Course Overview

LECTURE 5 - part I

Knowledge Representation & Logics
Logical Arguments • Propositional Logic
Validity, Equivalence, Satisfiability, Entailment
Inference by Natural Deduction

UNSW
SYDNEY

Planned Topics

AI, Agents & AI Programming

- What is AI?
- Agents, Agent Types
- Agents Tasks
- AI Programming

Machine Learning

- Supervised Machine Learning
- Perceptrons & Neural Networks
- Learning to Act - Reinforcement Learning

Solving Problems by Search

- Path Search – Basic search Algorithms
- Informed - Heuristic Path Search
- Constraint Satisfaction Problems

Knowledge and Reasoning

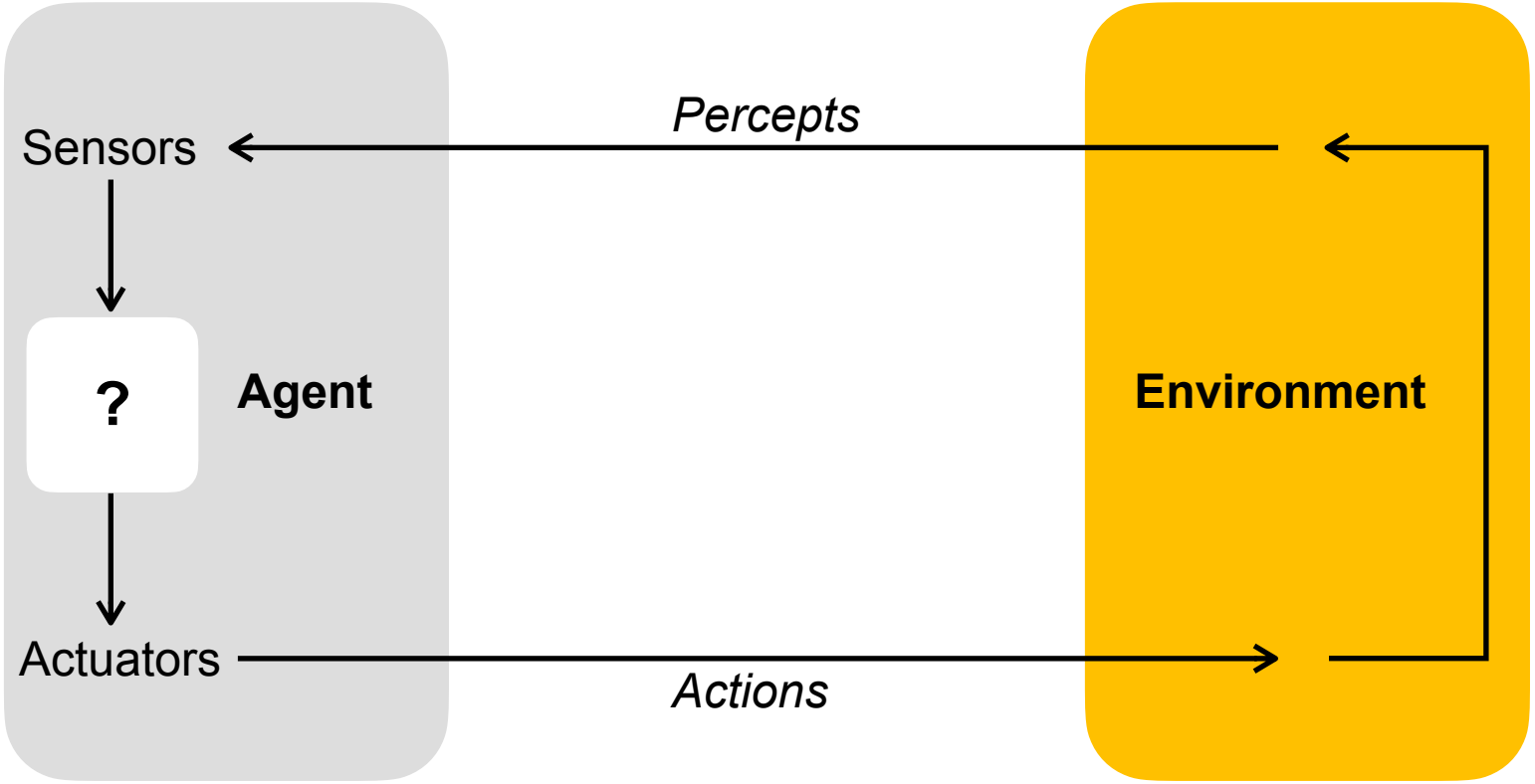
- Planning
- Propositions and Inference
- Reasoning with Uncertainty
- Knowledge Representation

Assessment

- Participation: 10%
- Assignments: 40%
  - assignment 1: 20%
  - assignment 2: 20%
- Exam: 50% + extra points

In order to pass the course, you must score:
- at least 20/40 for the assignments
- at least 20/50 for the exam
- a combined mark of at least 50/100

Agent Model



Classifying environment types

Simulated vs Situated/Embodied

Static vs Dynamic

Discrete vs Continuous

Fully Observable vs Partially Observable

Deterministic vs Stochastic

Episodic vs Sequential

Known vs Unknown

Single-Agent vs Multi-Agent

Environment Types

Simulated: a separate program is used to simulate an environment, feed percepts to agents, evaluate performance, etc.

Static: environment doesn't change while the agent is deliberating

Discrete: finite (or countable) number of possible percepts/actions

Fully Observable: percept contains all relevant information about the world

Deterministic: current state of world uniquely determines the next

Episodic: every action by the agent is evaluated independently

Known: the rules of the game, or physics/dynamics of the environment are known to the agent

Single-Agent: only one agent acting in the environment

Specifying Agents

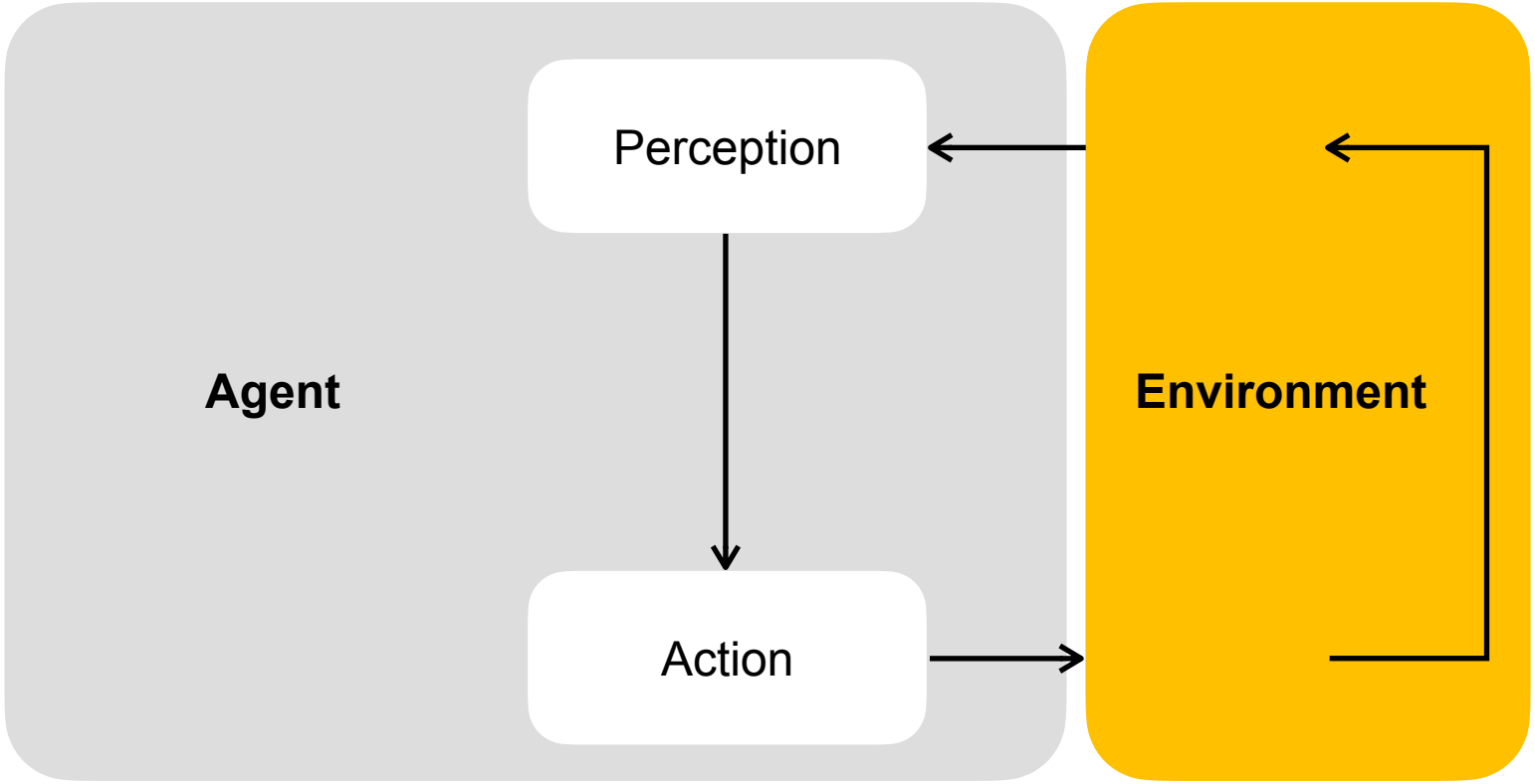
- **percepts**: inputs to the agent via sensors
- **actions**: outputs available to the agent via effectors
- **goals**: objectives or performance measure of the agent
- **environment**: world in which the agent operates

Most generally, a function from percept sequences to actions

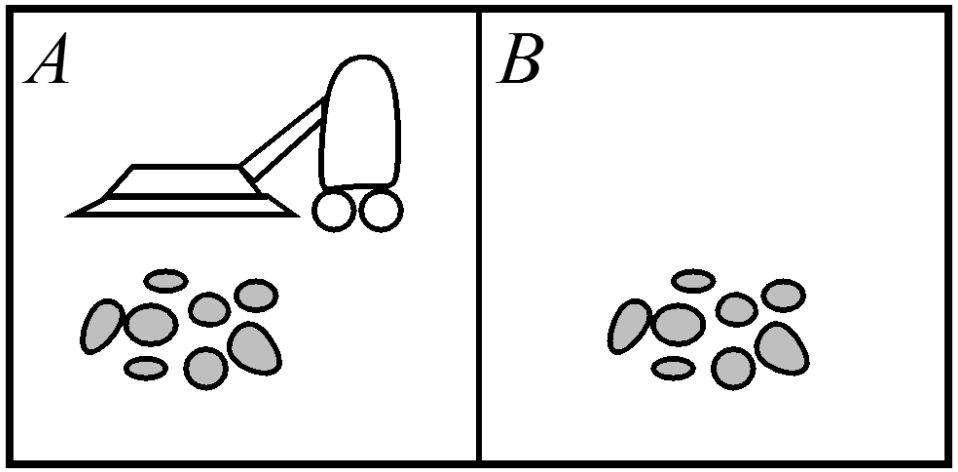
Ideally **rational agent** does whatever action is expected to maximise some performance measure – the agent may not know the performance measure (Russell and Norvig 2010)

Resource bounded agent must make "good enough" decisions based on its perceptual, computational and memory limitations (design tradeoffs)

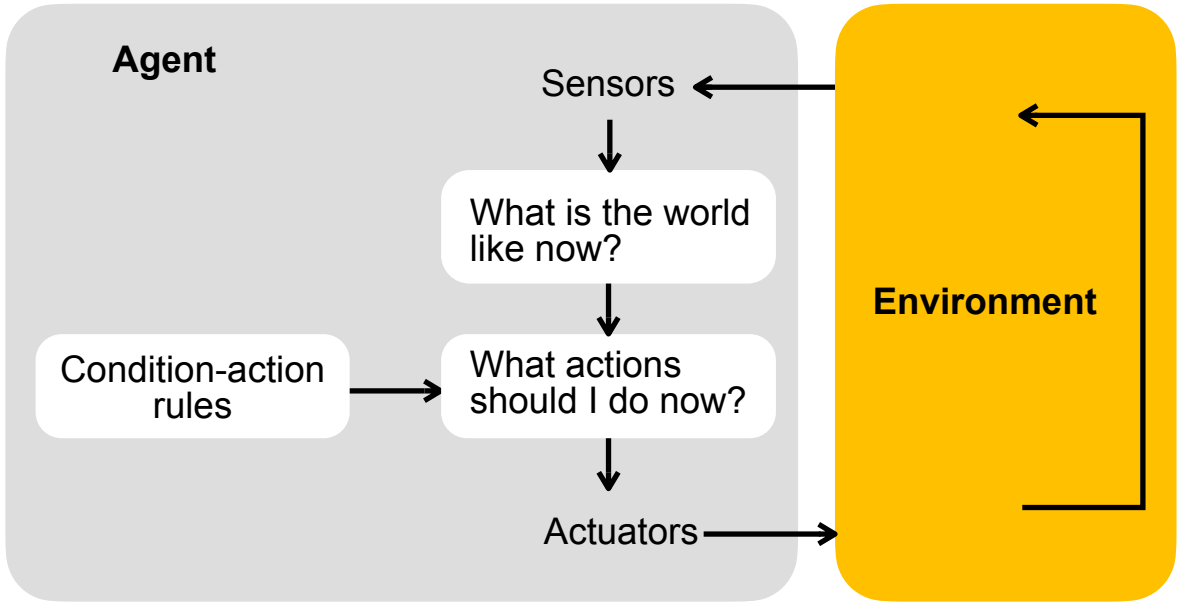
Reactive agent



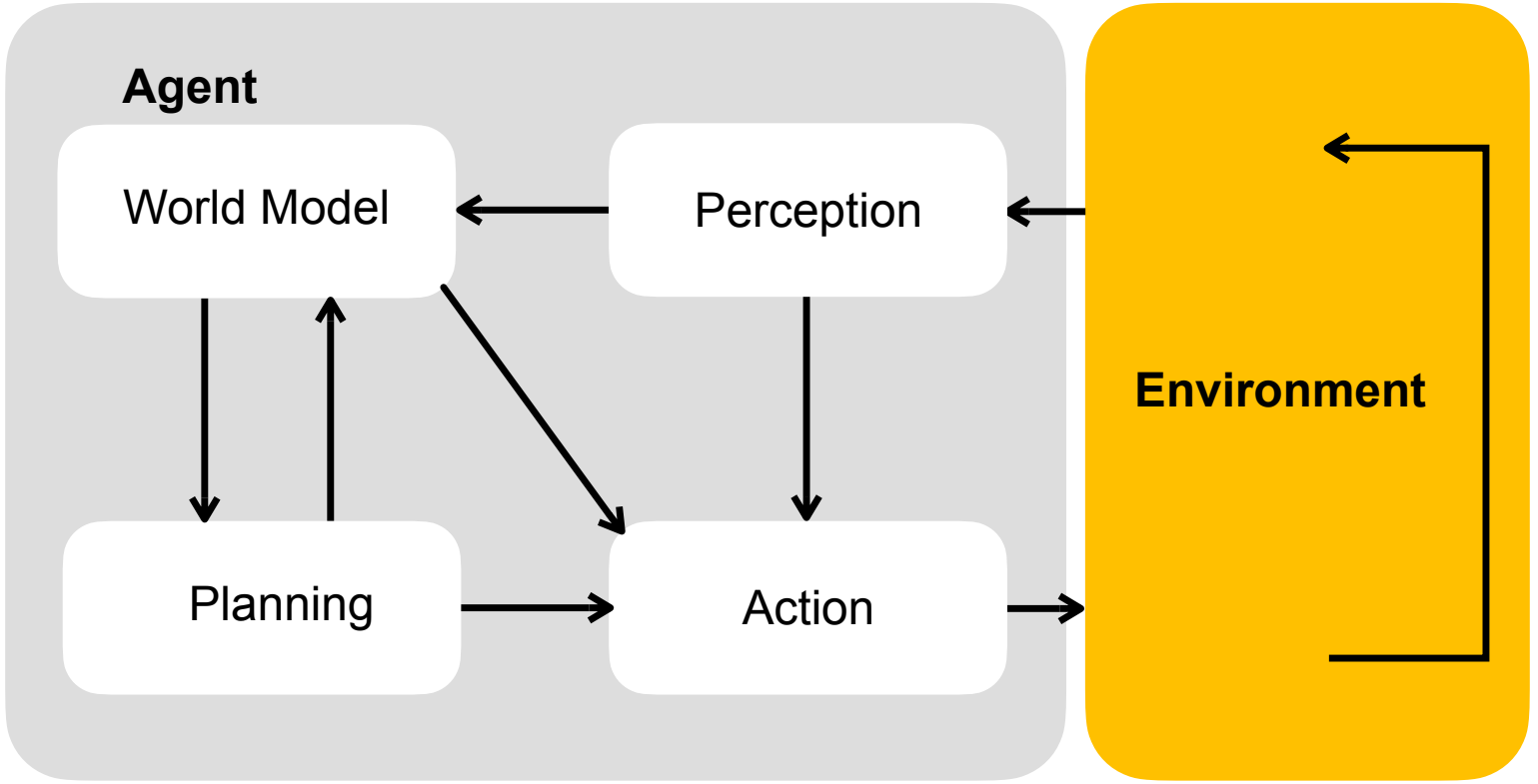
Vacuum-cleaner world



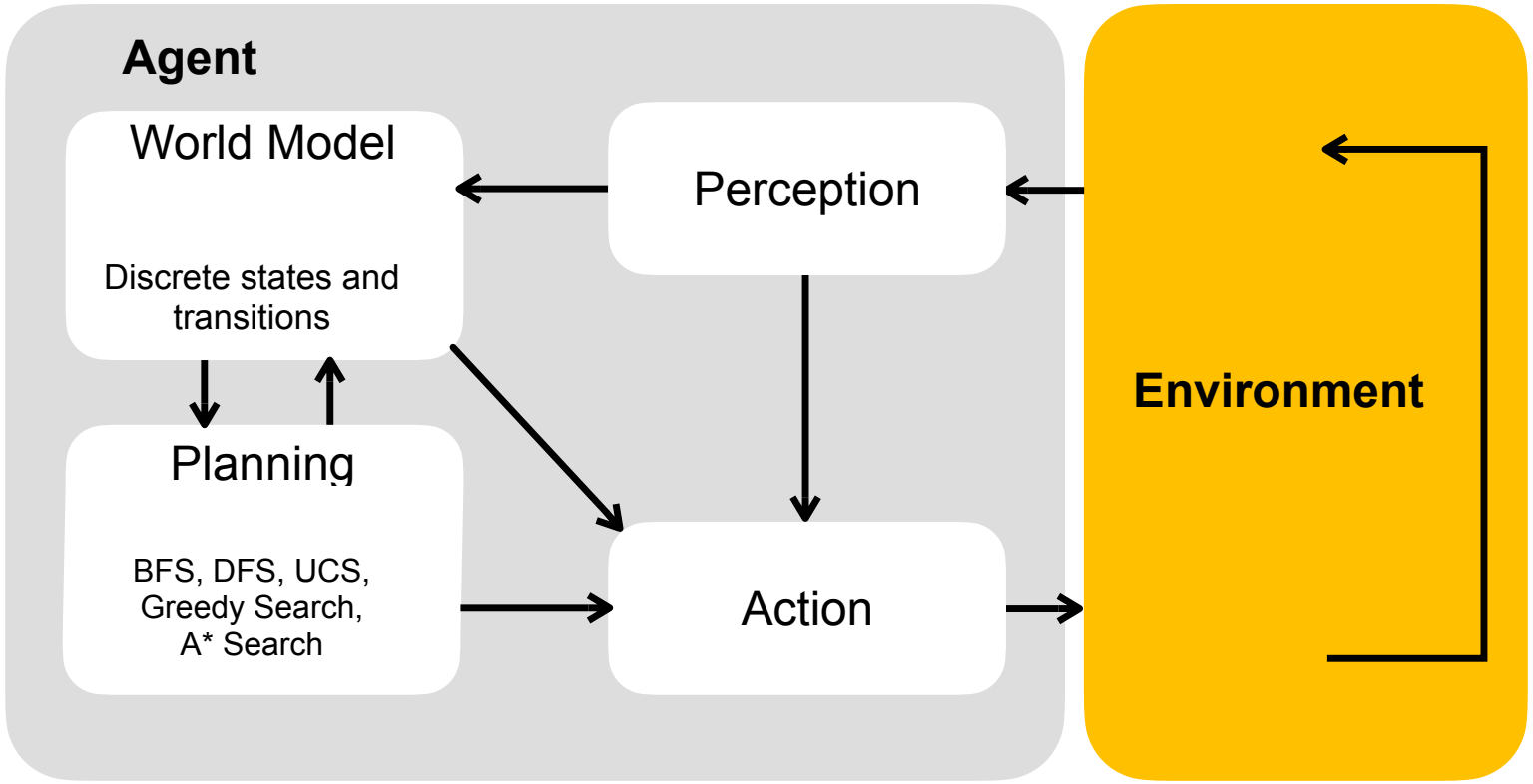
Percepts: location and contents
Actions: Left, Right, Suck, NoOp



Planning Agent



Path Search Agent



State space problem

A state-space problem consists of:

- a set of states
- a subset of states called the **start states**
- a set of actions
- an **action function**: given a state and an action, returns a new state
- a (set of) goal state, specified as Boolean function, goal(s)
- a criterion that specifies the quality of an acceptable solution.

Single-state Space Specification

A task is specified by states and actions:

- state space e.g. other rooms
- initial state e.g., “outside of room 103”
- actions or operators (or successor function S(x)) = set of action–state pairs
- goal test, check if a state is goal state
  - In this case, there is only one goal specified (“inside of room 123”)
- path cost e.g. sum of distances, number of actions etc.
- a solution is a sequence of actions leading from the initial state to a goal state

Assignment 2 - State Space

```
class GameFifteenProblem(Search_problem):

    def __init__(self, start, goal):
        return

    def start_node(self):
        """Returns the start node"""
        return

    def is_goal(self, node):
        """Returns True if the node is the goal, otherwise False"""
        return

    def neighbors(self, node):
        """Returns a list of the arcs for the neighbors of node, for example:
        return [Arc(node, to_neighbor_node1, cost1), Arc(node, to_neighbor_node2, cost2)]"""
        return

    def heuristic(self, node):
        """Returns the heuristic value of the node
        based on the Manhattan distance"""
        return
```

Assignment 2 - Heuristics

```
class GameFifteenProblemEuclidean(GameFifteenProblem):
    def __init__(self, start, goal):
        (super().__init__(start, goal))

    def heuristic(self, node):
        """Returns the heuristic value of the node
        based on the Euclidean distance"""
        return
```

```
class GameFifteenProblemInversions(GameFifteenProblem):
    def __init__(self, start, goal):
        (super().__init__(start, goal))

    def heuristic(self, node):
        """Returns the heuristic value of the node
        based on the sum of the inversion number of a permutation"""
        return
```

Assignment 2 - State Space

Heuristic	h-value: start14	h-value: start17	h-value: start23
Manhattan	#	#	#
Euclidean	#	#	#
Inversions	#	#	#

Heuristic	expanded: start14	expanded: start17	expanded: start23
Manhattan	#	#	#
Euclidean	#	#	#
Inversions	#	#	#

Heuristic	path cost: start14	path cost: start17	path cost: start23
Manhattan	#	#	#
Euclidean	#	#	#
Inversions	#	#	#

Path Search Algorithms - Search Strategies

General Search algorithm:

- add initial state to queue
- repeat:
  - take node from front of queue
  - test if it is a goal state; if so, terminate
  - "expand" it, i.e. generate successor nodes and add them to the queue

Search strategies are distinguished by the order in which new nodes are added to the queue of nodes awaiting expansion.

Search Strategies

- **BFS** and **DFS** treat all new nodes the same way:
  - BFS add all new nodes to the back of the queue (**FIFO**)
  - DFS add all new nodes to the front of the queue (**FILO**)

- **Best First Search** uses an evaluation function *f* () to order the nodes in the queue;
  - Similar to **UCS** $f(n) = g(n)$ of path from root to node n

- Informed or Heuristic search strategies incorporate into *f* () an estimate of distance to goal
  - **Greedy Search** $f(n)$ = estimate $h(n)$ of cost from node n to goal
  - **A$*$ Search** $f(n) = g(n) + h(n)$

Search Strategies

- A strategy is defined by picking **the order of node expansion**
- Strategies are evaluated along the following dimensions:
  - completeness – does it always find a solution if one exists?
  - time complexity – number of nodes generated/expanded
  - space complexity – maximum number of nodes in memory
  - optimality – does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - b – maximum branching factor of the search tree
  - d – depth of the least-cost solution
  - m – maximum depth of the state space (may be ∞)

Complexity Results for Search Algorithms

Algorithm	Completeness	Admissibility	Space	Time
Breadth-First Search	guaranteed, if b is finite	guaranteed, if arcs have the same cost	$O(b^d)$	$O(b^d)$
Depth-First Search	not guaranteed	not guaranteed	$O(bm)$	$O(b^m)$
Uniform Cost Search	guaranteed, if b is finite and cost ≥ ε	guaranteed	$O(b^{[1 + C / \varepsilon]})$	$O(b^{[1 + C / \varepsilon]})$
Depth Limited Search	guaranteed, if b is finite	not guaranteed	$O(bk)$	$O(bk)$
Iterative Deepening Search	guaranteed, if b is finite	guaranteed, if arcs have the same cost	$O(bd)$	$O(b^d)$
Bidirectional Search	depends on search algorithms used	depends on search algorithms used	$O(b^{d/2})$	$O(b^{d/2})$
Greedy Best-First Search	guaranteed, if b is finite and with repeated-state checking	not guaranteed	$O(b^m)$	$O(b^m)$
A* Search	guaranteed	guaranteed if heuristic is admissible/ consistent	$O(b^d)$	$O(b^d)$
Iterative Deepening A* Search	guaranteed	guaranteed if heuristic is admissible/ consistent	$O(d)$	$O(b^d)$

Assignment 2 - complexity vs reality

Part 3: Deceptive Starting States [3 marks]

Task 3.1

Run IDA* on the starting states below and report the number of the expanded nodes.

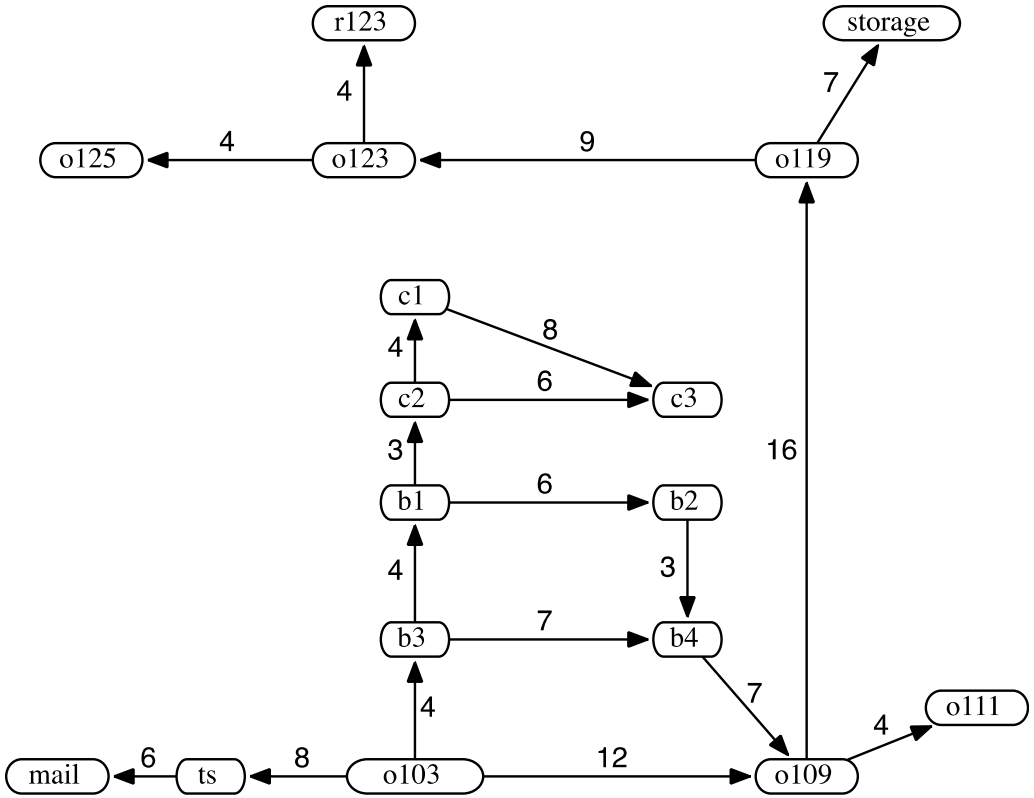
```
start37_1 = [[7, 11, 12, 4],
             [2, 3, 14, 1],
             [6, 5, 13, 0],
             [9, 10, 15, 8]]

start37_2 = [[7, 11, 12, 4],
             [2, 3, 8, 14],
             [6, 0, 1, 15],
             [9, 5, 10, 13]]
# your code
```

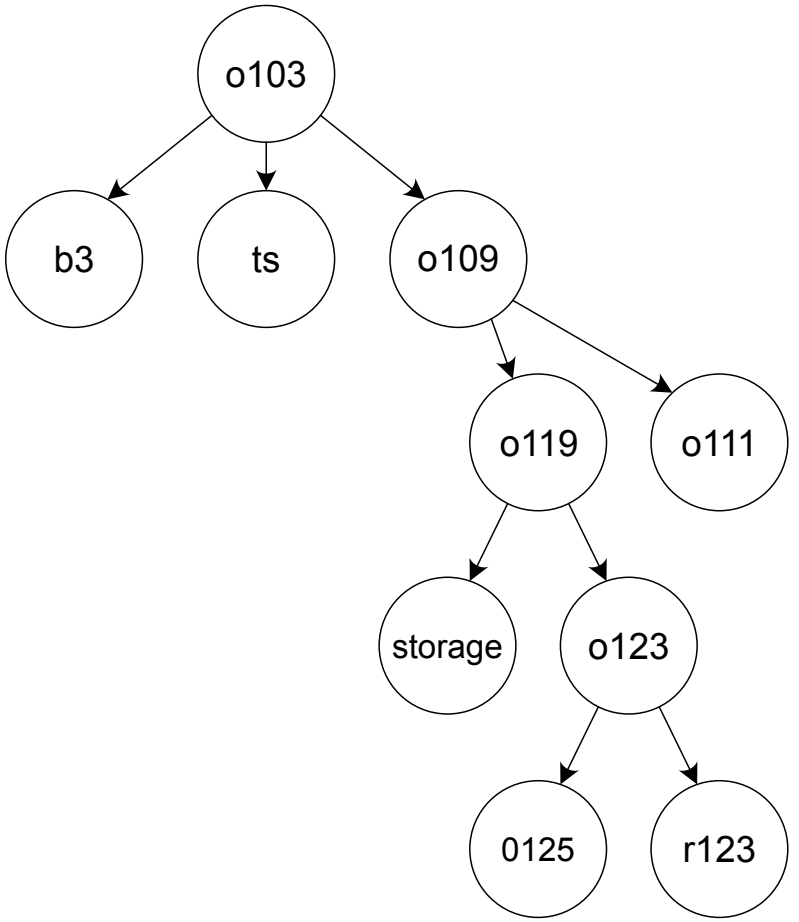
Task 3.2

Explain why there is a difference between the number of the expanded nodes for these starting states if their costs of the optimal paths are the same.

```
# your answer
```

DFS


r123
storage
4
7
o125
4
o123
9
o119
c1
8
4
c2
6
c3
3
16
b1
6
b2
4
3
b3
7
b4
7
4
o111
4
mail
6
ts
8
o103
12
o109


o103
b3
ts
o109
o119
o111
storage
o123
0125
r123

Constraint Satisfaction Problems (CSPs)

- Constraint Satisfaction Problems are defined by a set of variables X_i, each with a domain D_i of possible values, and a set of constraints C.
- The aim is to find an assignment of the variables X_i from the domains D_i in such a way that none of the constraints C are violated.

Example: Map-Colouring

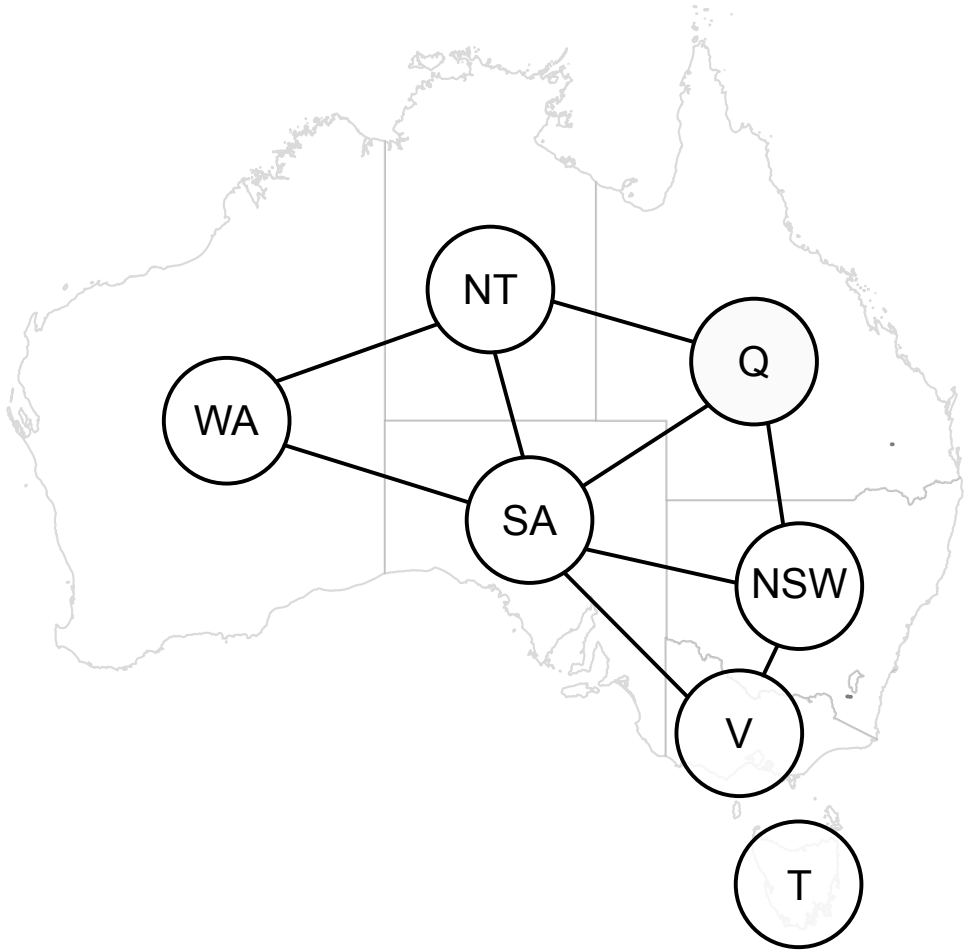
Variables: WA, NT, Q, NSW, V, SA, T

Domains: Di = {red, green, blue}

Constraints: adjacent regions must have different colours **e.g. WA≠ NT, etc.**

- Constraint graph: nodes are variables, arcs are constraints

- Binary CSP: each constraint relates two variables



Assignment 2 - CSP

Part 4: Constraint Satisfaction Problem [5 marks]

This part of the assignment is based on another puzzle called Sudoku. The game consists of 9x9 grid with a few cells filled in with digits. The objective of the game is to fill the remaining cells so that each column, each row, and each of the nine 3×3 sub-grids contain all of the digits from 1 to 9.

The start state of the puzzle is defined as a multidimensional array of numbers, where 0 represents empty cells, for example:

```
grid = [[9, 5, 0, 8, 2, 7, 3, 0, 0],
        [0, 8, 0, 1, 4, 0, 0, 5, 0],
        [0, 1, 0, 5, 9, 0, 0, 0, 0],
        [8, 3, 0, 0, 0, 0, 0, 7, 5],
        [1, 6, 9, 7, 5, 2, 4, 3, 0],
        [0, 7, 0, 0, 8, 0, 0, 6, 0],
        [0, 9, 1, 0, 6, 0, 8, 4, 0],
        [7, 0, 8, 0, 3, 1, 0, 0, 6],
        [6, 2, 0, 4, 7, 8, 0, 9, 0]]
```

Assignment 2 - CSP

```
grid = [[9, 5, 0, 8, 2, 7, 3, 0, 0],
        [0, 8, 0, 1, 4, 0, 0, 5, 0],
        [0, 1, 0, 5, 9, 0, 0, 0, 0],
        [8, 3, 0, 0, 0, 0, 0, 7, 5],
        [1, 6, 9, 7, 5, 2, 4, 3, 0],
        [0, 7, 0, 0, 8, 0, 0, 6, 0],
        [0, 9, 1, 0, 6, 0, 8, 4, 0],
        [7, 0, 8, 0, 3, 1, 0, 0, 6],
        [6, 2, 0, 4, 7, 8, 0, 9, 0]]
```

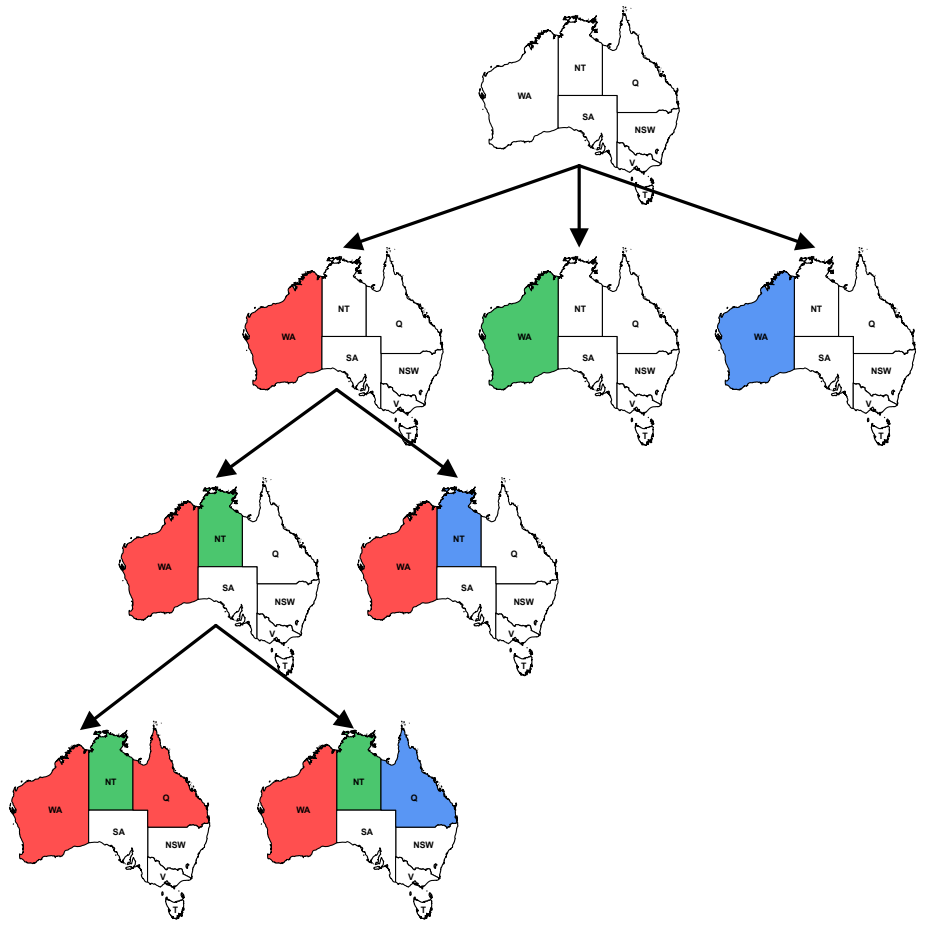
Task 4.1

Write the code that defines constraint(s) and a function grid_to_csp that returns a CSP for a Sudoku puzzle:

```
# define constraint function(s)

# function that returns a csp
def grid_to_csp(grid):
    domains = {}
    constraints = []
    return CSP(domains, constraints)

# csp
csp = grid_to_csp(grid)
```

Constraint Satisfaction Problems

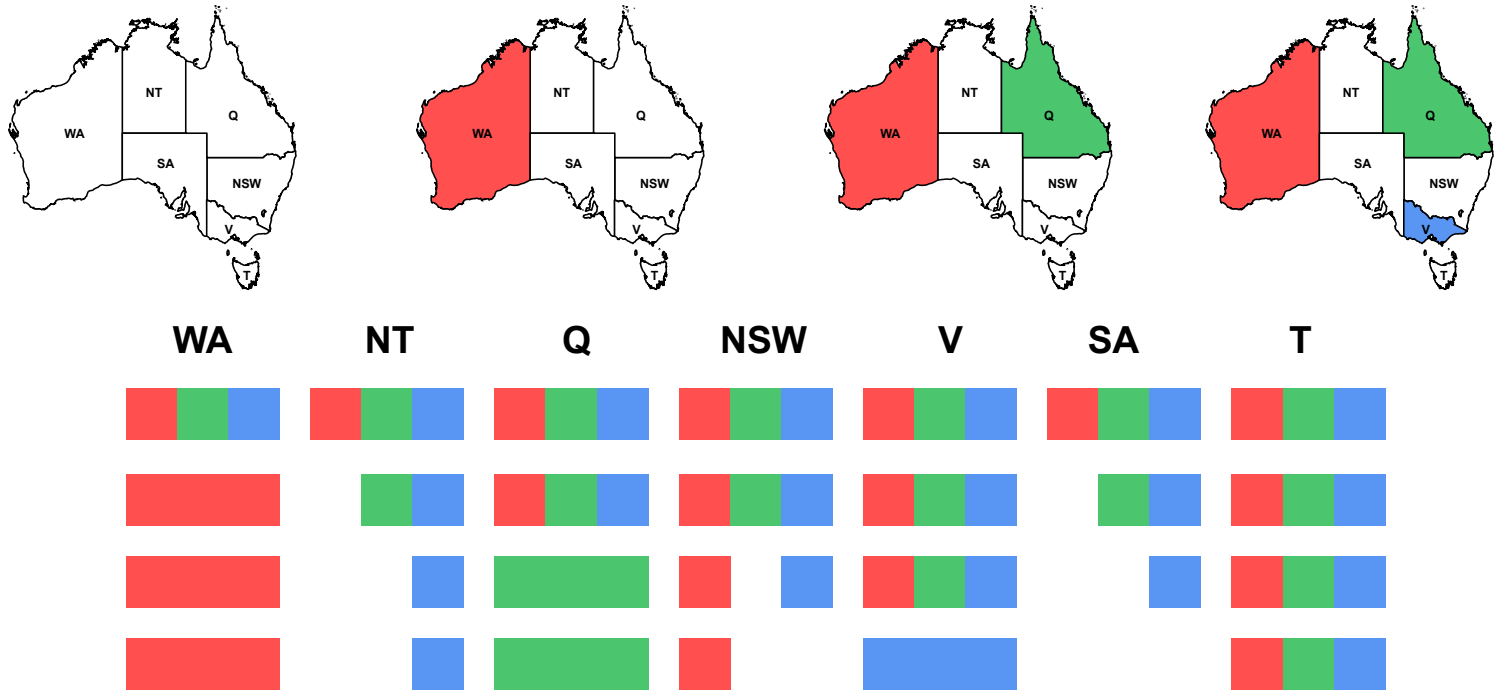
- Backtracking search
- Enhancements to backtracking search
- Local search
  - hill climbing
  - simulated annealing



Forward checking

Idea:

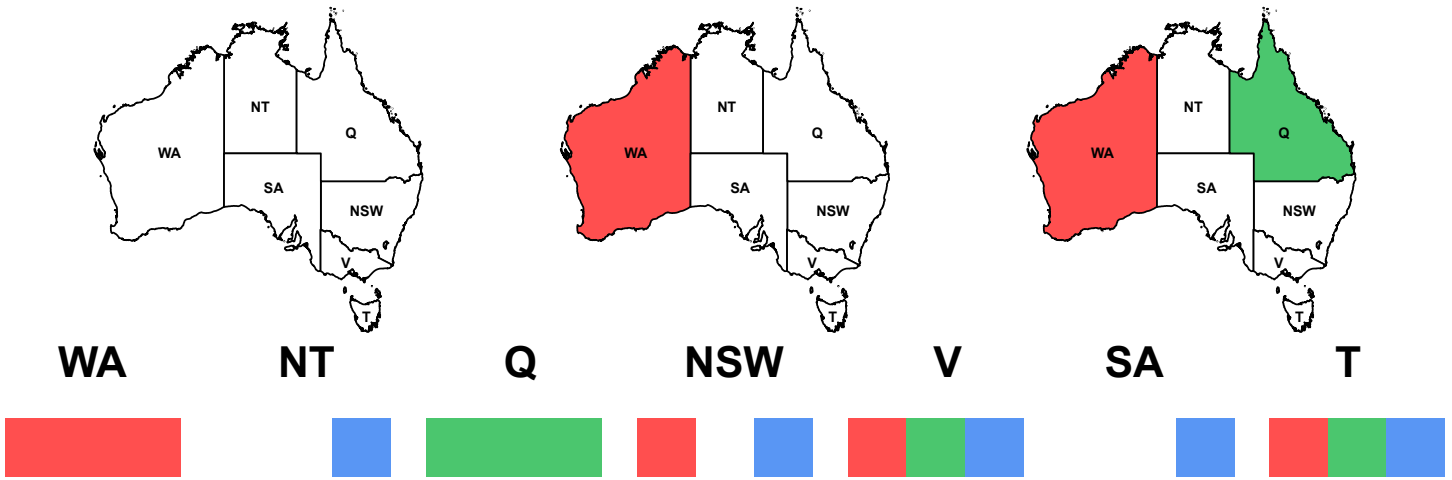
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
    - prune off that part of the search tree, and backtrack



Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent if *for* ***every*** *value x of X there is* ***some*** *allowed y*



If X loses a value, neighbours of X need to be rechecked.

Assignment 2 - CSP - Backtracking

Task 4.2

Write the code that solves Sudoku csp using a search algorithms of your choice, print the solution.

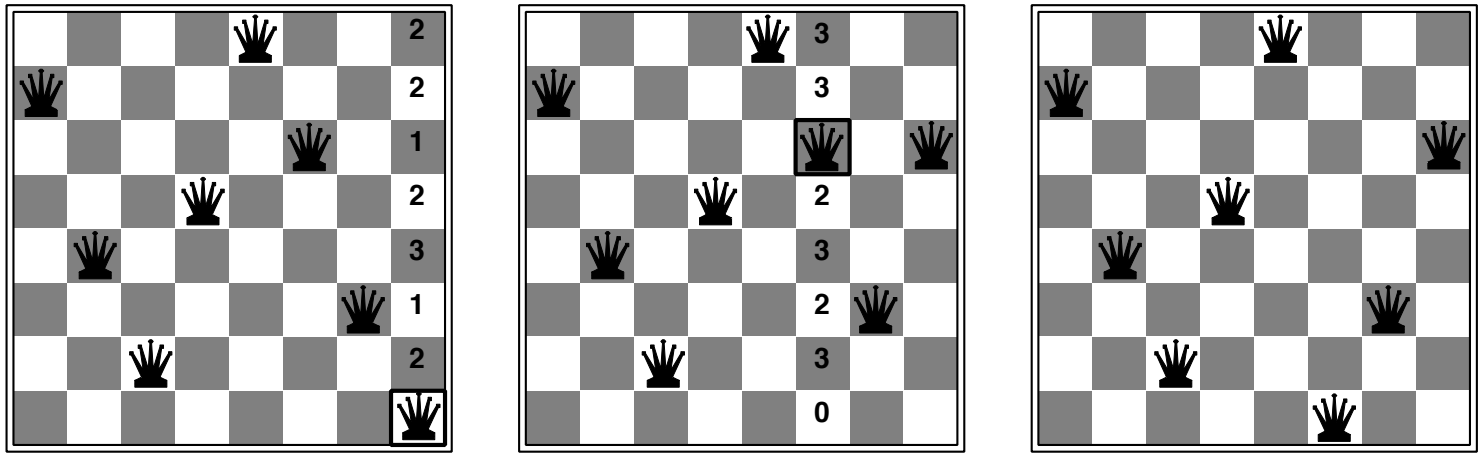
```
# your code
```

Task 4.3

Descibe what do nodes represent in the search tree. Explain your choice of the search algorithm and compare it with any other search algorithm implemented in this assignemnt.

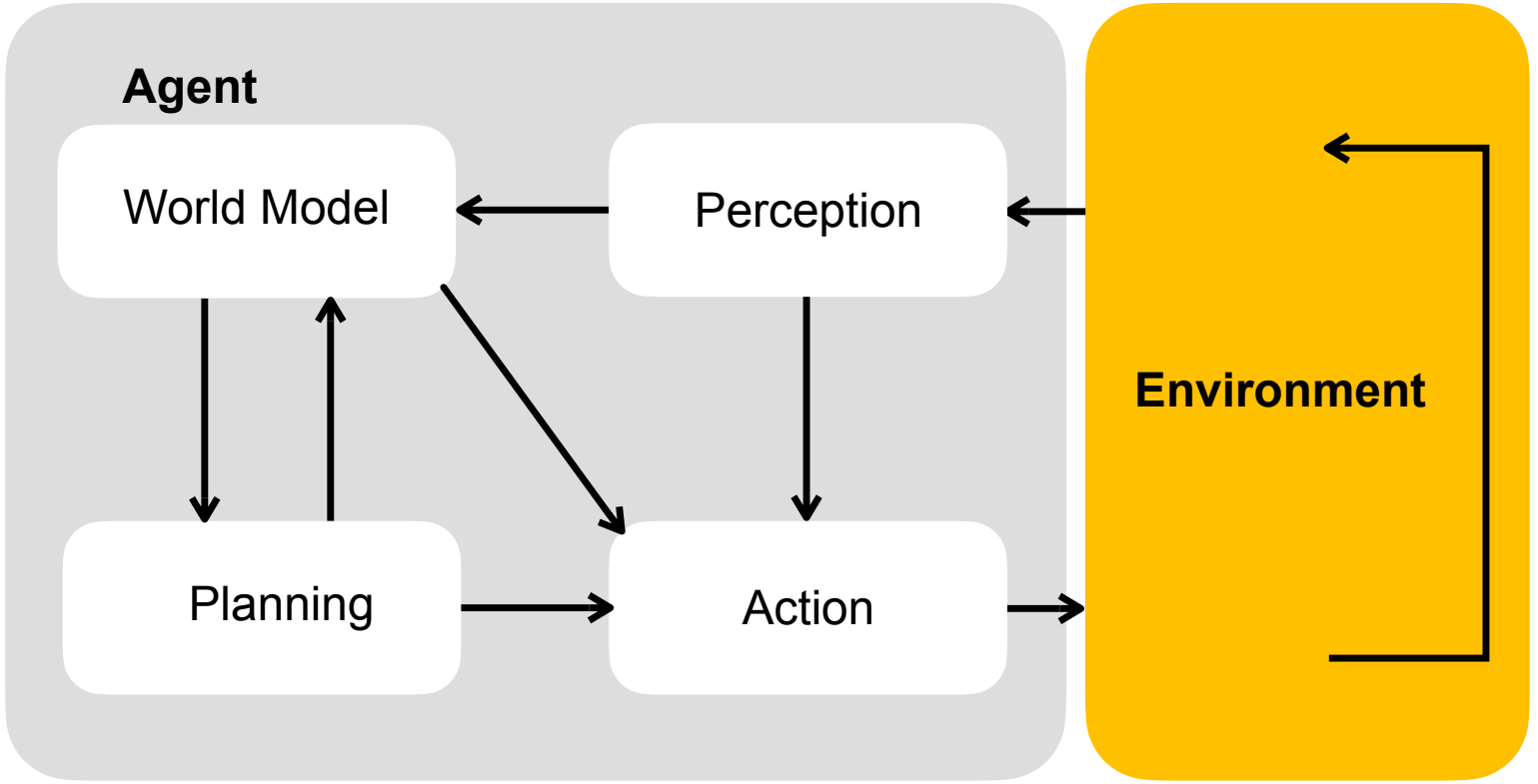
```
# your answer
```

Hill-climbing by min-conflicts



- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic
  - choose value that violates the fewest constraints

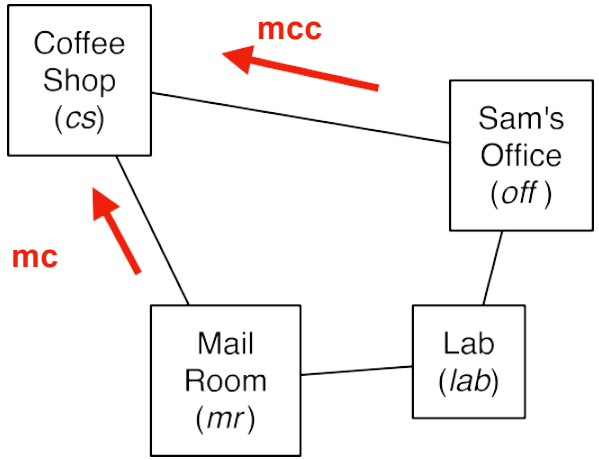
Planning Agent



Actions

- A deterministic **action** is a partial function from states to states.
- The **preconditions** of an action specify when the action can be carried out.
- The **effect** of an action specifies the resulting state.

Delivery Robot Example



Features:
RLoc – Rob's location
RHC – Rob has coffee
SWC – Sam wants coffee MW – Mail is waiting
RHM – Rob has mail

Features to describe states

Actions:
mc – move clockwise
mcc – move counterclockwise
puc – pickup coffee
dc – deliver coffee
pum – pickup mail
dm – deliver mail

Robot actions

STRIPS Representation

- Each action has a:
  - •precondition that specifies when the action can be carried out.
  - •effect a set of assignments of values to primitive features that are made true by this action.
  - •Often split into an ADD list (things that become true after action)
  - •and DELETE list (things that become false after action)

Assumption: every primitive feature not mentioned in the effects is unaffected by the action.

Example STRIPS Representation

Pick-up coffee (puc):

- precondition: [cs, ¬rhc]
- effect: [rhc]

Deliver coffee (dc):

- precondition: [off, rhc]
- effect: [¬rhc, ¬swc]

Feature-Based Representation of Actions

- For each action:
  - precondition is a proposition that specifies when the action can be carried out.

- For each feature:
  - causal rules that specify when the feature gets a new value and
  - frame rules that specify when the feature keeps its value.

Example Feature-Based Representation

- Precondition of pick-up coffee (puc):

  RLoc=cs $\wedge$ ¬rhc

- Rules for "location is cs":

  RLoc′=cs $\leftarrow$ Rloc = off $\wedge$ Act=mcc

  RLoc′=cs $\leftarrow$ Rloc = mr $\wedge$ Act=mc

  RLoc′=cs $\leftarrow$ Rloc = cs $\wedge$ Act $\neq$ mcc $\wedge$ Act $\neq$ mc

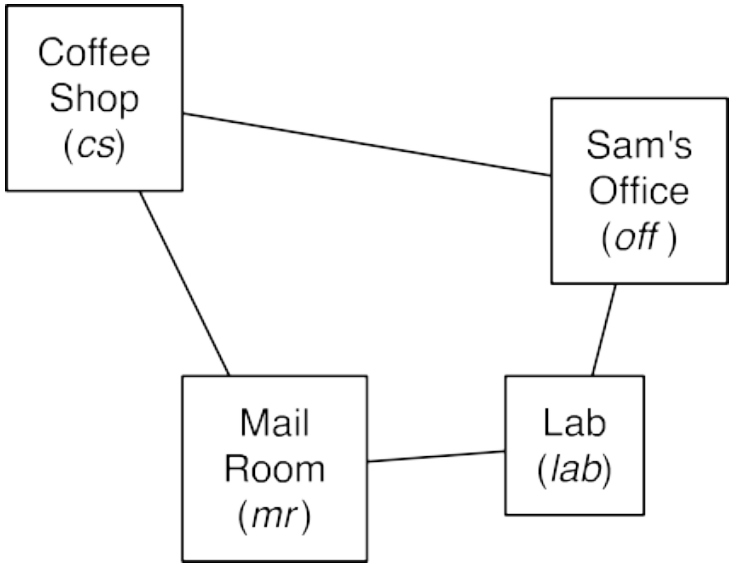
- Rules for "robot has coffee"

  rhc′ $\leftarrow$ Act=puc $\wedge$ ¬rhc

  rhc′ $\leftarrow$ rhc $\wedge$ Act $\neq$ dc

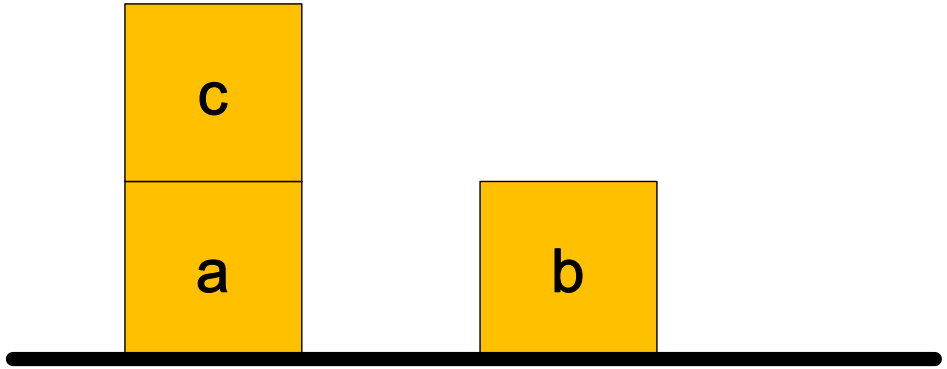
causal rules

frame rule



Defining Goals and Possible Actions

- Example of goals:
  on(a,b), on(b,c)
- Example of action:
  move(a, 1, b)
  (Move block a from 1 to b)
- Action preconditions:
  clear(a), on(a,1), clear(b)
- Action effects:
  on(a,b), clear(1), ~on(a,1), ~clear(b)



"add" (true after action)

"delete" (no longer true after action)
(~ = ¬)

Planning

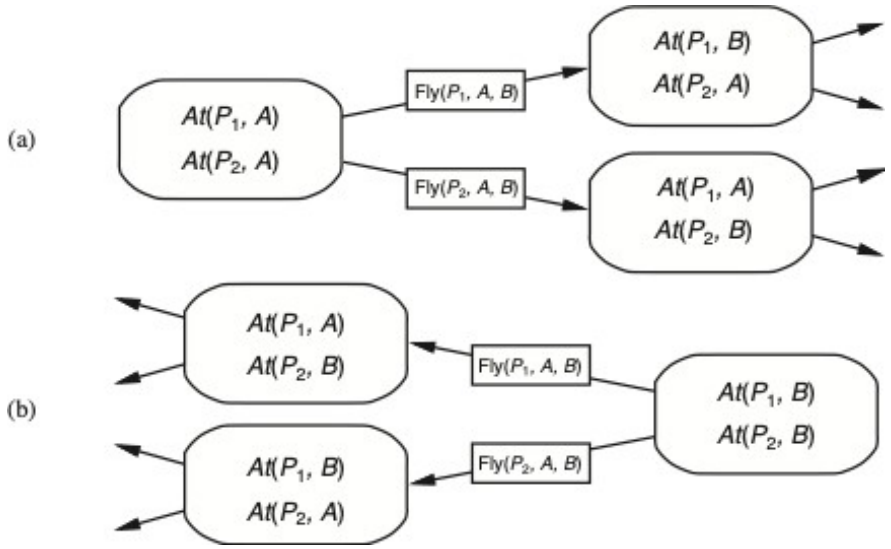
- Plan — sequence (or ordered set) of actions to achieve some goal
- Planner — problem solver that produces plans
- Goal – typically a conjunction of literals
- Initial State – typically a conjunction of literals

Blocks World Example for goal $on(B, C) \wedge on(C, Table)$

- $move(C, A, Table), move(B, Table, C)$

Simple Planning Algorithms

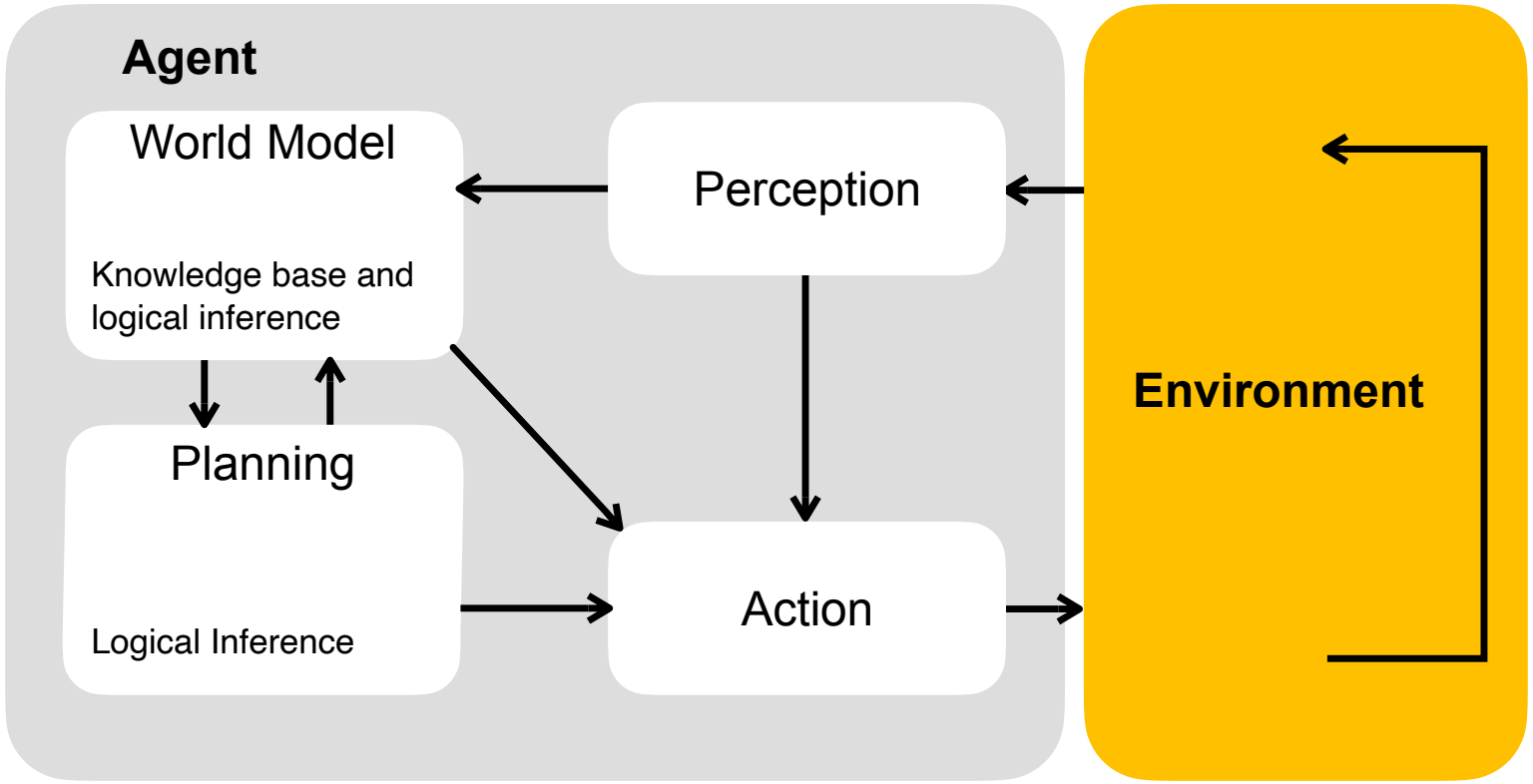
Forward search and goal regression



Planning

- Planning is the process of choosing a sequence of actions to achieve a goal.
- An action is a partial function from a state to a state.
- Two representations for actions that exploit structure in states are
  - the STRIPS representation, which is an action-centric representation,
  - the feature-based representation of actions, which is a feature-centric representation.
  - The feature-based representation is more powerful than the STRIPS representation; it can represent anything representable in STRIPS, but can also represent conditional effects.

Models and Planning



Knowledge Based Agent

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
- determine appropriate actions

Propositional Logic: Syntax

Propositional logic is the simplest logic

- "It is not the case that the sky is blue": $\neg B$
- (alternatively "the sky is not blue")
- "The sky is blue and the grass is green": $B \wedge G$
- "Either the sky is blue or the grass is green": $B \vee G$
- "If the sky is blue, then the grass is not green": $B \rightarrow \neg G$
- "The sky is blue if and only if the grass is green": $B \leftrightarrow G$
- "If the sky is blue, then if the grass is not green, the plants will not grow": $B \rightarrow (\neg G \rightarrow \neg P)$

Example – Complex Sentence

R	S	$\neg R$	R ∧ S	$\neg R \vee S$	$(R \wedge S) \rightarrow (\neg R \vee S)$
True	True	False	True	True	True
True	False	False	False	False	True
False	True	True	False	True	True
False	False	True	False	True	True

Thus $(R \wedge S) \rightarrow (\neg R \vee S)$ is a **tautology**

Propositional Logic

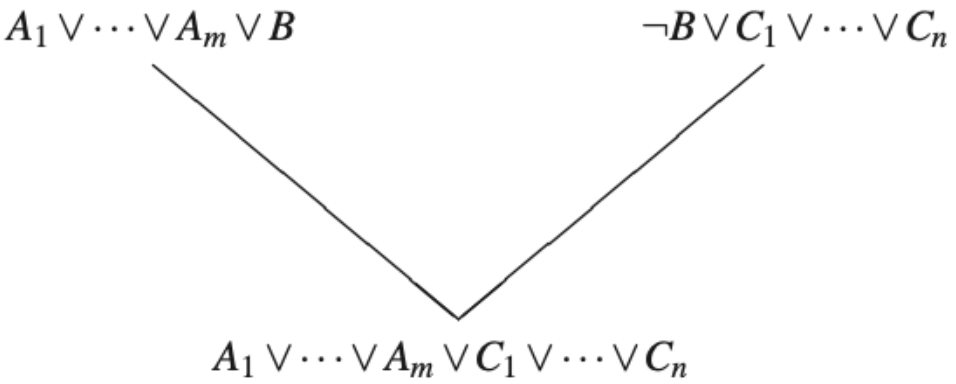
- Use letters to stand for "basic" propositions; combine them into more complex sentences using operators for **not, and, or, implies, iff** $(A \vee B \vee \neg C) \wedge (\neg B \vee D)$

- Propositional **connectives**:

$\neg$	*negation*	$\neg P$	*"not P"*
$\wedge$	*conjunction*	$P \wedge Q$	*"P and Q"*
$\vee$	*disjunction*	$P \vee Q$	*"P or Q"*
$\rightarrow$	*implication*	$P \rightarrow Q$	*"If P then Q"*
$\leftrightarrow$	*bi-implication*	$P \leftrightarrow Q$	*"P if and only if Q"*

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
True	True	False	True	True	True	True
True	False	False	False	True	False	False
False	True	True	False	True	True	False
False	False	True	False	False	True	True

Resolution



Consider A1 ∨⋯∨Am ∨B and ¬B∨C1 ∨⋯∨Cn

- Suppose both are True
- If B is True, ¬B is False and C1 ∨⋯∨Cn is True
- If B is False, A1 ∨⋯∨Am is True
- Hence A1 ∨⋯∨Am ∨C1 ∨⋯∨Cn is True

Conjunctive Normal Form

In order to apply Resolution, we must first convert the KB into Conjunctive Normal Form (CNF).

This means that the KB is a conjunction of clauses, and each clause is a disjunction of (possibly negated) literals.

$$(A \vee B \vee \neg C) \wedge (\neg B \vee D)$$

Conjunctive Normal Form

1. Eliminate double negations: rewrite ¬¬P as P

2. Eliminate → rewriting P → Q as ¬P ∨ Q

3. Use De Morgan's laws to push ¬ inwards (repeatedly)
   Rewrite ¬(P∧Q) as ¬P∨¬Q

4. Use the distributive laws to get CNF
   Rewrite (P∧Q)∨R as (P∨R)∧(Q∨R) [for CNF]

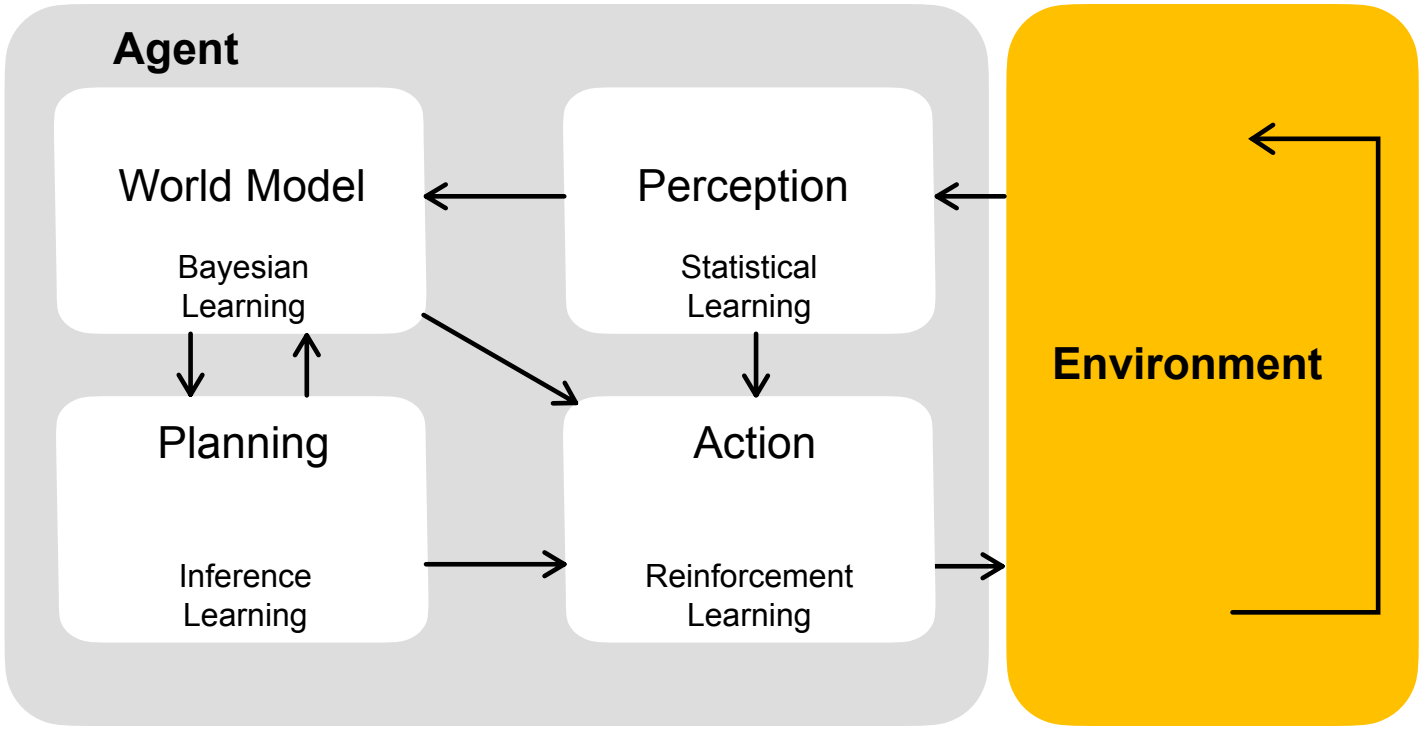
Resolution by Refutation

- Negate query to be proven (resolution is a refutation system)
- Convert knowledge base and negated query into CNF
- Repeatedly apply resolution to clauses or copies of clauses until either the empty clause (contradiction) is derived or no more clauses can be derived (a copy of a clause is the clause with all variables renamed)
- If the empty clause is derived, answer ‘yes’ (query follows from knowledge base), otherwise answer ‘no’ (query does not follow from knowledge base) . . . and if there are an infinite number of clauses that can be derived, don’t answer at all

Syntax of First Order Logic

- Objects: people, houses, numbers, theories, colors, football games, wars, centuries ...
- Predicates: red, round, bogus, prime, multistoried, ...
  brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
- Functions: father of, best friend, third inning of, one more than,...

Learning Agent



Types of Learning

- Supervised Learning
  - Agent is presented with examples of inputs and their target outputs, and must learn a function from inputs to outputs that agrees with the training examples and generalises to new examples
- Reinforcement Learning
  - Agent is not presented with target outputs for each input, but is periodically given a reward, and must learn to maximize (expected) rewards over time
- Unsupervised Learning
  - Agent is only presented with a series of inputs, and must find and aims to find structure in these inputs

Supervised Learning

- Given a training set and a test set, each consisting of a set of items for each item in the training set, a set of features and a target output
- Learner must learn **a model** that can **predict** the target output for any given item (characterised by its set of features)
- Learner is given the input features and target output for each item in the training set
  - Items may be presented all at once (batch) or in sequence (online)
  - Items may be presented at random or in time order (stream)
  - Learner **cannot** use the test set **at all** in defining the model
- Model is evaluated by its performance on predicting the output for each item in the **test set**

Inductive learning

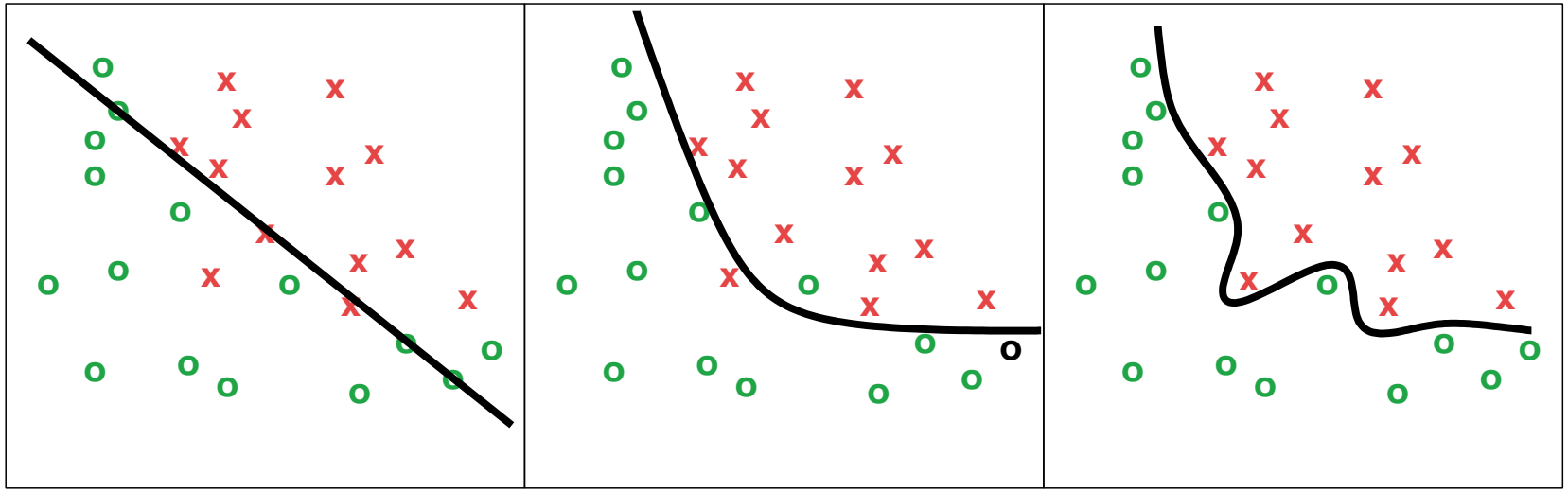
Simplest form: learn a function from examples

f is the target function
An example is a pair *(x, f(x))*

Problem: find a hypothesis *h*
such that $h \approx f$
given a training set of examples

Ockham's razor

"The most likely hypothesis is the **simplest** one consistent with the data."

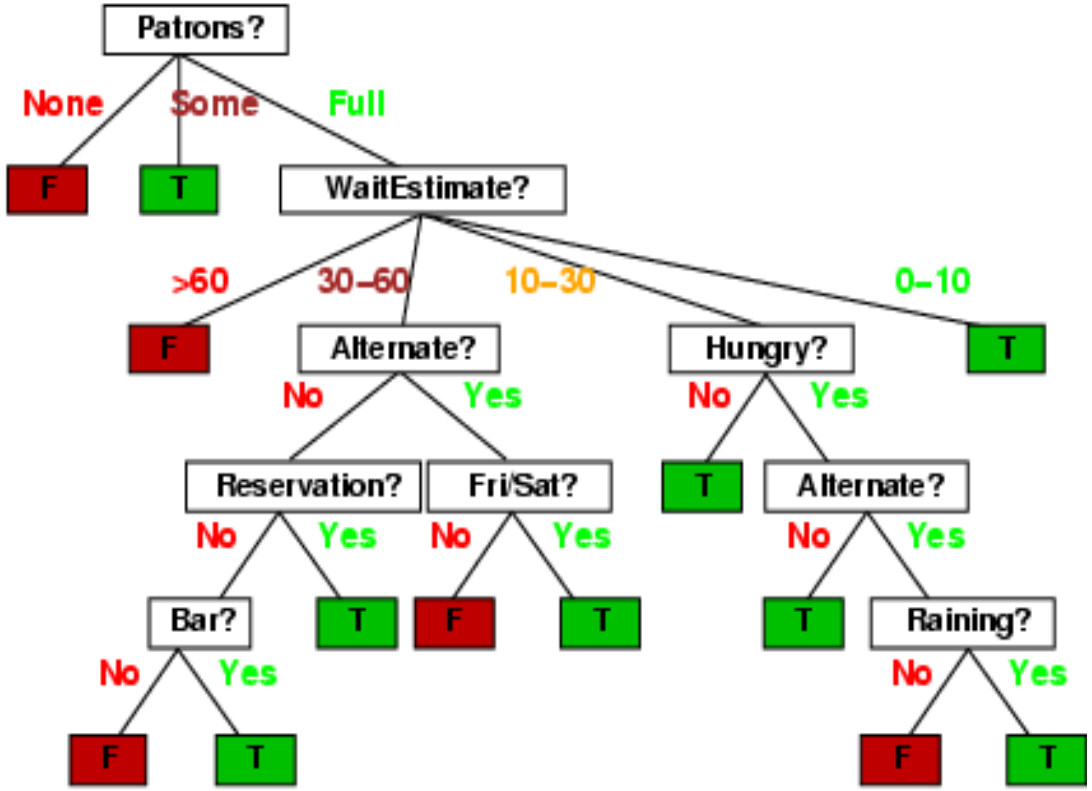


inadequate | good compromise | over-fitting

Since there can be **noise** in the measurements, in practice need to make a **tradeoff** between simplicity of the hypothesis and how well it fits the data.

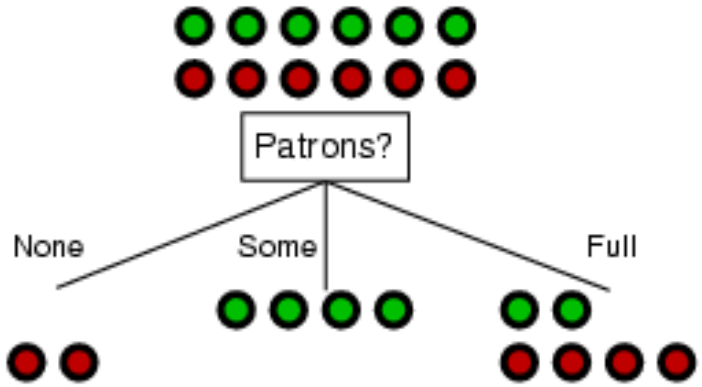
Decision trees

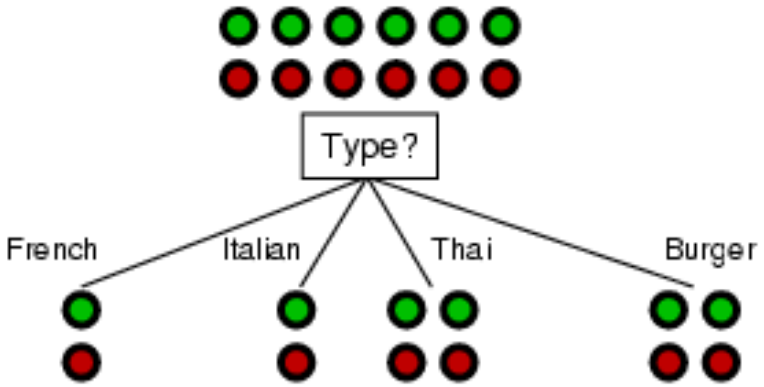
- One possible representation for hypotheses
  - E.g., here is the “true” tree for deciding whether to wait:



Choosing an attribute

- Patrons is a “more informative” attribute than Type, because it splits the examples more nearly into sets that are “all positive” or “all negative”.
- An important (or good) attribute splits samples into groups that are (ideally) all positive or negative.





Entropy

Entropy is a measure of how much information we **gain** when the target attribute is revealed to us. In other words, it is not a measure of how much we know, but of how much we don't know.

If the prior probabilities of the n target attribute values are $p_1, .., p_n$ then the entropy is

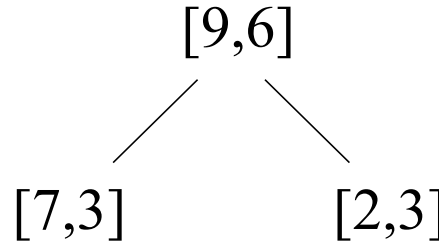
$$H(<p_1, .., p_n>) = \sum - p_i \log_2 p_i$$

Minimal Error Pruning

Should the children of this node be pruned or not?

Left child has class frequencies [7,3]

$$E = 1 - \frac{n+1}{N+k} = 1 - \frac{7+1}{10+2} = 0.333$$



Right child has $E = 0.429$

Parent node has $E = 0.412$

Average for Left and Right child is

$$E = \frac{10}{15}(0.333) + \frac{5}{15}(0.429) = 0.365$$

Since $0.365 < 0.412$, children should NOT be pruned.

Probability and Uncertainty

Subjective or Bayesian probability:

Probabilities relate propositions to one's own state of knowledge e.g. $P(A30|\text{no reported accidents}) = 0.06$

These are not claims of a "probabilistic tendency" in the current situation (but might be learned from past experience of similar situations)

Probabilities of propositions change with new evidence:

e.g. $P(A30|\text{no reported accidents, 5 a.m.}) = 0.15$

Prior probability

Prior or unconditional probabilities of propositions
e.g. $P(\texttt{Cavity = true}) = 0.1$ and $P(\texttt{Weather = sunny}) = 0.72$
correspond to belief prior to arrival of any (new) evidence.

Probability distribution gives values for all possible assignments:
$P(\texttt{Weather}) = \langle 0.72, 0.1, 0.08, 0.1 \rangle$ (normalized, i.e., sums to 1)

Inference by Enumeration

Start with the joint distribution:

	toothache		~toothache	
	catch	~catch	catch	~catch
cavity	0.108	0.012	0.072	0.008
~cavity	0.016	0.064	0.144	0.576

Can also compute conditional probabilities:

$$P(cavity \mid toothache) = \frac{P(cavity \wedge toothache)}{P(toothache)} = \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064}$$

Conditional Probability

If we consider two random variables a and b, with $P(b) \neq 0$, then the conditional probability of a given b is

$$P(a \mid b) = \frac{P(a \wedge b)}{P(b)} \quad \text{provided } P(b) > 0 \qquad P(a \wedge b) = P(a \mid b) \cdot P(b) = P(b \mid a) \cdot P(a)$$

When an agent considers a sequence of random variables at successive time steps, they can be chained together using this formula repeatedly:

$$P(X_n, \ldots, X_1) = P(X_n \mid X_{n-1}, \ldots, X_1) P(X_{n-1}, \ldots, X_1)$$

$$= P(X_n \mid X_{n-1}, \ldots, X_1) P(X_{n-1} \mid X_{n-2}, \ldots, X_1)$$

$$= \ldots = \prod_{i=1}^{n} P(X_i \mid X_{i-1}, \ldots, X_1)$$

Bayes' Rule

The formula for conditional probability can be manipulated to find a relationship when the two variables are swapped:

$$P(a \wedge b) = P(a \mid b) \cdot P(b) = P(b \mid a) \cdot P(a)$$

$$\rightarrow \text{Bayes' Rule } P(a \mid b) = \frac{P(b \mid a)\, P(a)}{P(b)}$$

This is often useful for assessing the probability of an underlying **cause** after an **effect** has been observed:

$$P(Cause \mid Effect) = \frac{P(Effect \mid Cause)\, P(Cause)}{P(Effect)}$$

Joint Probability Distribution

We assume there is some underlying joint probability distribution over the three random variables Toothache, Cavity and Catch, which we can write in the form of a table:

	toothache		~toothache	
	catch	~catch	catch	~catch
cavity	0.108	0.012	0.072	0.008
~cavity	0.016	0.064	0.144	0.576

Note that the sum of the entries in the table is 1.0 .
For any proposition φ, sum the atomic events where it is true:

$$P(\varphi) = \Sigma_{\{\omega:\, \omega \models \varphi)\}}\ P(\omega)$$

Bayesian Networks

- A Bayesian network (also Bayesian Belief Network, probabilistic network, causal network, knowledge map) is a directed acyclic graph (DAG) where
  - Each node corresponds to a random variable
  - Directed links connect pairs of nodes – a directed link from node
  - *X* to node *Y* means that *X* has a direct influence on *Y*
  - Each node has a conditional probability table quantifying effect of parents on node
- Independence assumption of Bayesian networks
  - Each random variable is (conditionally) independent of its nondescendants given its parents

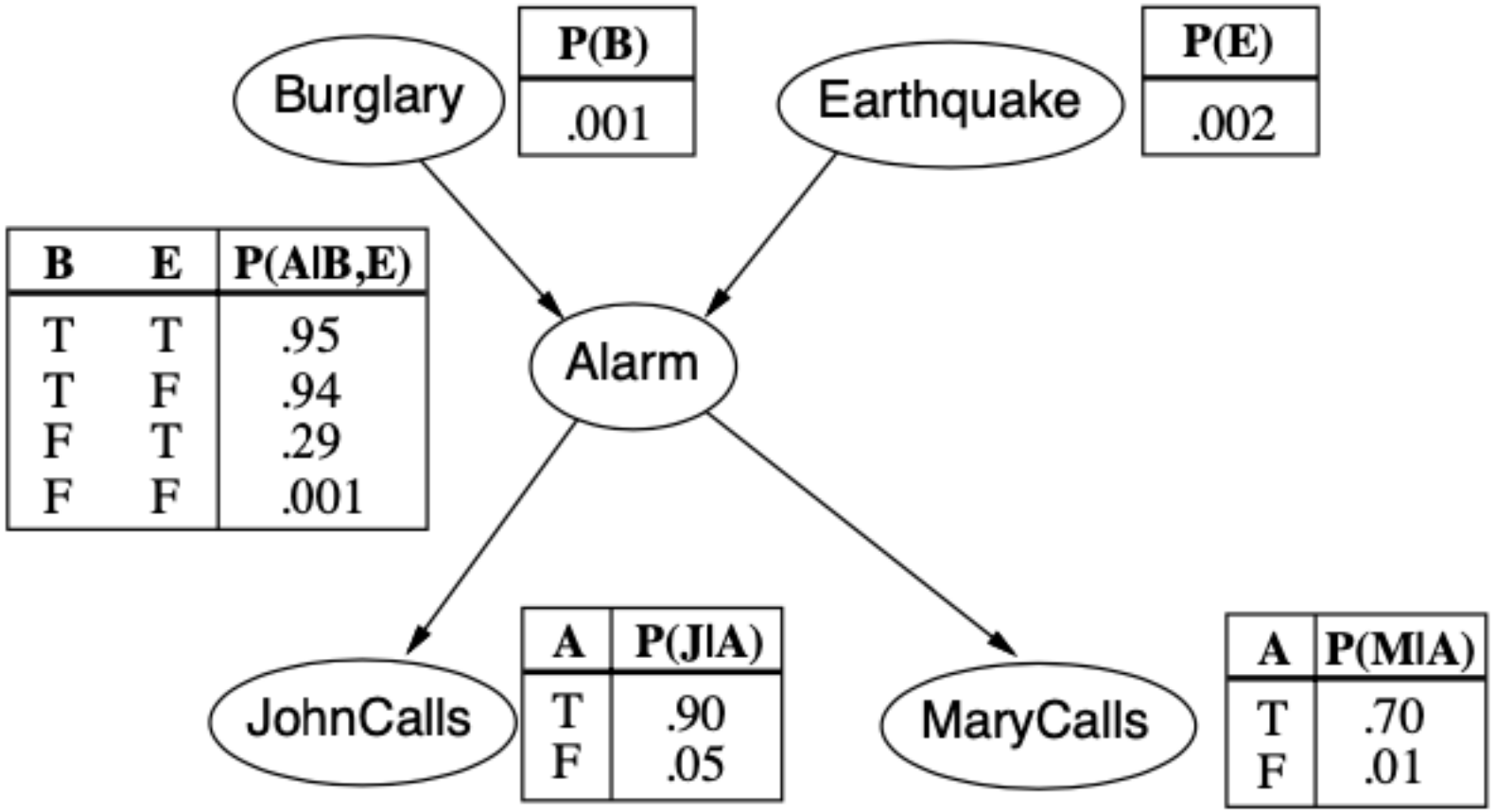
Belief Networks

- When there are multiple minimal sets of predecessors satisfying this condition, any minimal set may be chosen to be the parents.
  - There can be more than one minimal set only when some of the predecessors are deterministic functions of others.

- Putting the chain rule and the definition of parents together gives:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid parent(X_i))$$

Bayesian Networks



Artificial Neural Networks

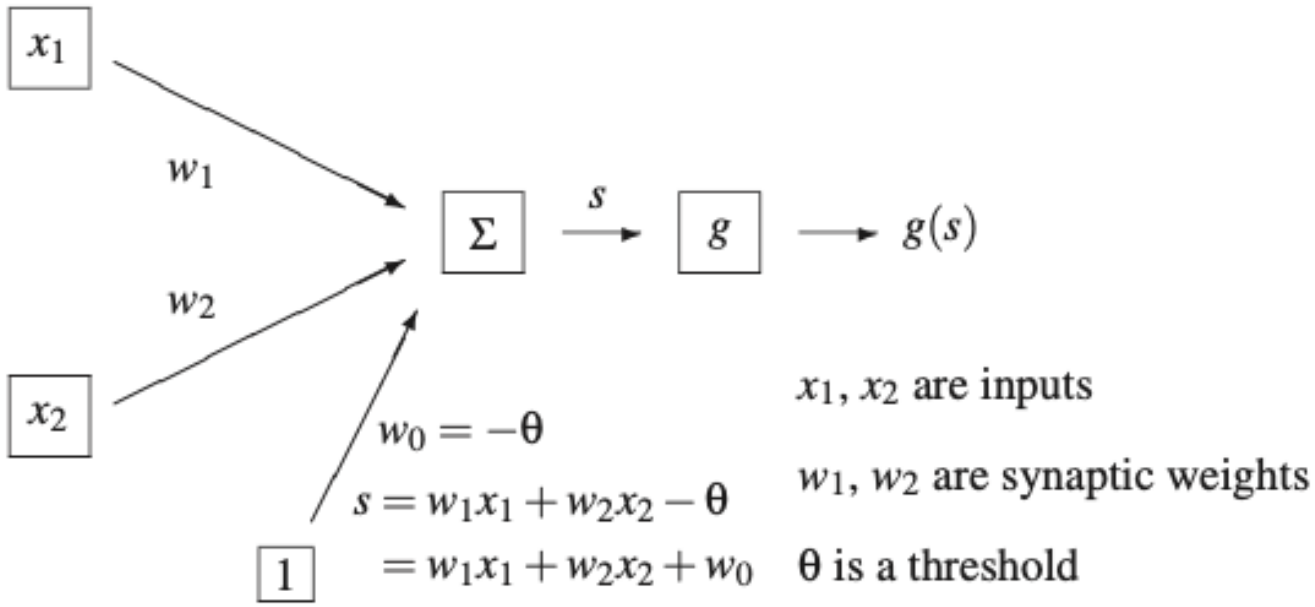
- (Artificial) Neural Networks are made up of nodes which have
  - inputs edges, each with some **weight**
  - outputs edges (with **weights**)
  - an **activation level** (a function of the inputs)

- Weights can be positive or negative and may change over time (learning).
- The **input function** is the weighted sum of the activation levels of inputs.
- The activation level is a non-linear **transfer** function *g* of this input:

$$\text{activation}_i = g(s_i) = g\left(\sum_j w_{i\,j} x_j\right)$$

Some nodes are inputs (sensing), some are outputs (action)

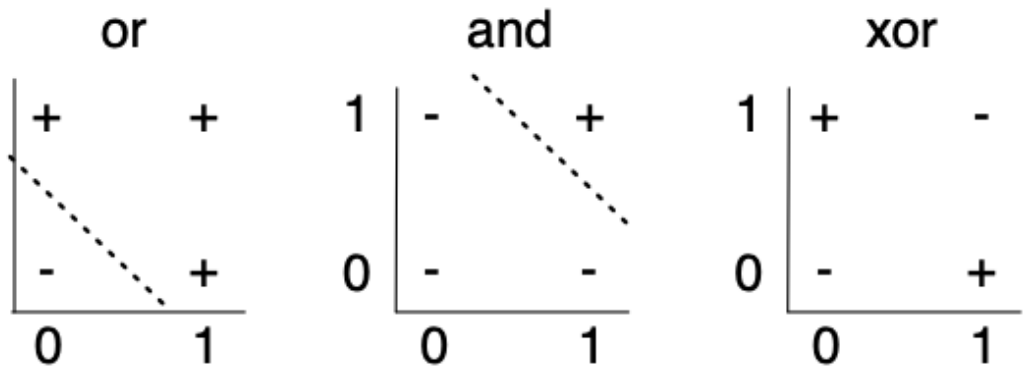
McCulloch & Pitts Model of a Single Neuron



Linearly Separable

- A classification is **linearly separable** if there is a hyperplane where the classification is true on one side of the hyperplane and false on the other side.
- For the sigmoid function, the hyperplane is when: $w_0 + w_1 X_1 + \cdots + w_n X_n = 0$

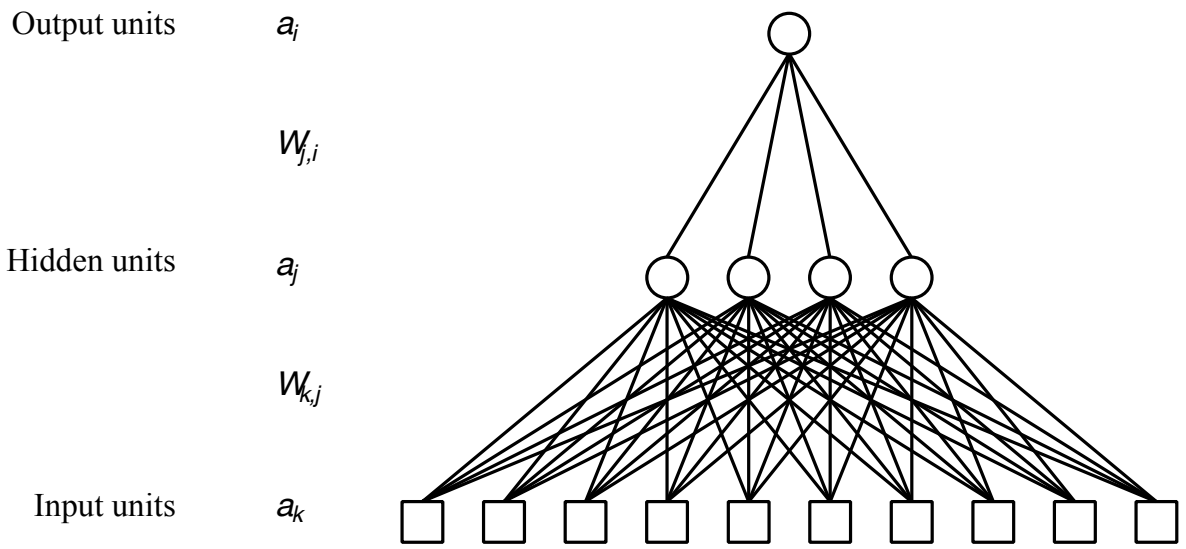
  This separates the predictions > 0.5 and < 0.5.
- Linearly separable implies the error can be arbitrarily small



Multilayer perceptrons

Layers are usually fully connected;

numbers of hidden units typically chosen by hand



Gradient Descent

Recall that the error function *E* is (half) the sum over all input patterns of the square of the difference between actual output and desired output

$$E = \frac{1}{2}\sum (z - t)^2$$

The aim is to find a set of weights for which *E* is very low.

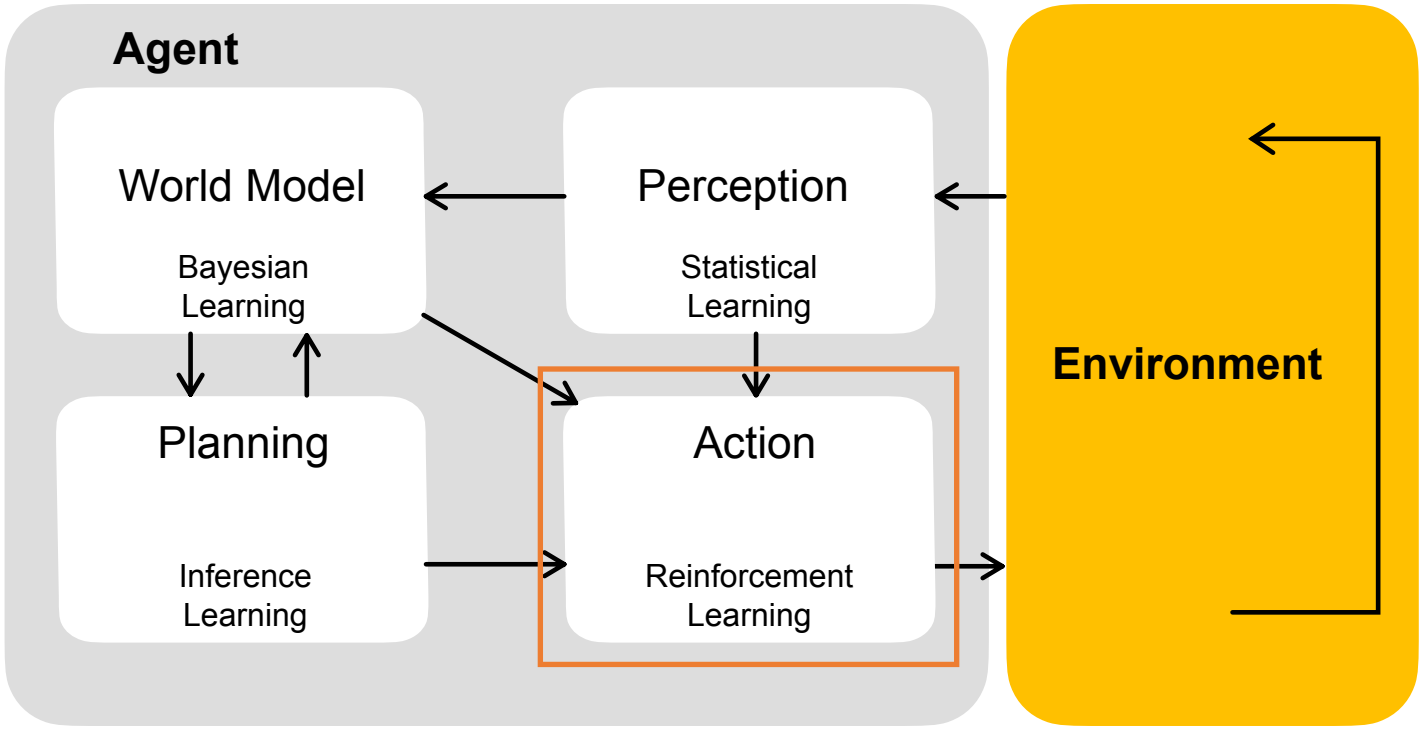
If the functions involved are smooth, we can use multi-variable calculus to adjust the weights in such a way as to take us in the steepest downhill direction.

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

Parameter η is called the learning rate.

- How the cost func effects the particular weight
- Find the weight

Learning Agent



States and Actions

- Each node is a *state*
- *Actions* cause transitions from one state to another
- A *policy* is the set of transition rules
  - i.e. which action to apply in a given state
- Agent receives a *reward* after each action
- Actions may be non-deterministic
  - Same action may not always produce same state

Reinforcement Learning Framework

An agent interacts with its environment.

There is a set *S* of *states* and a set *A* of *actions*.

At each time step *t*, the agent is in some state s_t. It must choose an action a_t , whereupon it goes into state

$s_t + 1 = \delta(s_t, a_t)$ and receives reward $r(s_t, a_t)$.

In general, *r*() and δ() can be multi-valued, with a random element

The aim is to find an optimal *policy* $\pi : S \rightarrow A$ which will maximize the cumulative reward.

Q-Learning – summary

Q Value

- How to choose an action in a state?

$$Q(s, a) = r(s, a) + \gamma V^*(s')$$

- The Q value for an action, a, in a state, s, is the immediate reward for the action plus the discounted value of following the optimal policy after that action
- V^* is value obtained by following the optimal policy
- $s' = \delta(s, a)$ is the succeeding state, assuming the optimal policy

Q-Learning – summary

Q Learning

initialise $Q(s,a) = 0$ for all s and a

observe current state s

repeat

select an action a and execute it

observe immediate reward r and next state s'

$Q(s,a) \leftarrow r + \max_{a'} Q(s',a')$

$s \leftarrow s'$

Examination Instructions

- Date and time: Saturday, 06 February 2021 at 2pm (Sydney time)
- Online mode: Moodle

- Reading time: 10 mins
- Time allowed: 2 hours

- This examination counts for 50% of the final mark
- Total number of questions: 36
- Answer all questions

Examination Instructions

- Questions: multiple choice & open-ended questions
- Questions can have different weight (open-ended questions weight more)

- Similar to the questions from tutorials & assignments
- Some inspired by your homework

- No mark taken for wrong answers

- You can use your personal notes

- **You cannot use any other material (including internet)**
- **Plagiarism policy**

- **Distributing, copying, printing is not permitted.**

Sample question

Completeness of a search algorithm answers the question:

(a) Is the algorithm guaranteed to find a solution when there is one?

(b) Does the strategy find the solution that has the lowest path cost of all solutions?

(c) How long does it take to find a solution?

(d) How much memory is needed to perform the search?

Sample question

Consider this joint probability distribution:

	short		¬ short	
	wide	¬ wide	wide	¬ wide
striped	0.07	0.05	0.08	0.12
¬ striped	0.14	0.17	0.12	0.25

Compute (to two decimal places): Prob(short ∨ ¬ wide | striped)

(a) 0.50
(b) 0.63
(c) 0.75
(d) 0.80

Sample question

Explain why depth-first search is used in the search tree for Constraint Satisfaction Problems. Compare it with breath-first search.

Beyond COMP3411/9814

- COMP9444 Neural Networks and Deep Learning
- COMP9417 Machine Learning and Data Mining
- COMP4418 Knowledge Representation and Reasoning
- COMP3431 Robotic Software Architecture
- COMP9517 Machine Vision
- 4th Year Thesis topics

UNSW myExperience Survey

Please remember to fill in the UNSW myExperience Survey.

Questions?

Good luck!