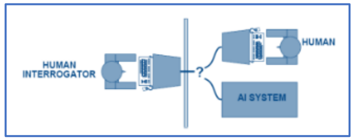
Introduction / What is AI?

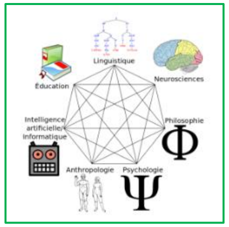
Definitions of AI

Acting humanly — The Turing test



- Can machines think / behave intelligently?
- Operational test for intelligent behaviour

Thinking humanly — Cognitive science



- Understanding the human mind by computer modelling
- Requires scientific theories of internal brain activities

Thinking rationally — Laws of thought

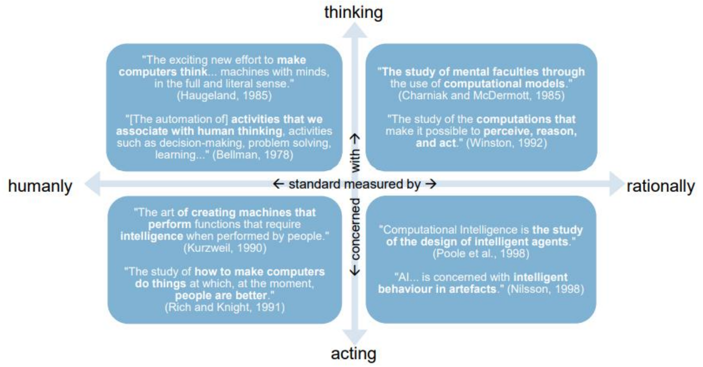
- How to make provably correct inference?
- Several forms of logic have been developed

Acting rationally — Rational behaviour

- Doing the right thing
  (expected to maximize goal achievement, given the available information)

Rational agents

- A practical way and goal of this course
- For any given class of environments and tasks, we seek the agent with the best performance



Example Definition

- AI: The scientific field concerned with generating intelligent behavior by computers.
- Intelligence: The power to solve previously unsolved problems.

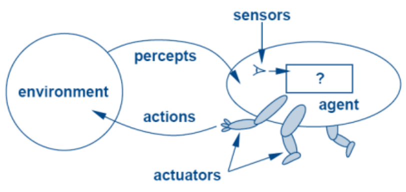
Overview

- Introduction
  - The field of AI
  - Intelligent agents
- Search
  - Problems solving through search
  - Local and adversial search
  - Constraint satisfaction problems
- Planning
  - Knowledge reasoning & logic
  - Datalog
  - Planning
- Learning
  - Supervised learning with neural networks
  - Unsupervised learning with autoencoders
  - Generative adversarial learning for image synthesis
- Other
  - Reinforcement learning for game play
  - AI & society

Intelligent Agent

A rational agent

For each possible *percept sequence*, a *rational agent* should select an *action* that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the *agent* has.



- *Agents* include humans, robots, softbots, thermostats, …
- The *agent function* maps from *percept sequence* to *actions* $f: P^* \rightarrow A$
- The *agent program* runs on the physical architecture to produce f

Rationality

Requires a fixed **P***erformance measure* to evaluate environment sequence.

Task Environment = PEAS (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors)

To design a rational agent, we must specify the task environment. PEAS specifies environment types: how we humans would perceive external features.

PEAS Example: The task of designing an automated taxi

- **P**erformance — safety, destination, profits, comfort, legality
- **E**nvironment — streets/freeways, traffic, pedestrians, weather
- **A**ctuators — steering, acceleration, brake, horn, speaker/display
- **S**ensors — video, accelerometers, lidar, GPS, engine sensors

Environmental Properties

- Fully observable vs Partially observable — Do sensors give full access to the *relevant* state of the environment?
- Single agent vs Multiagent — Do others optimize a performance measure dependent on our agent?
- Deterministic vs Nondeterministic — Do actions have certain consequences, or is the outcome probabilistic (other's actions don't count)?
- Episodic vs Sequential — Do current actions influence future decisions (probably not in classification settings)?
- Static vs Dynamic — Does the world keep turning while our agent decides what to do?
- Discrete vs Continuous — Regarding states, time, percepts and actions
- Known vs Unknown — Are the rules/laws governing the environment known to the agent?

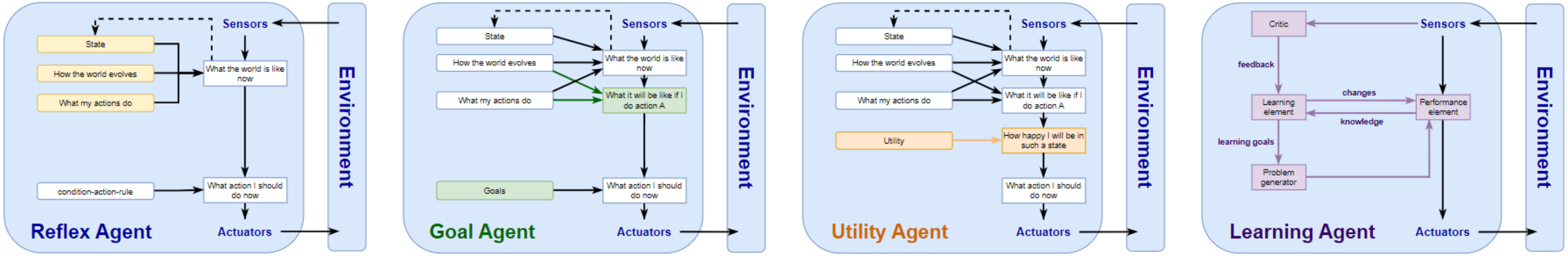
Examples: Environments and their properties

	Solitaire	Poker	Image analysis	Internet shopping	Taxi
Observable?	(x)	(x)	x	-	-
Single-agent?	x	-	x	x (except auctions)	-
Deterministic?	x	- (stochastic)	x	(x)	-
Episodic?	-	-	x	-	-
Static?	x	x	(x)	(x)	-
Discrete?	x	x	-	x	-

Four basic agent types

- Simple *reflex agents*: select action based on last percept
- *Reflex* agents *with state*: regard history
- *Goal-based* agents
- *Utility-based* agents

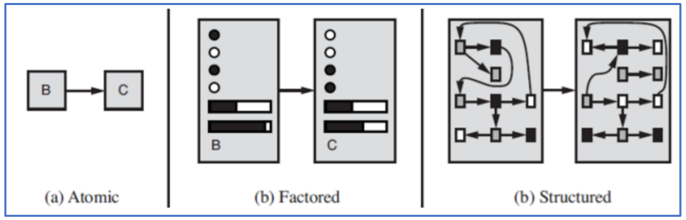
All these can be turned into learning agents!



A representation taxonomy

Consider the representation of any building block

- Atomic states are "just different" from each other
  - Search, game playing
- Factored states described by vectors (of attributes)
  - Constrained satisfaction, propositional logic, planning, machine learning
- Structured states as entities and their relationship with each other
  - First-order logic, first-order probability models, knowledge-based learning



(a) Atomic (b) Factored (b) Structured

Atomic → Factored → Structured is ordered by expressiveness. A more capable agent (more expressive agent) is not always better.

- More expressive – Advantages: Captures more, often much more concise
- More expressive – Disadvantages: Learning/reasoning becomes much harder

Problem solving through search

Problem formulation

- Real world is complex → State and Action must be abstracted
- Each abstraction should be easier than the original problem

A search problem can formally be defined as follows

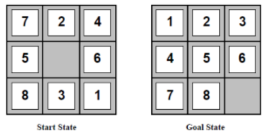
- *State space* Set of possible states
- *Initial state* Starting state
- *Goal states* Possible goal states
- *Actions* Actions available to the agent
- *Transition model* Describes what each action does
- *Action cost function* Function to determine the cost of an action

Strategy building

- Initial state
- Formulate goal
- Formulate problem (states and actions)
- Find solution

<u>8-Puzzle - Problem formulation</u>

- States Describes the location of each of the tiles (e.g. with an Int)
- Actions Move blank: LEFT, UP, DOWN, RIGHT
- Goal states All tiles in order
- Action cost 1 per action



Diversity of search approaches

- Uninformed (blind) search
  - All it can do: generate successors of tree-nodes
- Heuristic (informed) search
  - Knows whether one non-goal state is «more promising»
- Online search
  - Environments are dynamic
- Local search
  - Cares only to find a goal state rather than the optimal path
- Adversarial search
  - Search in the face of an opponent

Remarks

Heuristics

- An admissible heuristic is one that *never overestimates* the cost to reach a goal.
- Good heuristics can dramatically reduce search cost

Other

- Iterative deepening only uses linear space and not much more time than other uninformed algorithms

Problem solving through search

Uninformed (blind) **search**

- All it can do is generate successors of tree-nodes
- Distinguish goal- from non-goal states
- Suitable environments: fully observable, deterministic, discrete

Approach

- Tree search — Iteratively expand nodes until a goal node is hit
- Different Strategies — Order of node expansion

Evaluation criteria for strategies

- Completeness — does it always find a solution if one exists?
- Optimality — does it always find a *least-cost solution*?
- Time complexity — number of *nodes expanded*/generated
- Space complexity — maximum number of *nodes in memory*

Time and space complexity are measured in terms of

- b: Maximum *branching* factor
- d: *Depth* of the least cost solution
- m: *Maximum depth* of the state space

TODO: More details

Expand the shallowest unexpanded node
Expand node with lowest path cost $g(n)$
Expand deepest node
DFS only up to level l
Try DLS with $l = 1, l = 2, \ldots$ until goal is reached

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Heuristic (Informed) search

- Knows whether one non-goal state is «more promising»
- Suitable environments: Similar to uninformed search, but larger

Approach

- Tree- / graph search using additional knowledge beyond the definition of the problem.

Best-first search

- Select the node to be expanded next based on some evaluation function
- Typically, f is implemented by some heuristic

Greedy search

- Expand node with lowest subsequent cost estimate according to some h, i.e. $f(n) = h(n)$
- n may only appear to be closest to the goal

A*

- Obvious improvement, consider full path cost: $f(n) = g(n) + h(n)$
- $h(n)$ needs to be admissible
- Optimal and complete
- Complexity $O(2^{(error\ of\ h)\cdot d})$, keeps all nodes in memory

SMA* - simplified memory-bounded A*

- A* usually runs out of space first → SMA* overcomes this by
  - Fill up memory → forget the worst expanded nodes
  - Ancestors of forgotten subtrees remember the value of the best path within them
  - Thus, subtrees are only regenerated if no better solution exists

Local Search

Search for *optimal states* instead of paths.

- In many optimization problems the *path is irrelevant*, the *goal state* itself is the solution.

Iterative improvement algorithms are used to solve such problems

- Keep a single "current" state and try to improve it
- Constant memory usage, suitable for online and offline search

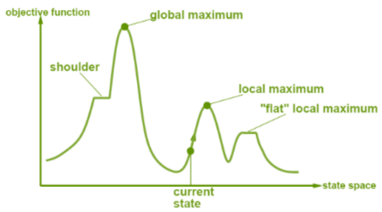
Hill climbing search (gradient ascent / descent)

Systematic search for an optimum

- Finds a state that is a *local maximum*, by selecting the highest valued successor iff its value is better than the current value.

The state space landscape

- Practical problems typically have an exponential number of local maxima
- *Random-Restart* hill climbing overcomes local maxima
- *Random sideways* moves escape from shoulders and loop on flat maxima (bad)



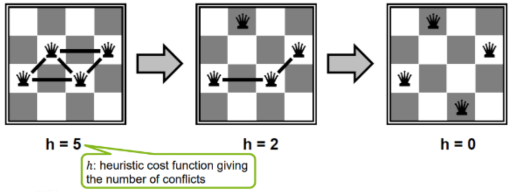
Simulated annealing: *Optimizing hill climbing search*

Idea — Escape local maxima by *allowing some "bad"* moves but gradually decrease their size and frequency.

Application — For *good schedule* of decreasing the temperature, it *always reaches the best state*. Widely applied for VLSI layout and airline scheduling.

Example: $n-$ queens problem

Put n queens on a $n \times n$ board with no two queens on the same row, column or diagonal.



Possible solution

- Initialize one queen per column
- Move one queen up/down at a time to reduce number of conflicts using heuristics h
- Almost always solves $n-$ queens problem almost instantaneously, even for large n, e.g. $n = 1'000'000$

Local beam search: *Optimizing hill climbing search*

- Keep k states instead of 1; choose top k of all their successors
- Choose k successors randomly, biased towards good ones

Problem — Often, all k states end up on the same local hill

Solution — Choose k successors randomly, biased towards good ones

Genetic algorithms (GA): *Improve on the idea of local beam search*

Idea — Combine stochastic local beam search + generating successors from pairs of states.

Application

- Require states encoded as strings
- Crossover helps iif *substrings are meaningful* components
- GAs ≠ evolution

Adversarial Search

Adversarial search

- Unpredictable opponent — Specify a move for every possible opponent reply
- Time limits — Unlikely to find goal → must approximate

Types of Games

	deterministic	stochastic
perfect information	**chess**, checkers («Dame»), go, othello («Reversi»)	backgammon, **monopoly**
only partial observability	**battleship**, kriegspiel (chess without seeing enemy pieces)	bridge (~ «Jass», «Skat»), **poker**, scrabble, *global thermonuclear war*

Minimax: *depth-first exploration of game tree*

Optimal strategy for a given game tree.

- MAX — 1st player — Wants to maximize utility of terminal states
- MIN — 2nd player — Wants to minimize (Max's) utility
- Utility — Numeric value ("payoff") of terminal state

Idea

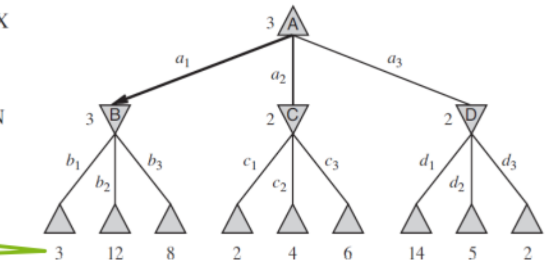
- Choose a move to position with highest *minmax* value
- Minimax value: highest value among options minimized by adversary
  - Best achievable payoff against best play

<u>Example</u>

- Any 2-Player game tree (each player moves once)
- MAX's best move at root a_1 because MIN's best reply will be b_1

Properties

- Complete (finite tree)
- Optimal (Optimal opponent)
- Time complexity $O(b^m)$
- Space complexity $O(bm)$



Min is going to do the least valuable thing here for Max (3).

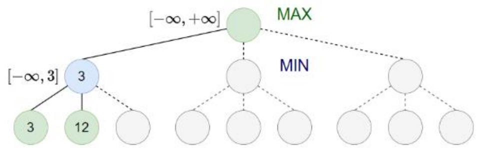
$\alpha - \beta$ pruning: *Overcoming exponential (b^m) number of states*

Successively tightening bounds on minmax values

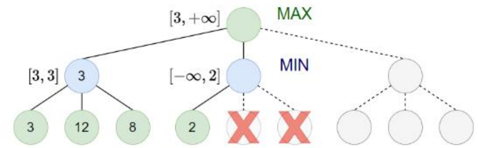
- α is the *best value* (to MAX) found so far in current subtree of a MAX node
- If any node v is worse than α, MAX will not choose it → *prune branch*
- Similarly: β is the best score MIN is assured of in current subtree of a MIN node

<u>Example</u>

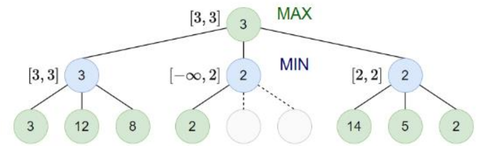
1. Root: $[\alpha, \beta] = [-\infty, +\infty]$



2. Root: $[\alpha, \beta] = [3, +\infty]$



3. Root: $[\alpha, \beta] = [3,3]$



Properties

- Pruning does not affect final results
- Good move ordering improves effectiveness of pruning
- Time complexity with «perfect ordering» = $O(b^{m/2})$

Resource Limits: *Towards real-world conditions*

Standard approach

- Use *Cutoff-Test* instead of Terminal-Test, e.g. depth limit
- Use *Evaluation* (heuristic) function instead of Utility
- *Lookup* of start/end games

Eval(uation) function: *Designing or learning effective cutoff tests*

For chess, typically linear weighted sum of *features*

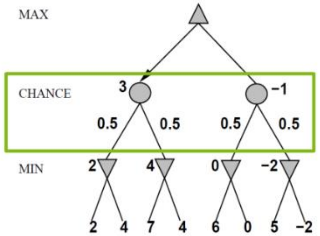
- $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$
- Example: $w_1 = 9, f_1(s) = count_{white-queens} - count_{black-queens}$
- Can be learned with machine learning techniques

Nondeterministic (stochastic) games

Chance is introduced by dice-rolling or card-shuffling…

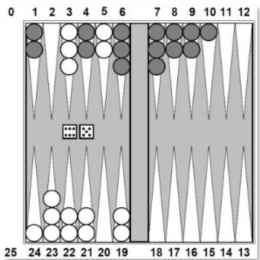
<u>Simplified example</u>

- A game with coin-flipping
- Nondeterminism is handled by an additional level in the tree, consisting of *chance nodes*



<u>Real-world example</u>

- 2048: Numbers appear with probabilities at random board positions
- Backgammon: Before each move, dice-rolls determine the legal moves



Expectiminimax – maximizing the expected value

- Words just like *minimax* – except chance-nodes are also handled
- Expectiminimax gives *perfect play*
- In case of only *1 player* → Expectimax
- Time complexity: $O(b^m n^m)$
  - $n =$ number of distinct random events
  - Possibilities are multiplied enormously in games of chance
  - No likely sequences exist to do effective $\alpha - \beta$ pruning

```
function ExpectiMinimax-Decision(state) returns an action
    inputs: state, current state in game
    return a in Actions(state) maximizing
           ExpectiMinimax-Value(Result(a, state))

function ExpectiMinimax-Value(state) returns a utility value
    if Terminal-Test(state) then
        return Utility(state)
    if state is a Max node then
        return highest ExpectiMinimax-Value of Successors(state)
    if state is a Min node then
        return lowest ExpectiMinimax-Value of Successors(state)
    if state is a chance node then
        return average of ExpectiMinimax-Value of Successors(state)
```

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\,\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

Constraint satisfaction problems (CSP)

Allows useful general-purpose algorithms with more power than standard search.

Components of a CSP

- X set of variables $\{X_1, \dots, X_n\}$
- D set of domains $\{D_1, \dots, D_n\}$ consists of allowed values $\{v_1, \dots, v_k\}$ for X_i
- C set of constraints consists of a pair $\langle scope, rel \rangle$ (scope = tuple of variables, rel = relation)

Varieties of CSPs

Discrete variables

- Finite domains of size d
- Requires constraint language

Continuous variables

- E.g. precise start/end times for observations
- Linear constraints solvable in polynomial time

Varieties of Constraints

- Unary involve a single variable
- Binary involve variable pairs
- Higher-order involve 3 or more variables (e.g. Sudoku)
- Preferences e.g. red *is better than* green

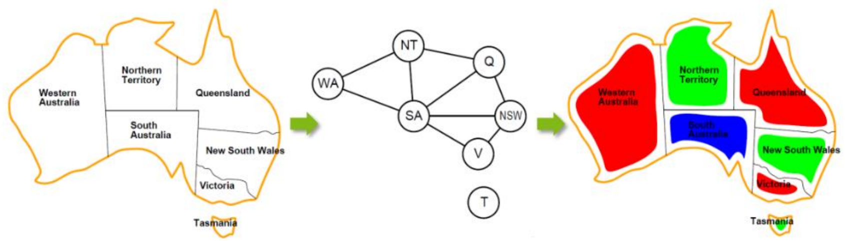
Real World CSPs / Usage of CSPs

- *Assignment* problems who teaches what class?
- *Timetabling* problems which class is offered when / where?
- *Optimization* with spreadsheets debugging
- Other *scheduling* tasks transportation or facotry workflow
- Other *layout* tasks floor planning / hardware configuration

<u>Example</u>: Map-coloring

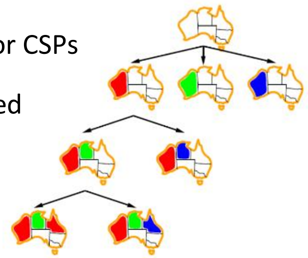
- Variables WA, NT, Q, NSW, V, SA, T
- Domains $D_i = \{red, green, blue\}$
- Constraints Adjacent regions must have different colors e.g. $WA \neq NT$
- Solutions Assignments satisfying all constraints e.g. $\{WA = red, NT = green, \dots\}$

Binary CSPs have a *constraint graph*. General Purpose CSP algorithms use the graph structure to speed up search.



Constraint satisfaction problems (CSP)

Backtracking search



Depth-first search with single-variable assignments for CSPs

General purpose methods can give huge gains in speed

- *Which variable* should be assigned next?
- In what *order* should its *values* be tried?
- Can we *detect* inevitable *failure early*?
- Can we *take advantage* of the *problem structure*?

Can be achieved by implementing the *bold/italic* functions below

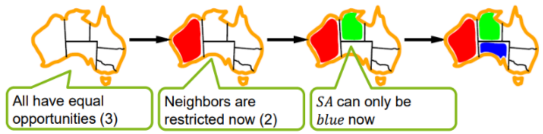
```
function Backtracking-Search(csp) returns solution/failure
    return Backtrack({}, csp)

function Backtrack(assignment, csp) returns solution/failure
    if assignment is complete then return assignment
    var ← Select-Unassigned-Variable(csp)
    for each value in Order-Domain-Values(var, assignment, csp) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← Inference(csp, var, value)          #optional
            if inferences ≠ failure then                      #optional
                add inferences to assignment                  #optional
                result ← Backtrack(assignment, csp)
                if result ≠ failure then return result
        else remove {var = value} from assignment
    return failure
```

Next variable: Ideas for *Select-Unassigned-Variable(csp)*

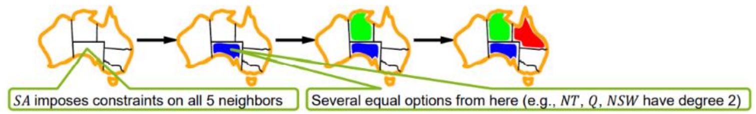
Minimum remaining values (MRV)

- Choose the variable with the *fewest legal values* → *fail fast*



Degree heuristic

- Choose the variable that adds most constraints on remaining variables
  → Works as *tie-breaker* in practice within MRV



Order of values: Ideas for *Order-Domain-Values(var, assignment, csp)*

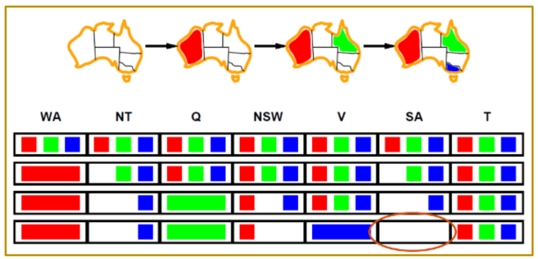
Least constraining value

- Given var, choose the value that rules out the fewest values in the remaining $vars$.

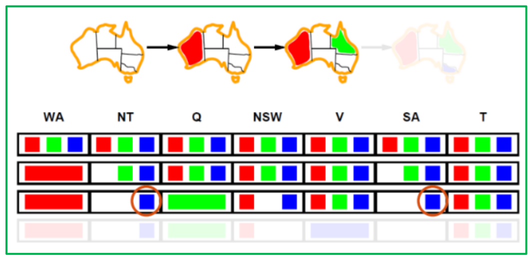
Detect inevitable failure: Ideas for *Inference(csp, var, value)*

Forward checking

- Keep track of remaining legal values for unassigned variables
  → Terminate search when any variable has no legal values



Constraint propagation

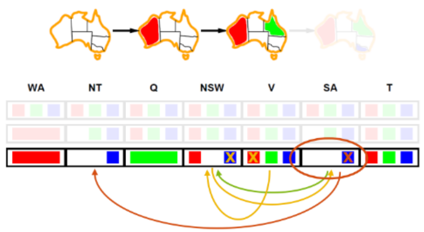


- Forward checking propagates information from assigned variables only to immediate neighbours.

Constraint satisfaction problems (CSP)

Detect inevitable failure: Ideas for *Inference(csp, var, value)*

- Arc consistency – the simplest form of constraint propagation
  - $X \to Y$ is consistent iff for every value x of X there is some allowed y for Y
  - Arc consistence detects failure earlier than forward checking



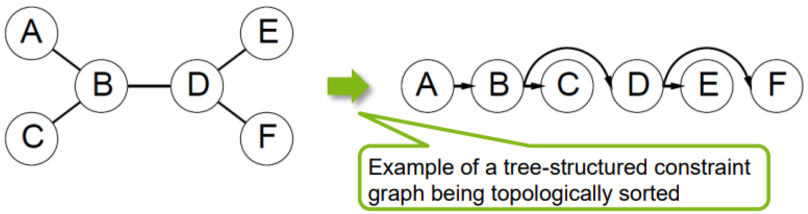
Taking advantage of problem structure

Tree-structure CSPs

- If the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time

Algorithm for tree-structured CSPs

- Do a topological sort
- Create directed arc-consistency by
  - For j from n down to 2, make arc consistent
- For j from 1 to n, assign X_j consistently with



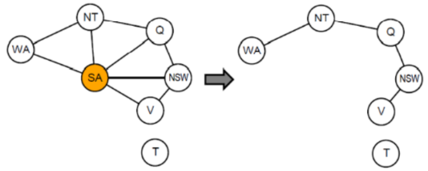
Terms

- Consistent / Legal = An assignment that doesn't violate any constraints
- Complete = An assignment in which every variable is assigned a value
- A solution is consistent and complete
- Partial assignment leaves some variables unassigned
- Partial solution consists of partial assignments

Solving CSPs in Practice

Nearly tree-structured CSPs

- Many real-world CSPs can be converted to tree-structured problems
- ...By choosing a cycle cutset: a set of variables that if removed make the graph a tree



- ...and subsequent cutset conditioning: Instantiate the variables in the cutset then prune choices from remaining variables in the tree.

Other advice

- Exploiting structure in the values by breaking symmetry reduces search space up to d
- Local search is very effective for CSPs
  - e.g. Hill climbing with min-conflicts heuristic
- Constraint learning is one of the most important techniques in modern CSP solvers
- Trade-off: enforcing consistency vs. search time

Knowledge, reasoning and logic

Knowledge bases (KB)

A set of sentences in a formal language

- Declarative approach to building an agent
- Two views of an agent
  - At the *knowledge* level
  - At the *implementation* level

```
function KB-Agent(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time
    Tell(KB, Make-Percept-Sentence(percept, t))
    action ← Ask(KB, Make-Action-Query(t))
    Tell(KB, Make-Action-Sentence(action, t))
    t ← t+1
    return action
```

The agent must be able to

- Represent **states**, **actions**, etc.
- Incorporate new **percepts**
- Update **internal representations** of the world
- **Deduce** hidden properties of the world
- Deduce appropriate actions

Logic is a formal language for representing information

- *Syntax* defines the «structure» of sentences
- *Semantics* defines the «meaning» of sentences

Entailment ($\vDash, \vdash$)

Entailment is a relationship between sentences that is based on semantics

$KB \vDash \alpha$

- Entailment means that one thing follows from another:
  from KB I know α
- KB entails sentence α *iff* α is true in all worlds where KB is true

Inference ($\vDash_i, \vdash_i$)

$KB \vDash_i \alpha$

- Sentence α can be derived from *KB* by procedure i

Desirable properties of i

- Soundness i is sound whenever $KB \vDash_i \alpha$, it is also true that $KB \vDash \alpha$
- Completeness i is complete if whenever $KB \vDash \alpha$, it is also true that $KB \vDash_i \alpha$

Propositional logic (Aussagenlogik)

Reasoning over unrelated facts

- The simplest of all logics to illustrate basic ideas

$\neg$	$\neg A$	NICHT A
$\wedge$	$A \wedge B$	A UND B
$\vee$	$A \vee B$	A ODER B
$\Rightarrow$	$A \Rightarrow B$	WENN A DANN B
$\Leftrightarrow$	$A \Leftrightarrow B$	A GLEICH B

Propositional logic: Pros and Cons

- Declarative: pieces of syntax correspond to facts
- Allows partial/disjunctive/negated information
- Meaning is context-independent

First-order logic (FOL) = Prädikatenlogik 1. Stufe

- Quantifiable variables $\forall, \exists$
- Objects people, houses, numbers,…
- Relations (predicates) red, round, prime,…

Logical equivalence: rules to manipulate sentences of logic

double-negation	$\neg\neg A$	$\Leftrightarrow$	A
contraposition	$A \Rightarrow B$	$\Leftrightarrow$	$\neg B \Rightarrow \neg A$
implication	$A \Rightarrow B$	$\Leftrightarrow$	$\neg A \vee B$
commutativity	$A \wedge B$ $A \vee B$	$\Leftrightarrow$	$B \wedge A$ $B \vee A$
associativity	$(A \wedge B) \wedge C$ $(A \vee B) \vee C$	$\Leftrightarrow$	$A \wedge (B \wedge C)$ $A \vee (B \vee C)$
distributivity	$A \wedge (B \vee C)$ $A \vee (B \wedge C)$	$\Leftrightarrow$	$(A \wedge B) \vee (B \wedge C)$ $(A \vee B) \wedge (B \vee C)$
De Morgan	$\neg(A \wedge B)$ $\neg(A \vee B)$	$\Leftrightarrow$	$\neg A \vee \neg B$ $\neg A \wedge \neg B$

A sentence is ... if it is ...

- Valid allgemeingültig *true* in *all* possible models
- Satisfiable erfüllbar *true* in *some* model
- Unsatisfiable Unerfüllbar *false* in *all* models

Example: Which of the following is correct?

- $False \vDash True$ $True$
- $True \vDash False$ $False$
- $A \wedge B \vDash A \Leftrightarrow B$ $True$

Example: Formuliere die folgenden Aussagen mithilfe der Prädikate

- $B(x)$ $istBloff(x)$
- $W(x)$ $istWuergel(x)$
- $P(x,y)$ $Pfennert(x,y)$
- $N(x)$ $Nausert(x)$

Zu jedem Würgel gibt es einen Bloff, der von diesem Würgel gepfennert wird.

$$\forall x \exists y \big(W(x) \wedge B(y) \wedge P(x,y)\big)$$

Wenn irgendein Bloff nausert, dann nausern alle Bloffs.

$$\exists x \big(B(x) \wedge N(x)\big) \rightarrow \forall y \big(B(y) \wedge N(y)\big)$$

Wenn es für jeden Bloff einen Würgel gibt, der diesen Bloff pfennert, dann nausern alle Würgel.

$$\forall x \exists y \big(B(x) \wedge W(y) \wedge P(y,x)\big) \rightarrow \forall z \big(W(z) \wedge N(z)\big)$$

Datalog

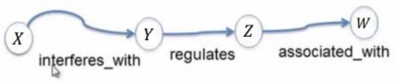
Reasoning in databases

Implementing an ontology (a graph) using trioples in a database. We're interested in pattern matching in graphs such as records in relational databases.

Task: For a given graph and pattern, find all instances of pattern.

Example: Given a graph with edge labels ... Find drugs that interfere with another drug involved in the treatment of a disease

- Drug X interferes with drug Y
- Drug Y regulates the expression of gene Z
- Gene Z is associated with disease W



Assuming a relation $r(subject, predicate, object)$ and pseudo syntax:

$result(X) <=$
$r(X, interferesWith, Y)$ & $r(Y, regulates, Z)$& $r(Z, associatedWith, W)$

Datalog – A relevant subset of FOL

Background

- Full FOL is very expressive, but not decidable in general
- Thus: Fallback to first-order definite clauses (Horn clauses)
- Can represent the type of knowledge typically found in relational databases

Datalog Terminology

- Knowledge base   a set of clauses
- Clause   is either an atomic symbol (fact) or a rule
- Atom   has either the form p or $p(t_1, \dots, t_n)$
  - Predicate
  - Term (variable or constant)

Inference in Datalog

Foundation: *Modus Ponens = implication elimination*

- If $P \Rightarrow Q$ and $P = true$, then $Q = true$

$$\frac{P \Rightarrow Q,\ P}{Q}$$

Forward chaining (data-driven approach):

- Search for true antecedents («if clauses»)
  → infer consequent («then clause») to be true
  → add this information to *KB*
- Intuitively understandable
- Sound and complete for Datalog
- Efficiently implemented for Datalog (CSP)

Backward chaining (goal-driven approach):

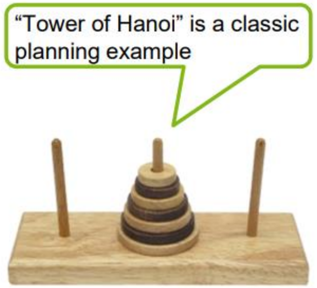
- Produces no unnecessary facts
- Sound and complete for Horn clauses
- Typically implemented using a form of SLD resolution (depth-first)

Planning as Search

Planning is the *art and practice of thinking before acting*. Classical planning is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.

Why is planning so big?

- Solved applications: Large logistics problems, operational planning, robotics, scheduling,…
- Community: Search is its basis; logic and knowledge representation is part of it



Automated Planning

- A Single agent in a → multi-agent / game-playing possible
- Fully observable, → conformant planning possible
- Sequential and discrete → temporal and real-time planning possible
- Deterministic and → probabilistic planning possible
- Static (offline) environment → online possible

Planning **D**omain **D**efinition **L**anguage (**PDDL**)

- Subset of FOL
- Used to define the planning task as a search problem
- Derived from STRIPS planning language
- Allows for factored representation

Restricted language allows for efficient algorithms

- Action precondition: conjunction of positive literals
- Action effects: conjunctions of literals
- Applicability of action a in state s: $iff\ s \vDash Precondition(a)$

<u>Example - Action</u>

$$
\begin{aligned}
&Action(Fly(p, from, to),\\
&\quad Precondition: At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)\\
&\quad Effect: \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Planning Algorithms

Planning as state-space search – approachable with any algorithm from V03 or local search

- Forward (progression): search considers actions that are *applicable*
- Backward (regression): search considers actions that are *relevant*

Heuristics for forward state-space search – enabled by factored representations for states and actions

Possible domain-independent heuristics (adding new links to the graph to ease the problem)

- Relaxing actions
  - Ignore-precondition heuristic   All actions are applicable anytime
  - Ignore-delete-list heuristic   Removing all negative literals from effects
- State abstraction   Reduce the state space by e.g. ignoring some fluents

Hierarchical planning: A modern, more general alternative

- Technical solution sketch
  - Hierarchical task networks (HTN): more factored representations for actions
  - Two kinds of actions: Primitive actions and High level actions (HLA)

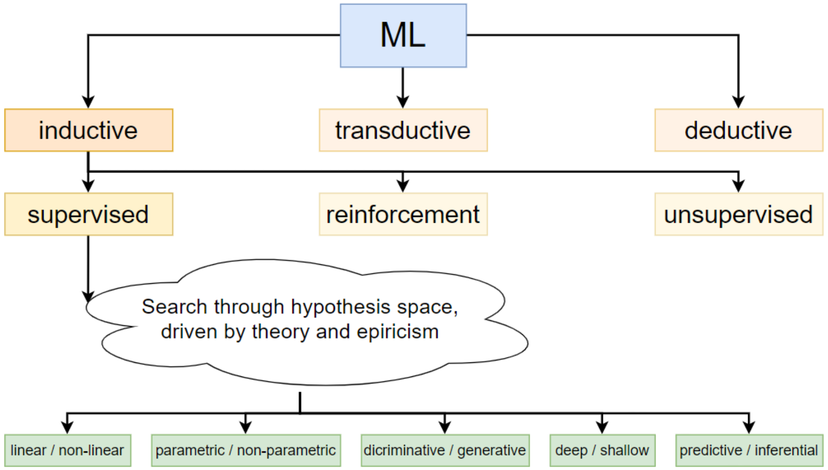
Hierarchical planning algorithms

- Search for primitive solutions: *Hierarchical-Search*
  - Recursively chose a HLA in current plan
  - Replace HLA with one of its refinements, until plan achieves its goal
- Search for abstract solutions
  -
- *Angelic-Search*

Supervised Learning

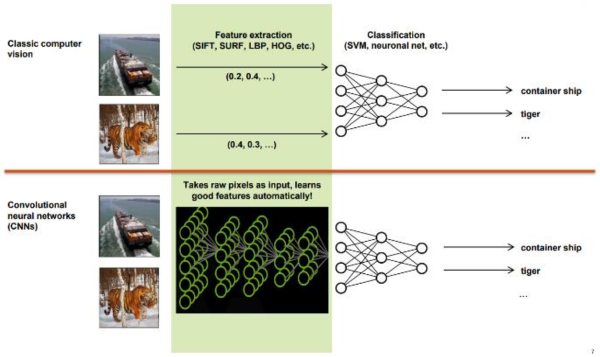
Types of Feedback

- Supervised learning — observes input-output pairs, learns function
- Unsupervised learning — detects clusters, learns patterns in the input
- Reinforcement learning — learns from rewards and punishments



Shallow vs deep learning: *Add depth to learn features automatically*

- Classic computer vision — Uses manually extracted features
- **C**onvolutional **N**eural **N**etworks (CNN) — Uses raw input to learn features



The task of **supervised learning** is this

Given a *training set* of N example input-output (feature-label) pairs
$$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$$
Where each pair was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f.

The function / model h is the *hypothesis*, drawn from a *hypothesis space* $\mathcal{H}$ of all possible functions.

The model h should find the *best-fit function*. *Overfitting* and *underfitting* should be avoided.

- Underfitting — h fails to find a pattern in the data
- Overfitting — h pays too much attention to a particular data set
- Test / train data
- Bias-variance tradeoff
  - complex (low-bias) h, that fits training data better
  - simple (low-variance) h, that may generalize better

A good model h complies with *Ockham's razor* principle: Maximize a combination of *consistency* and *simplicity*.

Doing machine learning

Performance measurement

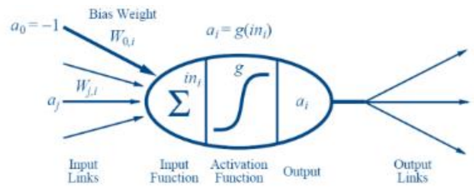
1. Use theorems of computational/statistical learning theory
2. Try h on a new test set (→ use cross-validation)
3. Report performance (*A*ccuracy, *P*recision, *R*ecall)
   - $A = \frac{TP+TN}{TP+TN+FP+FN}$, $P = \frac{TP}{TP+FP}$, $R = \frac{TP}{TP+FN}$

classification → ↓ label	1	0
1	true positive (TP, "hit")	false negative (FN, "miss")
0	false positive (FP, "false alarm")	true negative (TN)

Supervised Learning using Neural Networks

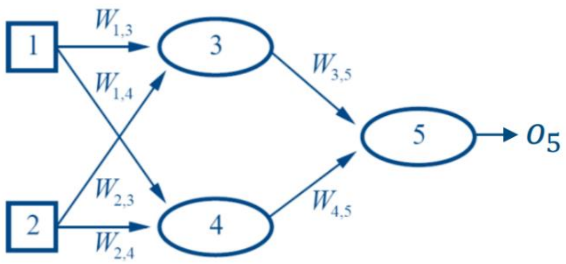
Neurons

- Oversimplification of real neurons
- Output is a thresholded linear function of the inputs: $a_i = g(in_i) = G(\sum_j w_{j,i} \cdot a_j)$
- Changing the *bias weight* $W_{0,i}$, moves the threshold location



Feed-forward neural network (FNN)

A FNN, has connections only in one directions. Each node computes a function of its inputs and passes the result to its successors.



Convoluational Neural Network (CNNS)

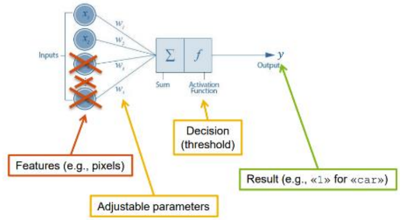
- Goal fewer free parameters → eases learning
- Idea exploit 2D-correlated local structure in (image) input data
- Principle
  - A «filter» moves over every input pixel and calculates a feature that describes the pixels' local context
    → map result to same spatial location
    → filter weights is trainable
  - Have several such «filters» to encode different features
  - After each filtering layer, sub-sample result to reduce spatial resolution and increase «field of vision»

Neural Network: Weight Adjustment

Our example *neural network*

$$f_W(x) = y$$

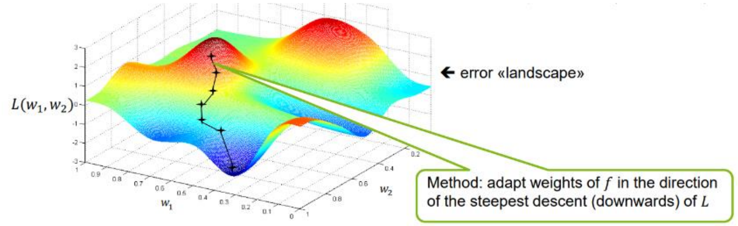
with image x, ground truth y and params W
($W = \{w_1, w_2, \dots\}$ initialized randomly)



Error measure

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} (f_W(x_i) - y_i)^2$$

Average of quadratic difference on all images (loss function L)



Trained by gradient descent (complete network ist differentiable)

- Forward pass: calculation of loss function L for a mini batch of training samples
- Backward pass: calculation of $\frac{\partial L}{\partial W_{l,i}}$ for each weight $W_{l,i}$ on overall loss

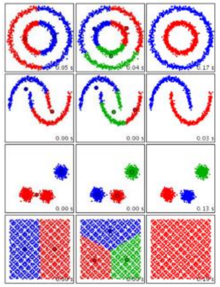
Unsupervised Learning with Autoencoders

Flavors of unsupervised learning (UL)

Usual task: Clustering

- Separate N examples described by feature vectors into K groups.

Challenges



- similarity by distance or density
- Choice of parameters

Other tasks

- Discovery of unobserved variables
- Dimensionality reduction
- Feature / representation learning (e.g. autoencoders)
- Matrix completion (e.g. for recommendation)
- Discovery of dependency structure in features (graph analysis)

Observation: UL is less employed than SL.

Problem: cost function is unclear!

Reason

- UL is often *used to improve SL* in absence of enough labeled data
- Without labels, UL cost *doesn't know which SL task to focus on*

Solution: **Output distribution Matching (ODM)**

Use distribution instead of exact constraint for cost function:

- SL maps data X to labels Y via $Y = F(X), (X, Y) \sim D$
- Impose constraint on F using uncorrelated samples
  $$x \sim D, y \sim D: Distr[F(x)] = Distr[y]$$
- Use it as UL cost function: $KL(Distr[y] || Distr[F(x)])$

➢ Cost works towards matching distribution of inferred labels to the one in known(x, y) pairs
➢ High chance of practically improving SL if ODM cost can be optimized

Summary

Learning from the data itself is also the main learning signal in biological learning.

UL is deemed the greatest innovation area in ML by many experts. UL is more than clustering, in particularly, feature learning via deep models. UL to facilitate some SL task may benefit from output distribution matching.

AE learn the structure of the data by balancing approximate reconstruction with some regularization penalty. They thus learn to capture lower-dimensional manifolds and important aspects of the underlying data-generating distribution.

Unsupervised Learning with Autoencoders

Autoencoders (AE)

An *autoencoder* 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… *Autoencoders* are designed to be **unable to learn to copy perfectly**.

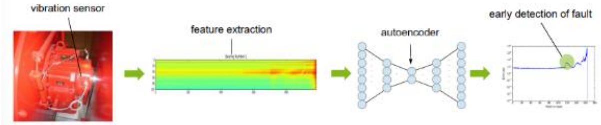
- Desired effect — Learn useful properties of the data
- Application scenarios
  - Traditionally — dimensionality reduction, feature learning
  - Recently — generative modelling (VAE, GAN)

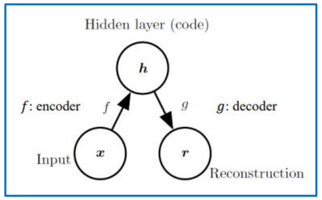
Use case 1 for AE: Learning embeddings

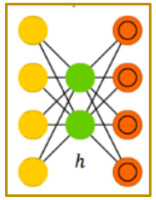
- Embedding ≔ Lower-dimensional representation in an "embedded subspace"
- Applications
  - ✓ Unsupervised pre-training
  - ✓ Feature learning
  - ✓ Dimensionality reduction

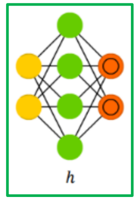
Use case 2 for AE: Novelty detection

- Because the AE learns to *encode* / capture *variations in the training data*, it is by design *bad in encoding* previously *unseen variation*
- Application: Predictive maintenance
  - Vibration signal → feature extraction via spectrogram → autoencoder
  - Monitor reconstruction error as a "novelty signal"









Undercomplete (compressing) autoencoders

- h has lower dimension than x
- Must discard / *compress* some information in h

Overcomplete (regularized) autoencoders

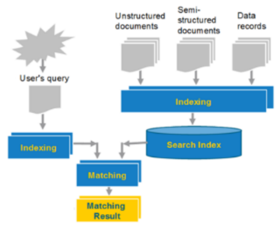
- h has higher dimension than x
- Must be *regularized*

Use case 3 for AE: Information retrieval (IR) via semantic hashing

Efficient IR by dimensionality reduction

- Given a set of documents
- Train an AE to produce a code that is low dimensional and binary
- Create a hash table from binary code to document

- Retrieve all docs that have the same binary code as the query
- Enlarge the similar results: flip bits from query's encoding



Generative Adversarial Learning

Generative Adversarial Nets (GANs)

Flavors of generative models

- Statistical — models that directly model the pdf
- Graphical — models with latent variables
- Autoencoders

Promises (Pros)

- Learning about high-dimensional, complicated probability distribution
- Simulate possible futures for planning or simulated reinforcement learning
- Handle missing data
- Some applications actually require generation

Common drawbacks (Cons)

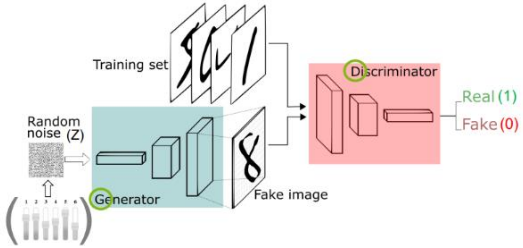
- Statistical models suffer severely from curse of dimensionality
- Approximations needed for intractable probabilistic computations during ML estimation
- Unbacked assumptions and averaging

Adversarial nets: *Bootstrapping implicit generative representations*

Train 2 models simultaneously

- G: Generator — learns to generate data points x
- D: Discriminator — learns $p(x\ not\ being\ generated)$

D and G learn, while competing!

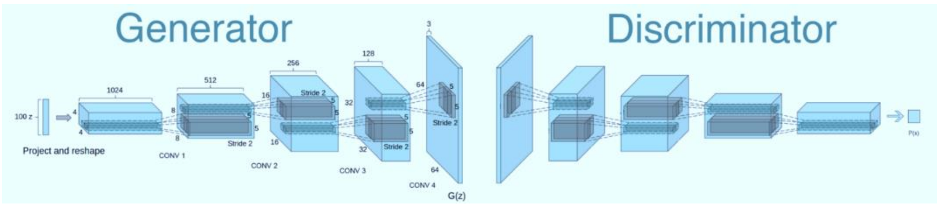


The latent space Z serves as a source of variation to generate different data points. Only D has access to real data.

GAN model formulation (improved):

Implement both G and D as deep convnets (DCGAN)

- No pooling, only fractionally-strided convolutions (G) and strided convolutions (D)
- No fully connected hidden layers for deeper architectures



Features of (DC)**GANs**

Learn semantically meaningful latent space

Training is not guaranteed to converge

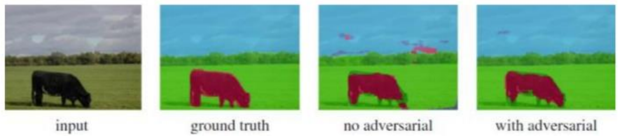
- Gradient descent isn't meant to find the corresponding Nash Equilibria
- How to sync D's and G's training is experimental
- Research on adversarial and neural networks is still ongoing (2022)

GAN use cases

Research has gained a lot of momentum very quickly. GANs have shown to produce realistic output on a wide range of image, audio, and text generation tasks.

- Generate images from text

- Segment images into semantically meaningful parts



- Complete missing parts in images

Reconstruction formulation

Given

- Uncomplete/corrupted image x_{corr}
- Binary mask M (1 = given, 0 = corr)
- Trained networks G and D

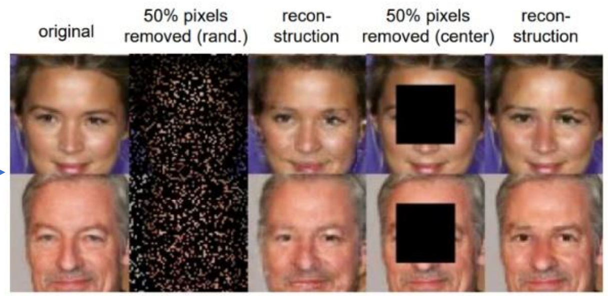
Problem: Find $\hat{z}$ such that $x_{reconstructed} = M \cdot x_{corr} + (1 - M) \cdot G(\hat{z})$

Solution: Define contextual and perceptual loss as follows

- $L_{contextual}(z) = \|M \cdot G(z) - M \cdot x_{corr}\|_1$
- $L_{perceptual}(z) = \log\left(1 - D\big(G(z)\big)\right)$
- $L(z) = L_{contextual(z)} + \lambda \cdot L_{perceptual}(z)$

Image inpainting as a sampling problem approached by ML

Use Case: Complete missing parts in images



Training:

Regard images as samples of some underlying probability distribution P_G.

1. Learn to represent this distribution using a GAN setup (G and D)

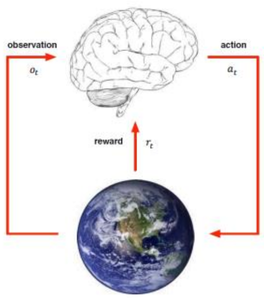
Testing / Application:

Draw a suitable sample from P_G by

1. Fixing parameters Θ_G and Θ_D of G and D, respectively
2. Finding input $\hat{z}$ to G such that $G(\hat{z})$ fits two constraints:
   - Contextual: Output must match the known parts of the image that needs inpainting
   - Perceptual: Output must look generally “real” according to D’s judgement
3. Using gradient-based optimization on $\hat{z}$

Reinforcement Learning

Agent learns by interacting with a stochastic environment!



Faces of reinforcement learning

- Optimal control
- Dynamic Programming (Operations Research)
- Reward systems (Neuroscience)
- Classical/Operant Conditioning (Psychology)

Characteristics

- No supervisor, no goals – only rewards signals
- Feedback is delayed
- Objective: maximize cumulative reward
- Trade-off between exploration and exploitation
- Sequential decisions: actions effect observations

Application Areas

Automated vehicle control, Chat bots, Game playing, DB query optimization, Medical treatment planning, Data Center Cooling,…

The game of Go

- Perfect information, deterministic, two-player, turn-based, zero-sum
- Played on a 19x19 board
- Two possible results: win or lose
- Search space ($\sim 10^{170}$ states, chess: 10^{50})

Alpha Zero Go

Goal

- In state s_t, choose next move a_t

Ingredients

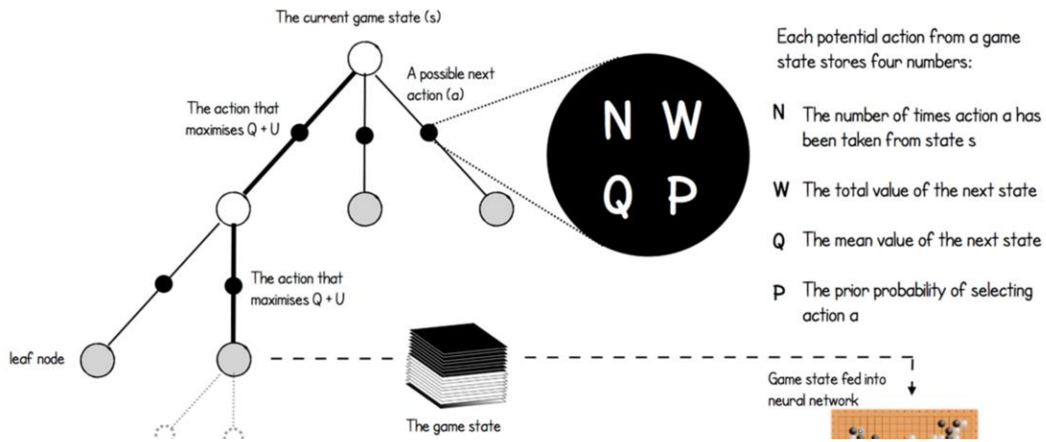
- Neural Network $\vec{p}, v = f_\theta(s_t)$ that outputs two quantities
  - $\vec{p}$ Policy vector distribution over all actions
  - v Value estimated probability of winning
- Monte Carlo Tree Search (MCTS) to build ad hoc search tree
  - MC: tree not fully grown → explore only likely branches

Perform an MCTS search: *provide the basis for a move*

Create (empty or partly re-used) tree with root s_t

Perform 1'600 simulations

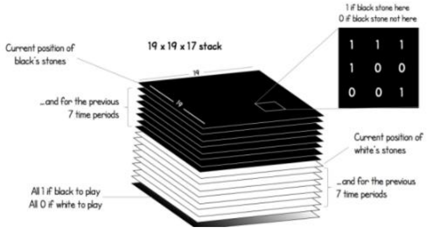
1. Start at $s = s_t$
2. Traverse tree
   - While s is not a leaf node: chose a that maximizes $Q + U$
3. Expand tree: query neural net for $\vec{p}, v = f_\theta(s)$
   $N = 0, W = 0, Q = 0, p = \vec{p}_a$
4. Backup: update statistics of each visited node:
   $N = N + 1, W = W + v, Q = W/N$

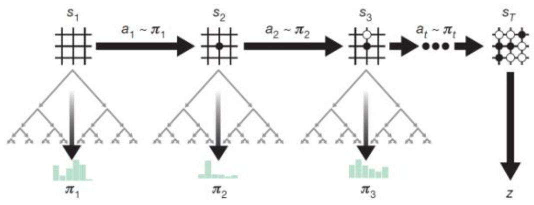


Training the policy / value network by policy iteration

Step 1: Create experience by **selfplay** evaluate the current policy (create training set)

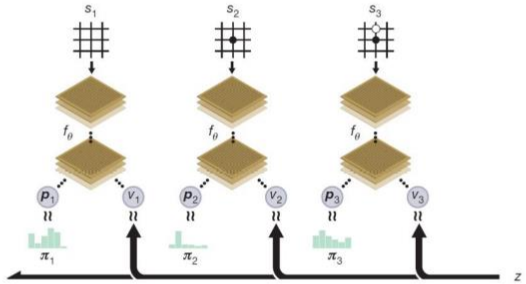
1. Initialize f_θ randomly
2. Play 25'000 games against yourself
   - Use MCTS and current best f_θ for both player's moves
   - For each move, store
     - Game state
     - search probabilities
     - winner ($z = \pm 1$)





Step 2: **(Re-)train** neural **network** improve the current policy (optimise weights)

1. Experience replay: sample mini batch of 2'048 positions from last 500'000 self-play games
2. Retrain f_θ on this batch using supervised learning
   - Input: game states
   - Output move-probabilities p
   - Labels: search-probabilities π, actual winner z
   - Loss: cross-entropy between $p, \pi + MSE(v, z) + L_2 - reg(\theta)$



Step 3: **Evaluate** current **network** test if the new network is stronger

1. Play 400 games between current best vs. latest f_θ
   - Choose each move by MCTS and respective network
   - Play deterministically (no additional exploration)
2. Replace best network with latest f_θ if the latest wins $\geq 55\%$

After 1,600 simulations, the move can either be chosen:

Deterministically (for competitive play)
Choose the action from the current state with greatest N

Stochastically (for exploratory play)
Choose the action from the current state from the distributior

$$\pi \sim N^{1/\tau}$$

where τ is a temperature parameter, controlling exploration

Societal Impact

Responsible AI: Developing for algorithmic fairness (FAT / ML)

Purpose

- Help to build *algorithmic systems in publicly accountable ways*
- Accountability: the *obligation to report, explain, or justify* algorithmic decision-making / *mitigate* any *negative* social *impacts* or potential harms

Premise

- A *human ultimately responsible* for decisions made/informed by an algorithm

Principles

• Responsibility	Make somebody Available who will take care of adverse individual / societal effects
• Explainability	Explain an algorithmic decision in non-technical terms to end users
• Accuracy	Report all sources of uncertainty / error in algorithms & data
• Audibility	Enable 3rd parties to probe & understand system behaviour
• Fairness	Ensure algorithmic decisions are not discriminatory to people groups

Unintended Threats trough AI systems

Algorithmic bias occurs when a computer system behaves in ways that reflects the implicit values of humans involved in the data collection, selection or use.

Different error types, e.g. in a policing application

- Mostly *false positives* for blacks
- Mostly *false negatives* for whites

Semantics by pattern recognition methods can be hard.

According to scholars Walter Krämer, Götz Trenkler, Gerhard Ritter, and Gerhard Prause, the story of the posting on the door, even though it has settled as one of the pillars of history, has little foundation in truth. The story is based on comments made by Philipp Melanchthon, though it is thought that he was not in Wittenberg at the time.

What story of little truth is a pillar of history?
Ground Truth Answers: posting on the door | story of the posting on the door | posting on the door
Prediction: the posting on the door

On whose comments is the posting on the door based?
Ground Truth Answers: Philipp Melanchthon | Philipp Melanchthon | Philipp Melanchthon
Prediction: Philipp Melanchthon

Where was Melanchthon at the time?
Ground Truth Answers: not in Wittenberg | not in Wittenberg | not in Wittenberg
Prediction: Wittenberg

What do scholars agree on about the posting on the door story?
Ground Truth Answers: little foundation in truth | has little foundation in truth | settled as one of the pillars of history
Prediction: little foundation in truth

Indirect threat: *mass unemployment*

- Fear — Less qualified jobs vanish due to robots
- Likely
  - Repetitive tasks vanish due to AI
  - Other jobs are created

Societal Impact

Guardian against malicious use

Includes all practices that are *intended* to compromise the security of individuals, groups or a society.

What enables potential threats by AI systems?

- Dual-use area technology
- Efficiency and scalability
- Potential to exceed human capabilities
- Potential to increase anonymity
- Rapid diffusion
- Novel unresolved vulnerabilities

Potential impact areas

- Digital security
  - By using AI systems to automate cyberattacks or social engineering
  - By attacking AI systems
- Physical security
  - By individual drones or autonomous weapons
  - By coordinating swarms that would otherwise not be controllable
  - By making normal autonomous agents malfunction
- Political security
  - By Surveillance and mass collection of data
  - By persuasion through targeted propaganda
  - By deception through synthetic news, videos

Potential interventions

- Learning from and with the cybersecurity community
- Exploring different openness models
- Promoting a culture of responsibility
- Developing technological and policy solutions

Possible futures

The singularity is near (Ray Kurzweil):

- Superintelligence will enhance human life

Autonomous robots will … (Jürgen Schmidhauber)

- Be curious about human life
- Be enabled by artificial curiosity and LSTM neural nets
- Colonize space on the look for resources to reproduce

Humans can become godlike (Yuval Noah Harari):

- Humans will upgrade themselves in 3 ways: biological engineering, cyborg engineering and robotics
- A new class of people will emerge by 2050: the useless class
- The most important skill will be learning to learn

The vision of Gene Roddenberry

- The acquisition of wealth is no longer a driving force in our lives. We work to better ourselves and the rest of humanity.