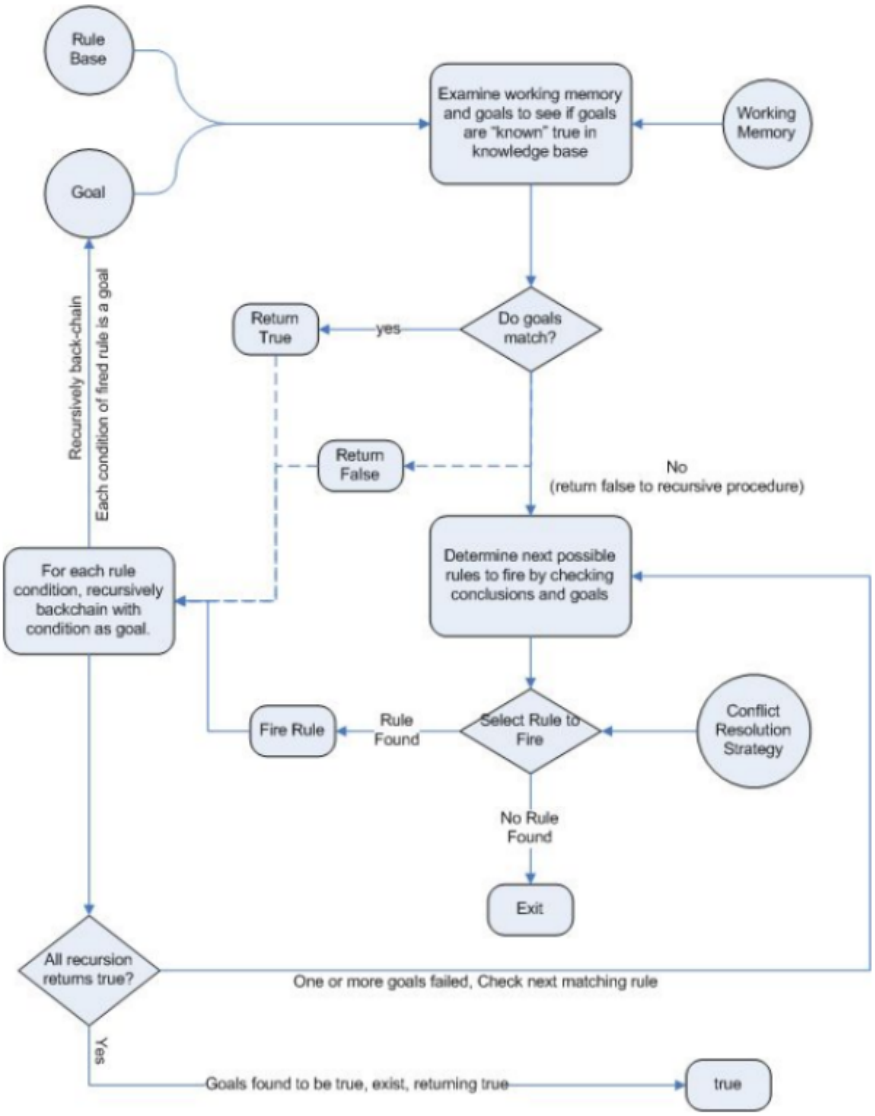
- **Knowledge Elicitation:** Encoding human knowledge into expert systems
- Difficult to elicit (obtain/extract) knowledge from expert systems:
    - Experts are often **busy and expensive**
    - Knowledge is **not easy to code**
    - Different experts prefer different facts
    - Facts and methods change over time



- People involved in RBS
    - **Domain Expert:**
        - MVP - Skilled, specialized problem-solver
        - Must communicate their knowledge and devote their time
        - Must be willing to develop the expert system
    - **Knowledge Engineer:**
        - Designing, building, and testing the expert system
        - Questions the Domain Expert for a solution to a problem
        - Decides knowledge representation and reasoning methods to handle facts in the expert system
        - Chooses software/language/expert system shell
        - Testing, revising, and integrating the system into the workplace
    - **Programmer:**
        - Actual programming - describing knowledge in computer-readable form
        - Needs symbolic programming skills - Prolog
        - Needs conventional programming skills - C, Pascal, FORTRAN, Basic
    - **Project Manager:**
        - Leader of development team; responsible for keeping the project on track
        - Makes sure deadlines are met
        - Interacts with all the other people involved
    - **End-User:**
        - Uses the system when it is developed



FORWARD CHAINING



BACKWARD CHAINING

- **Conflict Resolution**
  - Interpreter must choose one path - resolve conflicts in case of multiple rules
  - Methods:
    - **FCFS (Queue) -** choose the first rule that applies
    - **Priority Queue -** rules are ordered into priorities by expert; choose the highest rank
    - **Most specific rule -** apply the rule with the most elements in antecedent - most "beneficial"
    - **Random -** choose a rule at random
  - Keep a historical trace and allow this to choose a different rule next time
  - If the chosen rule leads to a dead-end, it must be possible to try another
- **Example of RBS**
  - Banking

        - Rules of charges for different services
        - Interest rate rules
        - Risk & Portfolio Management
        - Securities trading rules

    "If billing amount of customer for cellphone use per month is greater than ₹20,000/-, and m-commerce purchases are more than 5, grant 5% discount on the purchases"

```
IF  (Customer.BillAmount > 20000) AND
    (Customer.mTransactions > 5)
THEN
    SET mPurchase.discount TO 5%
```

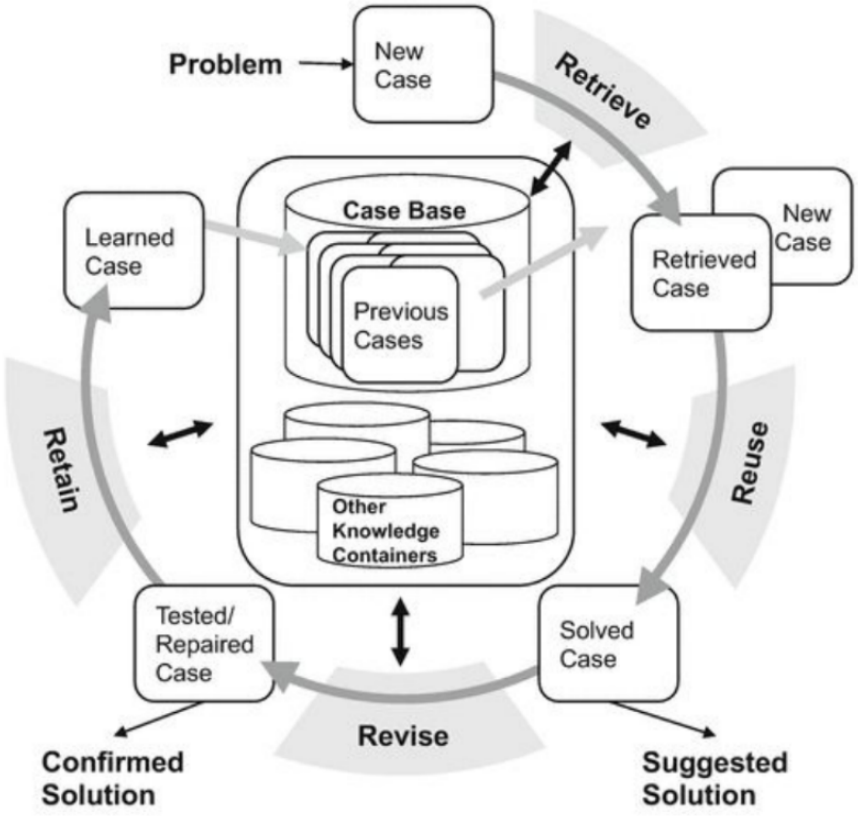
- **Disadvantages of RBS**
    - **Opaque relations between rules**
        - Individual rules are simple and concrete, but as size increases, their logical interactions/connections may not be clear
        - Difficult to observe how individual rules serve the overall strategy
    - **Ineffective search strategy**
        - In each cycle, the inference engine exhaustively searches all rules
        - Large set of rules (>100) - slow, unsuitable for real-time applications
    - **Inability to learn**
        - Do not learn from experiences
        - Cannot modify knowledge base - adjust existing rules or add new ones
- **Uncertainty Management**
    - Information can be **incomplete, inconsistent, or uncertain**
    - **Uncertainty:** lack of exact knowledge that enables us to reach a reliable conclusion
    - Classical logic allows exact reasoning only, but all real life facts aren't discrete
- **Solution: Certainty/Confidence Factor (cf) or Measure of belief (mb)**
    - cf [0, 1] OR [-1, 1], P: Proposition
    - **`cf(P1 "and" P2) = min(cf(P1), cf(P2))`**
    - **`cf(P1 "or" P2) = max(cf(P1), cf(P2))`**
    - **`p(A "or" B) = p(A) + p(B) - p(A "and" B)`**
    - For A→B, **`cf(P2) = cf(P1) * cf(P1→P2)`**
    - Example 1:
        - cf(IF it rains THEN John will catch a train) = 0.6
        - cf(it rains) = 0.5
        - cf(John will catch a train) = 0.5 * 0.6 = 0.3
    - Example 2:
        - cf(IF it rains AND I forget the umbrella THEN I'll be drenched) = 0.9
        - cf(I forget the umbrella) = 0.6
        - cf(it rains) = 0.7
        - cf(it rains AND I forget the umbrella) = min(0.7, 0.6) = 0.6
        - cf(I'll be drenched) = 0.6 * 0.9 = 0.54
    - This is similar to, but does not use Probability Theory because of its disadvantages

        - Difficult for experts to express real-life events as probabilities
        - Incorrect information of probabilities and uncertainties
        - Ex: to evaluate the certainty of (IF A or B THEN C), we need probabilities of A, B and the correlation between the occurrence of A and B
- **When (not) to use a Rule Engine:**
    - Rule engines are dynamic - rules can be stored, managed, updated
    - Work best when **declarative rules can be written**
    - Alternative: data-driven designs / lookup tables, case-based engines

==========

Lecture 14

- **Case-Based Reasoning/System (CBS)**
    - In RBS, rules need to be specific, but generic enough to include all cases
    - In CBS, reasoning is developed with cases
    - Problem-solving strategy that includes **maintaining a case-base**
    - **Case-Base:** a collection of prior cases or solutions to different problems
    - Has a cyclic nature; has the ability to learn
    - The solution from a CBS is approximate, as compared to an RBS which gives a definitive solution if the case matches a rule
    - Advantages:
        - Gets rid of the need for "Experts"
        - Problems and solutions tend to recur
        - Similar problems have similar solutions and require similar actions
        - Similar objects have similar properties



- **Cycle of Case-Based System**
    - **Problem-solving power improves over time**
    - Simple model of human memory
    - **Retrieve:** get the “closest” case to the problem case from the case base
    - **Reuse:** reuse an already defined solution from the case base which was used previously for another problem
    - **Revise:** revise the system if the current problem has a “better” solution
    - **Retrain:** learn the solution to new problems and retrain the system to include it
    - Properties of the cycle:
        - **Is Inexact:** computes similar correspondences; NOT exact
        - **Is not brittle:** retrieves the “nearest” possible solution
        - **Learns incrementally:** new cases are learned and reused as they arrive
        - **Retraining ability:** system improves over time
- Examples where a CBS is useful:
    - Situations that are **similar, but not identical**: Call Center
    - **Well-defined problem solving**: Legal Reasoning, Medical Diagnosis
    - **Rapidly changing situations**: Disaster Management
    - **Boundaries** between rules: Credit Risk Assignment, Loan Evaluation
    - **Inexact Reasoning** System: Job Placement
- Types of features of cases:
    - **Unindexed Features:** Not predictive and not used for retrieval (background information)
    - **Indexed Features:** Predictive and used for retrieval (usually numerical)

- **Similarity Matching**
  - Distance formula for difference between Target case and Source case

$$d(T,S) = \sum_{i=1}^{m} |T_i - S_i| w_i$$

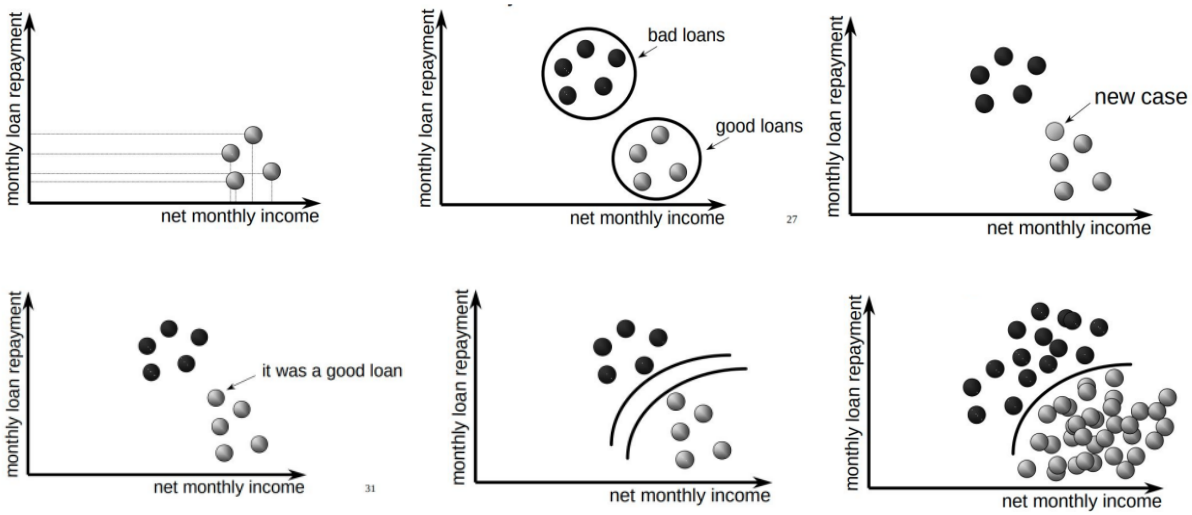
  - T_i: i-th field/feature of Target case
  - S_i: i-th field/feature of Source case
  - w_i: Weight (defined importance) of the i-th field/feature for the cases
- Example 1:

Case	Monthly Income (£K)	Account Balance (£K)	Home Owner	Credit Score
1	3	2	0	2
2	2	1	1	2
3	3	2	2	4
4	0	-1	0	0

Case	Monthly Income (£K)	Account Balance (£K)	Home Owner	Credit Score
5	3	1	2	?

$$\begin{aligned} d(T,S_1) &= |3-3| + |1-2| + |2-0| = 0+1+2 = 3 \\ d(T,S_2) &= |3-2| + |1-1| + |2-1| = 1+0+1 = 2 \\ d(T,S_3) &= |3-3| + |1-2| + |2-2| = 0+1+0 = 1 \\ d(T,S_4) &= |3-0| + |1+1| + |2-0| = 3+2+2 = 7 \end{aligned}$$

- Example 2:



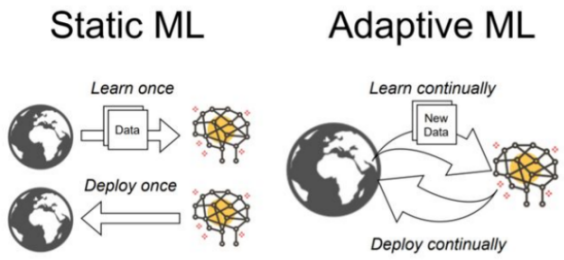
- **Issues with CBS**

- How many cases are needed - **large storage space**
- How to remove overlapping/similar cases - **large processing time** in case base
- How to **search efficiently:**
  - Create abstractions from cases
  - Multiple case bases
- What features to use for indexing
- How to weigh the features **(What weights to assign to w_is)**
- Cases may need to be created manually
- Adaptation may be difficult
- Give **reasonable/good solutions**, but **not optimal ones**

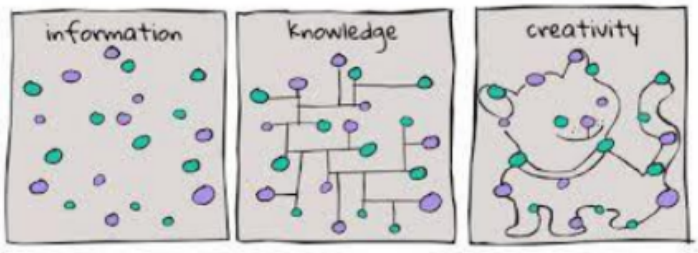
==========

Lecture 15

- **Learning**
  - An agent that improves performance on future tasks after making observations about the world
  - From a collection of I/O pairs, learn a function that predicts outputs for new inputs
  - Cannot anticipate all possible situations that the system will encounter
  - Cannot anticipate changes over time; adapt when conditions change
  - Methods of Learning:
    - From experience - has human intervention
    - From data - ML models
  - **Types of Machine Learning**
    - **Static:** Collect large amounts of data once and learn from it; deploy once
    - **Adaptive:** Continuous collection of data, modification of learning habits, and deploy the latest model

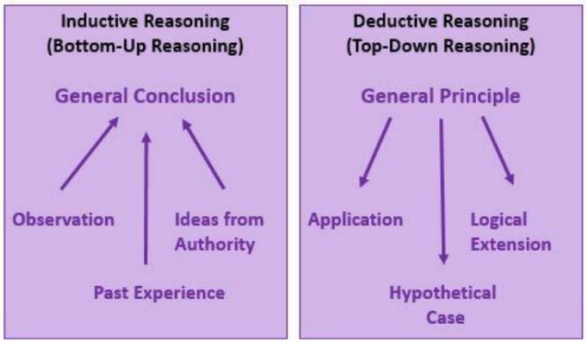


- **Types of Learning - four major factors:**
  - What components to improve?
  - What prior knowledge is already available?
  - What representation is used for the data?
  - What feedback is available to learn?



- **Knowledge-Based Learning**
  - **Inductive Reasoning (Bottom-up):** Infer a general rule or function from datasets of I/O pairs
  - **Deductive Reasoning (Top-down):** Start with a series of rules and infer new rules that allow more efficient processing
- **Feedback-Based Learning**
  - **Unsupervised Learning (No Feedback):**

        - No feedback is supplied
        - AI system **learns patterns in inputs**
        - Most common learning task: **Clustering** (Detecting potentially useful clusters of the input examples)

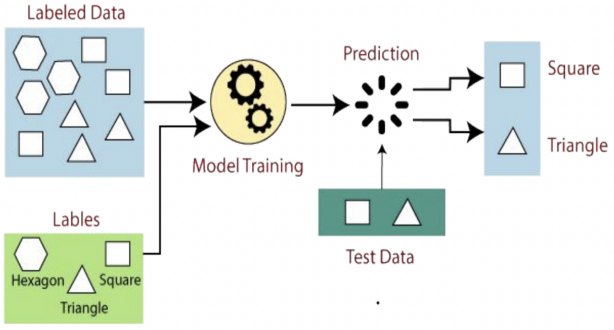


    - **Supervised Learning:**
        - Example **I/O pairs are given**
        - AI system **learns a function** that maps an input to its output
    - **Semi-Supervised Learning**
        - Given a **few labeled examples**
        - AI system must do what it can with **large sets of unlabeled examples**
    - **Reinforcement Learning**
        - AI system learns from a series of **reinforcements/feedback - rewards or punishments**

==========

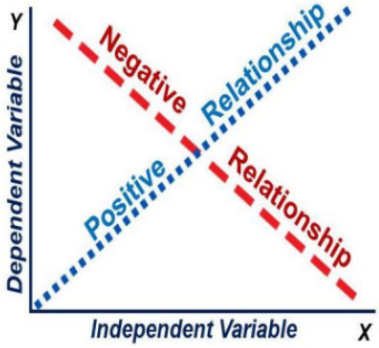
Lecture 16

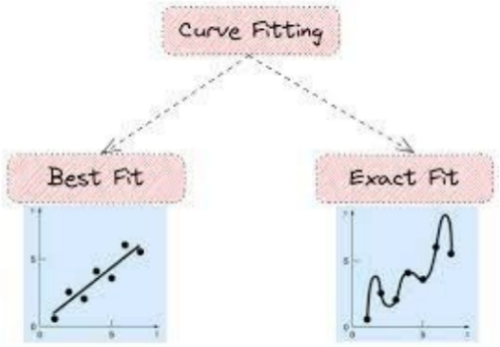
- **Supervised Learning**
    - Given a training set of N I/O pairs: $\{(x_1, y_1), (x_2, y_2), \ldots, (x , y)\}$
    - Task: discover a function **h(x)** that approximates the true function **f(x)** that maps each x_i to y_i
    - **f()** is unknown to the system; **h()** is a hypothesis
    - **Hypothesis is tested** across a set of examples **distinct from the training set** to **measure its accuracy**
    - Hypothesis is said to **"generalize well"** if it correctly predicts the **Y** value for new **X**s
    - Example:
        - What's wrong: **test set overlaps with training set**
        - To make the model better, rotate the shape or give only part of a shape or incomplete shape
        - A good model can identify patterns and generalize them to approximate the result

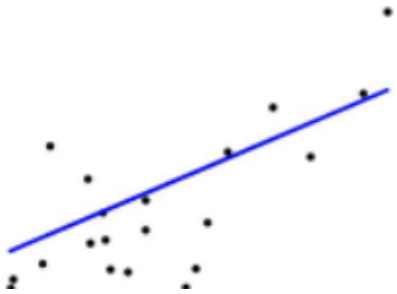


    - Types of learning problems:
        - **Classification:** Output is from a finite set of values (ex: sunny, rainy, cloudy)
        - **Regression:** Output is a number (ex: house price)
    - **Stochastic function f(x):**
        - **f()** is not strictly a function of **x**
        - Need to learn a conditional probability distribution **P[Y|X]**

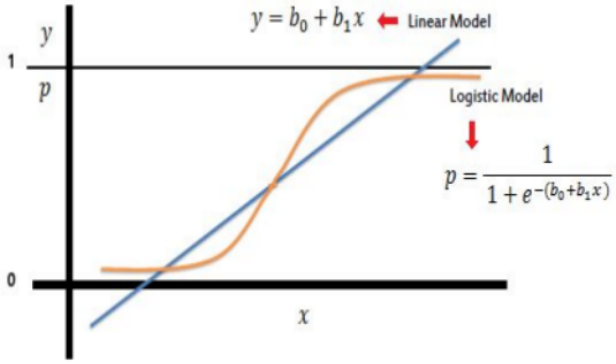
- **Regression Analysis:**
  - **Model:** High-level mathematical concept used to describe behaviour
  - **Algorithm:** A process (usually involves data structures)
  - It is a statistical technique for **estimating the relationship** between dependent and independent explanatory variables
  - Curve-fitting **under context** (dependencies between variables)

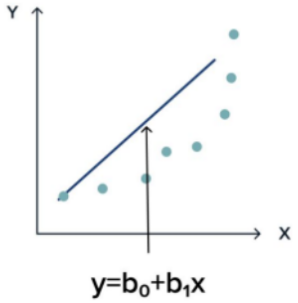


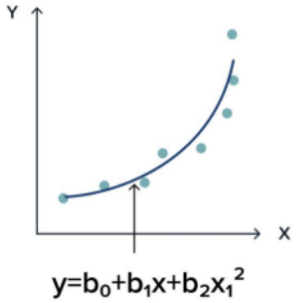




- **Types of Regression**
  - # of independent variables: **Univariate or Multivariate**
  - **Linear Regression:** model is to fit the data on a line $y = mx + c$
  - **Polynomial Regression:**
    - Model is to fit the data on a polynomial curve $y = a\ x^n + \ldots + a_1x + a_0$
    - Linear Regression ⊂ Polynomial Regression
  - **Logistic Regression:**
    - **Dependent variable** is (almost) **discrete [0/1, True/False]**
    - May be an S-curve (as shown)
  - Other variants:
    - Ridge Regression
    - LASSO Regression
    - Bayesian Regression







- **Univariate Linear Regression**
  - To fit all data points on the closest line **(find weights "m" and "c")**
  - Find the value of $b = [m, c] = [b_0, b_1]$ that minimize total loss

- Loss (**y*:** ideal output; **y:** predicted output):
  - Absolute Loss: $|y^* - y|$
  - Square Loss: $|y^* - y|^2$
  - Total Loss: Loss over all I/O pairs
  - Minimized Loss: $b^* = \text{argmin}(\Sigma(y_i^* - y_i)^2)$
- **Optimal solution:**
  - minimize square loss over chosen b_0, b_1
  - Loss is stb minimized when its partial derivatives wrt each variable of interest (b_0, b_1) are zero (separately)
    - $\partial \sum(y_i^* - y_i)^2 / \partial b_0 = 0$
    - $\partial \sum(y_i^* - y_i)^2 / \partial b_1 = 0$

- **Multivariate Linear Regression**
  - Consider the following vectors and matrices:
    - $b = [b_0\ b_1\ \cdots\ b\ ]^T$ AND $y = [y_1\ y_2\ \ldots\ y\ ]^T$
    - $X = [X_1\ X_2\ \ldots\ X\ ]^T$, where $X_i = [1\ x_{i1}\ x_{i2}\ \ldots\ x_i\ ]$
  - Then:
    - $y = Xb = x_ib = b_0 + b_1x_{i1} + \ldots + b\ x_i$
    - $y = Xb + e$, where **e** is an error
  - When $y = Xb$, then $b = X^{-1}y$
  - This is not always possible, as inverse of a rectangular matrix X does not exist
    - $b = (X^TX)^{-1}X^Ty$ (Moore's pseudo-inverse)
  - Ordinary Least Square (OLS) Estimate: $b = (X^TX)^{-1}X^Ty$
    - Unbiased Estimator
    - What happens if you have correlated variables - ask sir
  - Gradient Descent: $b_{new} = b_{old} - \eta\ \nabla\ loss$
  - Gradient Descent is a better approach; Computing OLSE is a time-heavy process

$b = [b_0, b_1, b_2, b_3 \ldots\ldots]^T$

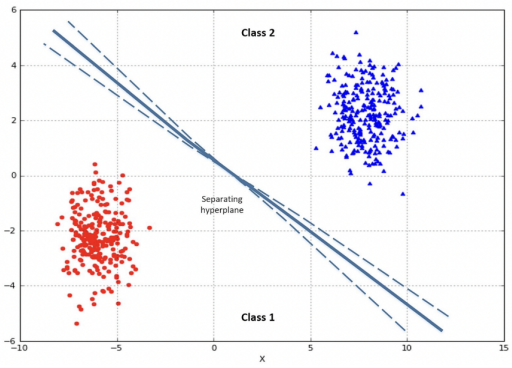
- $y = b_0 + b_1 x_{j1} + b_2 x_{j2} + \ldots\ldots\ldots = [1\ x_{j1}\ x_{j2} \ldots.]\ b$
- $y = x_j b \qquad y = Xb + e$
- $b = (X^TX)^{-1}\ X^Ty$ **Ordinary Least-square Estimate**
- $b_{new} = b_{old} - \eta\ \nabla\ loss$ **Gradient Descend**

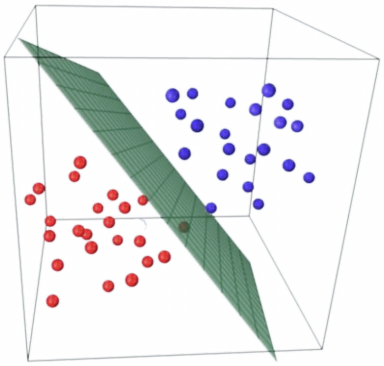
$$X^TX = \begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix} \qquad X^Ty = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{bmatrix}$$

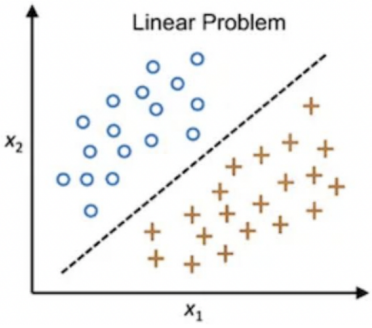
==========

Lecture 17

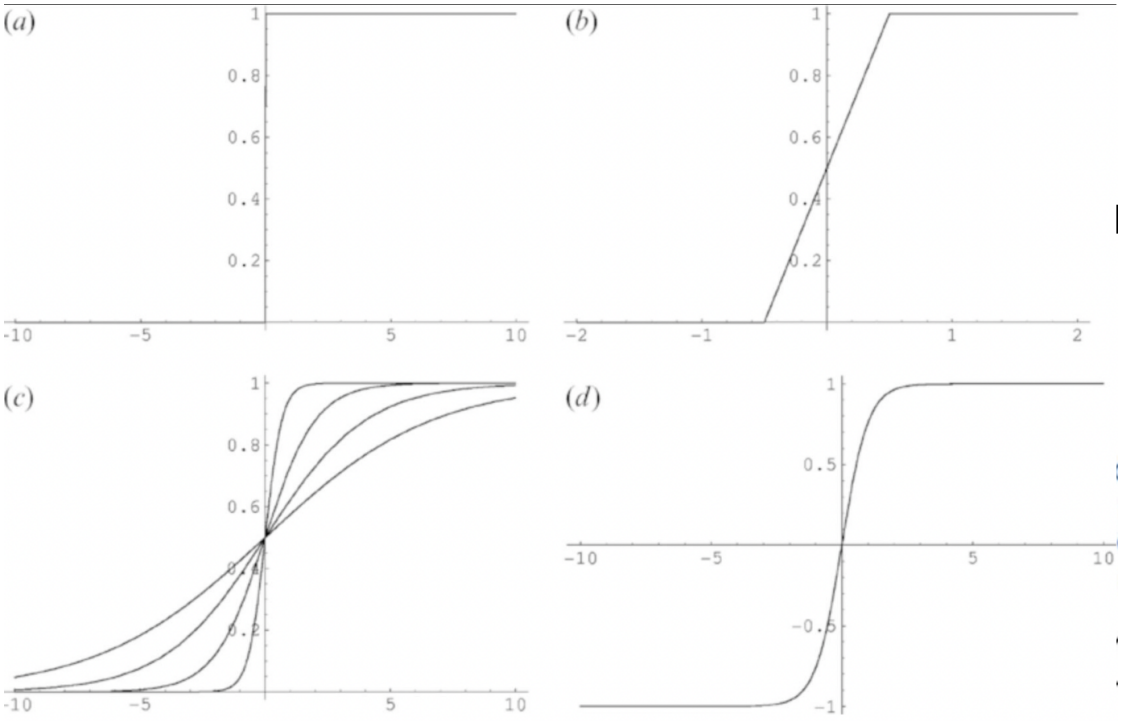
- **Logistic Regression**
  - Estimating the probability of occurrence of an event
  - **`0 <= Dependent variable <= 1`** since it is a probability
  - Most common: Binary Classification - **Linearly separable decision boundary** i.e. clusters of data are separated by a straight line





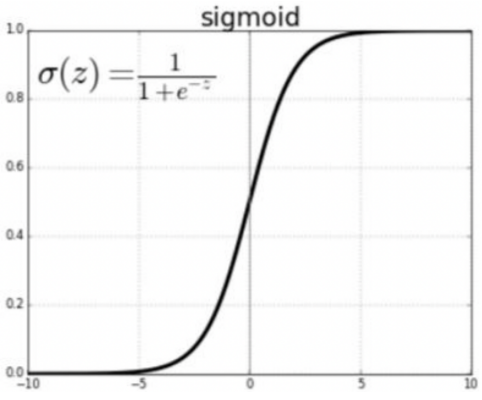


  - Define two functions: **`y = x₁+c+e`** and **`y = x₂-c+e`** (approximately linearly related)
  - **`y = Xb = [1 x₁ x₂][b₀ b₁ b₂]ᵀ`**
  - Classification hypothesis: **`h(x) := 1 if Xb > 0 else 0`**



  - Graph resembles a threshold function; threshold is the point beyond which the **`f(x) = 1`** and before which **`f(x) = 0`**
  - Analysis of the graphs:
    - *(a)* Hard Threshold at **`x = 0`**: function switches from 0 to 1 abruptly at **`x = 0`**

        - *(b)* Soft Threshold at **x = 0**: threshold is established gradually - function decides the threshold while traversing the slope
            - Has points of discontinuity
            - Gradient of the function at those points is undefined
    - S-curves *(c)* and *(d)*:
        - Have different gradients depending upon discreteness of the variable
        - Precision of graph: slope during which graph goes from 0 to 1
        - Equivalent to sigmoid function: $y = 1/(1+e^{-x})$
        - General sigmoid function: $y = 1/(1+Ae^{-Bx})$
        - For **B > 0**: function **[0 → L]** over the threshold
        - For **B < 0**: function **[L → 0]** over the threshold



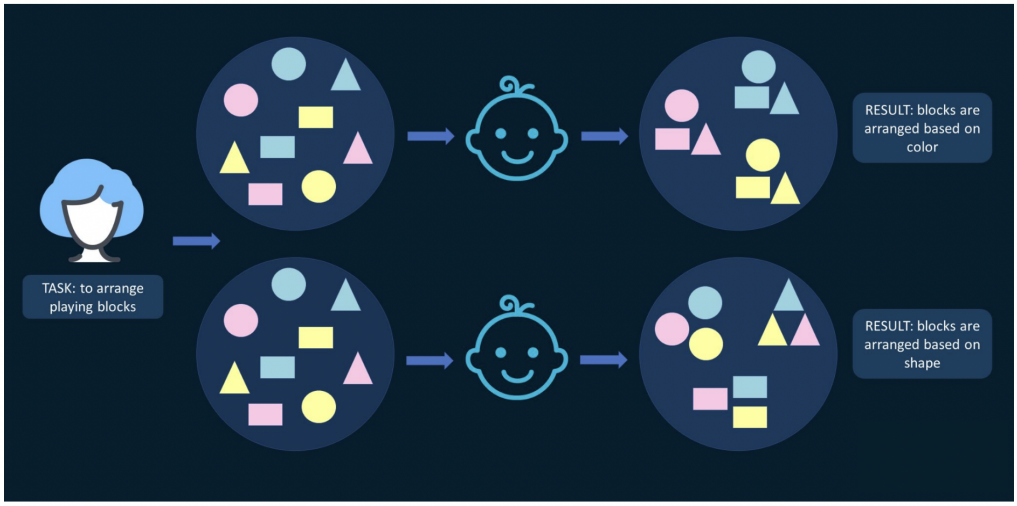
$$Logistic(z) = \frac{1}{1+e^{-z}}$$

- **Logistic Regression Model Update**

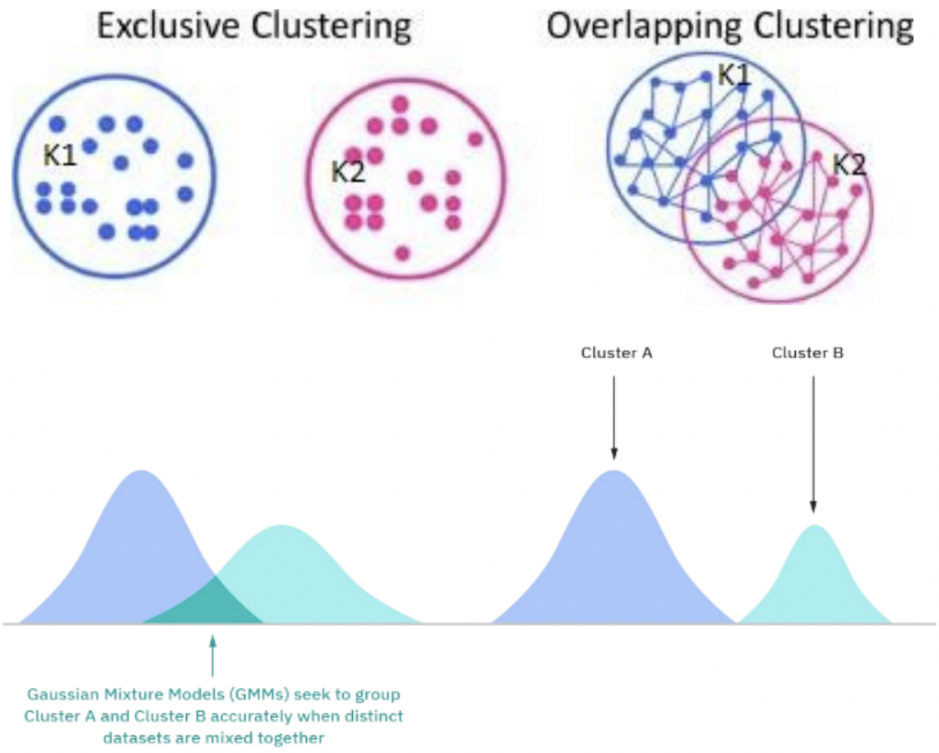
$= \partial (y^* - h_b(x))^2 / \partial b$
$= 2(y^* - h_b(x)) [\partial (y^* - h_b(x)) / \partial b]$
$= 2(y^* - h_b(x)) [-\partial (h_b(x)) / \partial b] [\partial xb / \partial b]$
$= 2(y^* - h_b(x)) [-\partial (h_b(x)) / \partial b] x$
$= 2(y^* - h_b(x)) [h_b(x) (1 - h_b(x))] x$

≡ (1+e^-z)^-1 when you take differential then you get (-1(1+e^-z)^-2) (-e^-z) which is ≡ (1+e^-z)^-2) (e^-z) ≡ (1+e^-z)^-1 (1+e^-z)^-1 (e^-z) which is ≡ g(z)(1 – g(z))

- **Unsupervised Learning**
    - Infers patterns from historical data
    - Does not require human intervention; the **machine identifies** similarities and differences in the dataset **by itself**
    - **Includes:** Clustering, Association, and Dimensionality Reduction
    - **Used in:** Data Exploration, Anomaly Detection, Recommender Systems etc.
    - Disadvantages:
        - **High time complexity:** it needs to iterate over the entire dataset
        - Takes longer to train
        - **More chances of inaccuracies**
        - **Need to validate** the output of the model manually
        - Basis of Classification is unclear (example given)



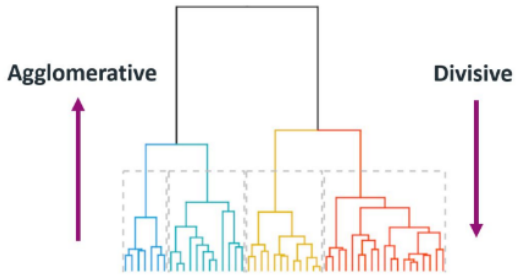
- **Clustering**
  - **Exclusive/Hard:** each piece of data belongs to one cluster only
  - **Overlapping/Soft:** some data belong to multiple clusters; degree of belonging of a data piece is the # of clusters it belongs to
  - **Hierarchical:** clusters are arranged in a hierarchy
  - **Probabilistic:** density estimation; includes Gaussian variables which appear to form clusters in the form of a probability graph



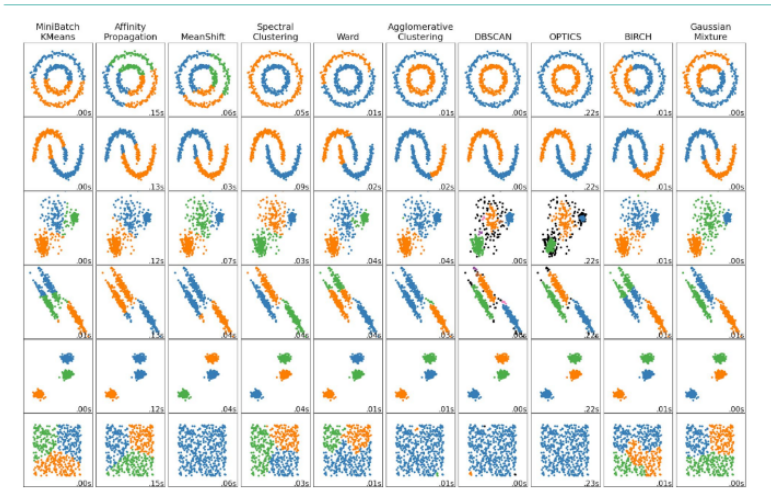
==========

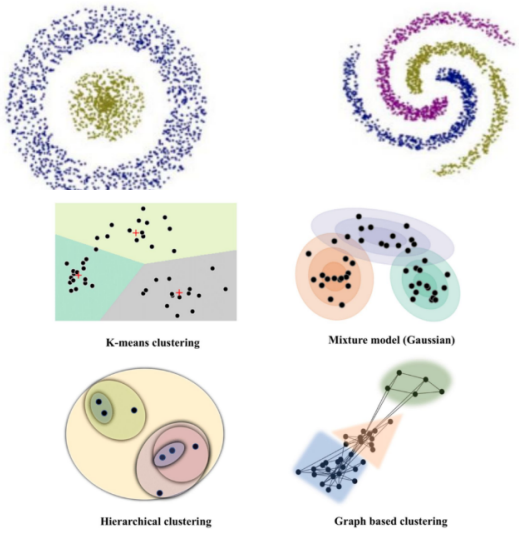
Lecture 18

- Given the advantage of the datasets (clusters) being far apart in the case of unsupervised learning, the task of drawing a distinction between these clusters is relatively easier as compared to supervised learning, for ex: can use distance to a root point (mean point) as a criteria to determine the belonging of each data point
- Hyper parameters can broadly classify all data points into certain initial clusters which can be further fine tuned to establish more definitive clusters - decides the number of clusters
- Example: for people purchasing from a supermarket, they could buy fruits or vegetables but these sort of clusters have overlaps (people buying both); hence, **probabilistic clustering (fuzzy division)**



- Further, in the same example as above, define hierarchical clustering as subcategories of each cluster of fruits and vegetables, for ex: vegetables that grow below the ground and vegetables that grow above the ground
- **K-means Clustering**
    - Identify **k** clusters and **locate k centroids (mean points)**
    - Classify points into clusters depending on their distance from the centroids (generally the data point is made part of the cluster to the centroid of which, its distance is minimum)
- **Mixture Model (Guassian)**



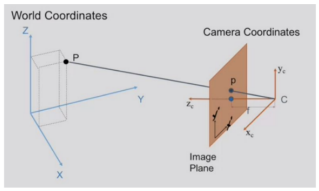


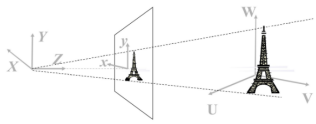
==========

Lecture 19 (Computer Vision)

- A scientific field that deals with **extracting information from digital images**
- Origin: MIT undergrad summer project, 1966
- **Applications:** 3D urban modelling, scene reconstruction, face recognition, visual search, self-driving cars, AR/VR etc.
- Science → Images (a by-product of complex transformations) → Image processing (understand how the image was generated) → Vision

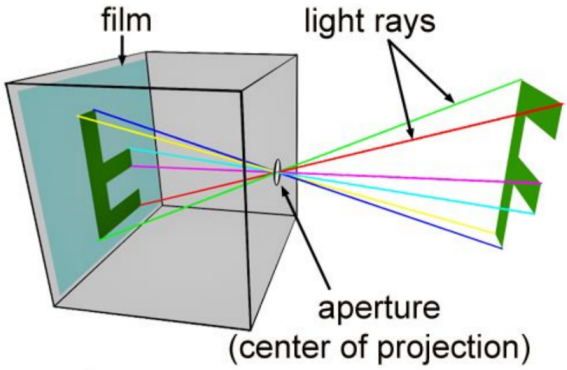
- **Image Geometry:** how to map a 3D world onto a 2D image
- **Types of images: colour, grayscale, depth**
- Camera: any sensor that is able to capture an image (even MRI scanners)
- Computers see a 200x200 image as a list (ordered set) of 120000 values
- Combine information from multiple images - grouping pixels to represent real objects
- **Issues:**
    - How do sensors capture images?
    - How are attributes like texture, shape, motion, and colour encoded?
    - Which algorithms to use to store image data
    - How to decipher patterns in different images?



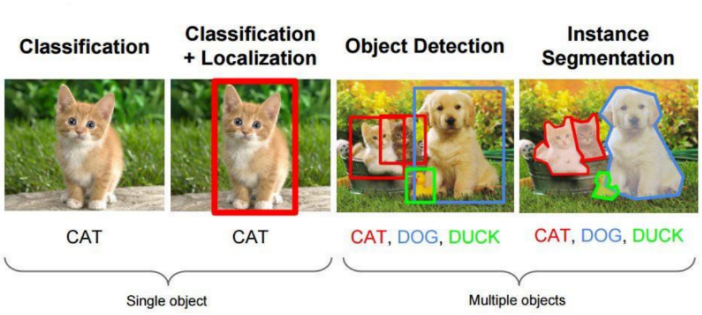


3D to 2D Projection! We lose information

How we are oriented to the world?



- **Example Tasks:**
    - High Level: Recognition, understanding and integration
    - Medium Level: Feature extraction, motion, and image segmentation
    - Low Level: Pre-processing, Image enhancement, filtering, and simple transformations



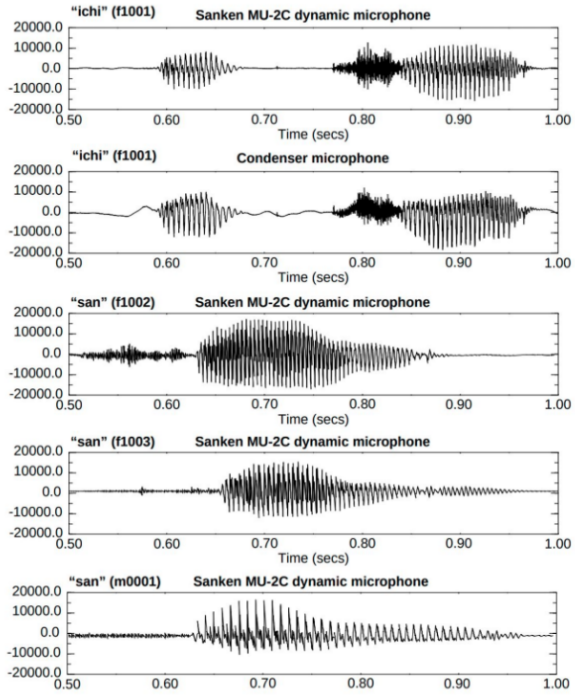
- **Applications:**
    - Vision-based HCI:
      Gesture/eye-based control, face biometrics, eye/head movement tracking
    - GIS: Satellite imagery
    - Medical Imaging: MRI, CT, X-Ray
    - Video Surveillance: Traffic, License Plate Recognition

==========

Lecture 20 (Speech)

- **Speech processing: application of signal processing techniques** to processing and analysis of speech signals
- **Speaker → Signal Processing → Humans/Machines**
- Speech processing deals with how to make machines learn and develop useful applications

- **Superset of NLP**
- **Audio Recording and Playback:**
    - Audio → Mic → Amplifier → Recorder → Playback
- Different types of audio formats are just results of different processing techniques; some are bound to be “better” than others (.wav vs .mp3)
- **Types of microphones:** carbon, piezo-election, dynamic-moving coil
- **Types of recordings:** omnidirectional, unidirectional, bidirectional
- Playback: Loud-speaker, headsets
- **Speech production:**
    - Mic being far from mouth results in a lot of noise; an important factor
    - Distortion and clipping are results of noise
    - Speech is composed of sequence of sound units and transitions between them; basically a signal
    - Symbolic representation of speech information is important to decode a message



- **Linguistics:** the study of **rules of language**
- **Phonetics:** the study of **sounds of speech**
- Speech may also reveal hidden information like linguistic identity, mother tongue, mental health, race, age, sex, education level, religious orientation, background etc.
- **Example Tasks:**
    - Voice activity detection
    - Pitch detection/source modelling (excitation_
    - Filter formants/system modelling
    - Time/Frequency domain processing
- **Applications:**
    - Speech to text (ASR) and text to speech (TTS)
    - Speech-to-speech and speech enhancement
    - Speech coding
    - Speaker identification, verification, and diarization
    - Sound source separation and noise reduction

==========
==========

COURSE ENDED