CS50: Introduction to IA with Python

Introduction

Plan
- → Search
- → knowledge
- → Uncertainty
- → optimization
- → Machine Learning
- → Neural Networks
- → Language.

Chapter 1: Search

Def : **An agent** is an entity that perceives its environment and acts upon that environment

Def : **A state** : a configuration of the agent and its environment

Def : **An action** : choices that can be made in a state.
function : Actions(s) → output : set of actions that can be executed
(s : state input)

Def : **Transition model** : a description of what states results from performing any applicable action in any state
function : Results(s, a) : → output : state resulting from performing a in state s.

Def : **State Space** : the set of all states reachable from the initial state by any sequence of actions

Representation :

- nodes (state)
- → actions

Def. goal test: way to determine wether a given state is a goal state

Def: path cost: Numerical cost associated with a given cost

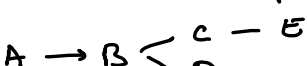
=> goal: find goal state with lowe path cost.
=> optimal solution

=> Search Problem definition
- Init state
- action
- transit model
- goal test
- path cost function

Def: a node: data structure that keep tracks of:
- a state
- a parent (previous node)
- an action (applied to parent)
- a path cost (from init state to node)

Approach:
- start with a frontier that contains the initial state.
- repeat:
  - → if frontier is empty, then no solution
  - → Remove a node from the frontier
  - → if node contains goal state, return the solution
  - → Expend node, add resulting nodes to the frontier

Ex: Find path from A to E.



Start init: Frontier = A.
→ Remove A (Not goal
→ Expend

Step 1: Frontier = B
→ Remove B (Not goal)
→ Expend

Step 2: Frontier = C, D
→ Remove D (Not goal) (arbitrarily)
→ Expend

Step 3: Frontier = C, F
→ Remove C
→ Expend

Step 4: Frontier = E, F
→ Remove E: find solution
A → B → C → E

⚠ if we can go back to a state it can create infinite loop between 2 states.

→ **Revised Approach**
- → Start frontier that contains the initial state
- → Start with an empty explored set
- → Repeat
  - If frontier is empty, then no solution
  - Remove a node from frontier
  - if node contains a goal state, return solution
  - Add Node to explored set
  - Expend Node, add resulting nodes to the frontier if they aren't already in the frontier or the explored set

⚠ How we remove a node is super important.

a. Uniformed Search

Def : **Stack** : Last-in, first-out data type

Def **Depth-first search** : search algo that always expend the deepest node in the frontier (
↳ what happen if we use stack as frontier, we will go as deep as possible in a branch and change only if we reach the end.

Def **Breadth-first search** : search algo that always expend the shallowest node in the frontier
↳ use opposite of stack as frontier

Def **queue** : first in, first out data type (opposite of stack)

⇒ If finit state both algo will always find a solution.
However, Breath first search will always find the best solution (≠ Depth-first algo)

⚠ Need to explore a lot of state to find the solution

Def **Uninformed search** : search strategy that uses no problem-specific knowledge.
(Ex BFS / DFS)

Def **Informed search** : search strategy that uses problem-specific knowledge find a solution more efficiency.

b. Informed Search

Def : **Greedy best-first search**: search algo that expend the node that is closest to the goal, as estimated by a heuristic function $h(n)$

Def **A* search**: search algo that expends node with lowest value of $g(n) + h(n)$
- $g(n)$: cost to reach node
- $h(n)$: estimated cost to goal.

⚠ GBFS do not always find optimal solution

A* search optimal if:
- $h(n)$ is **admissible** (never over estimate the true cost)
- $h(n)$ is **consistent** (for every node n and successor n' with step cost c, $h(n) \leq h(n') + c$)

c. Adversial approach.

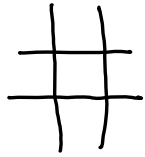
Adversial search : someone try to make you fail.

ex: morpion. (X et O)

X wins = 1, O wins = -1, no one = 0

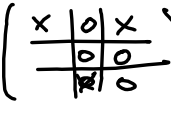
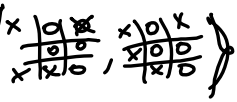
Def **Minimax** : give a value to each output possible.
- Max(X) aims to max score
- Min (O) aims to min score.

Game:
- s_0 : initial player
- Player(s) : returns which player to move in state s
- Actions : returns legal moves in state s
- Results(s, a) : return state after apply a in state s
- terminal(s) : check if state s is a terminal state
- Utility(s) : final numerical value for terminal state s.

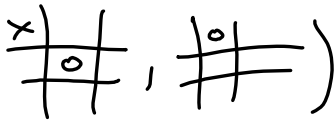
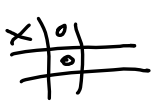
Ex: Init 

player() = X

(X start)

Actions() = {  }

player() = O

Results() = 

Terminal() = True.

MinMax Algo:

- given a state s:
  - Max picks action in Actions(s) that produces highest value of Min-Value(Results(s, a))
  - Min picks action in Actions(s) that produces smallest value of Max-Value(Results(s, a)).

```
function Max-Value(state):
    if Terminal(state):
        return Utility(state).
    v = -∞
    for action in Actions(state):
        v = Max(v, Min-Value(Result(state, action))
    return v.

function Min-Value(state):
    if Terminal(state):
        return Utility(state).
    v = +∞
    for action in Actions(state):
        v = Min(v, Max-Value(Result(state, action))
    return v.
```

→ Optimizations

As we maximize (or minimize) between all possibility it is possible that we don't have the best but we maximize the min value we will have.

Depth limited Min Max :

Same but stop after n iteration even if game not stop

Need evaluation function instead of utility to find who is likely to win.

(ex: in chess : 10^{29000} possibility so it is a way to reduce calculation)

Chapter 2 : Knowledge

Def : **knowledge - based agents** : agent that reason by operating on internal representat° knowledge.

a. Logic

Def : **Sentence** : assert° about the world in a knowledge representation language.

⇒ **Propositional Logic**

- **Proposition Symbol** :

P Q R

- **Logical connective**

¬ : not ∧ : and ∨ : or → : implicat° ↔ : biconditional

Ex : P = True : ¬P : false

P	Q	P ∧ Q	P ∨ Q	P → Q	P ↔ Q
false	false	false	false	True	True
false	True	false	True	True	false
True	false	false	True	false	false
True	True	True	True	True	True

Def : **model** : assignm° of a truth value to every propositional symbol ("possible world")

ex : P : it is Raining model : {P = true, Q = false}
Q : it is Tuesday

Def: knowledge base: a set of sentences known by a knowledge-base agent.

Def: Entailment: $\alpha \models \beta$. (read "α entails β")
In every model in which sentence α is true, sentence β is also true

Ex. P: it is a Tuesday in February
$P \models Q$ Q: it is February

Def inference: the process of deriving new sentences from old ones.

Ex: P: it is Tuesday
Q: it is raining
R: Harry will go for a run.

knowledge base
KB: • $(P \wedge \neg Q) \rightarrow R$: If it is Tuesday and not raining, Harry will go for run
• P
• $\neg Q$

Inference → R is true.

Algo goal: $KB \models \alpha$?

b. Model Checking algo.

Algo
- To determine if $KB \models \alpha$:
  - → enumerate all possible models
  - → if in every model where KB is true, α is true then KB entails α.
- otherwise, KB does not entails α.

Ex: Same P, Q and R.

KB: $(P \wedge \neg Q) \rightarrow R$, P, Q

Query: R (what we are looking)

P	Q	R	KB	
false	false	false	false	
false	false	True	false	} Because P is True
false	True	false	false	
false	True	True	false	
True	false	false	false	} Because $(P \wedge \neg Q) \rightarrow R$
True	false	True	True	
True	True	false	false	} Because $\neg Q$
True	True	True	false	

⇒ One possible world: in this world R is True
so R is True. (where KB True)

c. Knowledge Engineering.

Create algo to resolve sol with knowledge.

Ex: Clued, Logic Puzzle.

⚠ Model checking is not very efficient (2^N N: variables)

d. Inference Rules

Modus Ponens.

if $\alpha \rightarrow \beta$ and α then β

And elimination

if $\alpha \wedge \beta$ then α

Double negat elimination

if $\neg(\neg\alpha)$ then α

Implication Elimination

if $\alpha \rightarrow \beta$ then $\neg \alpha \vee \beta$

Biconditional Elimination

if $\alpha \leftrightarrow \beta$ then $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$

De Morgan's Law

if $\neg(\alpha \wedge \beta)$ then $\neg \alpha \vee \neg \beta$

if $\neg(\alpha \vee \beta)$ then $\neg \alpha \wedge \neg \beta$

Distributive property

if $(\alpha \wedge (\beta \vee \gamma))$ then $(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

if $(\alpha \vee (\beta \wedge \gamma)$ then $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

d. Theorem proving.

Theorem Proving:

- Initial state: starting knowledge base (KB)
- actions: inference rules
- transit. models: new KB after inference.
- goal test: check statement we are trying to prove
- path cost: number of steps in proof.

rule: if $(P \vee Q) \wedge \neg P$ then Q

Generalise: if $(P \vee Q_1 \vee \ldots \vee Q_n) \wedge \neg P$ then $(Q_1 \vee \ldots \vee Q$

Rule: if $(P \vee Q)$ and $(\neg P \vee R)$ then $Q \vee R$

Generalise if $(P \vee Q_1 \vee \ldots \vee Q_n)$ and $(\neg P \vee R_1 \vee \ldots \vee R_n)$
then $Q_1 \vee \ldots \vee Q_n \vee R_1 \vee \ldots \vee R_n$

Def **clause** : a disjunction of literals (ex : $P \vee Q \vee R$)

Def **conjonctive normal form** : Logical sentence that is a conjunction of clauses. (ex: $\underbrace{(A \vee B \vee C)}_{\text{clause 1}} \wedge \underbrace{(D \vee \neg E)}_{\text{clause 2}} \wedge \underbrace{(F \vee G)}_{\text{clause 3}}$; conjonction)

Every logical sentence can have this form. with inference rule.

Conversion to CNF (conjonctive normal form)

- Eliminate biconditionals:
  - turn $(\alpha \leftrightarrow \beta)$ into $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$
- Eliminate implication
  - turn $(\alpha \rightarrow \beta)$ into $\neg\alpha \vee \beta$
- Move $\neg$ inwards using De Morgan's Laws
  - ex turn $\neg(\alpha \wedge \beta)$ into $\neg\alpha \vee \neg\beta$
- use distributive law to distribute $\vee$ wherever is possible.

Ex: $(P \vee Q) \rightarrow R$

$\neg(P \vee Q) \vee R$	eliminate implicat.
$(\neg P \wedge \neg Q) \vee R$	De Morgan's Law
$(\neg P \vee R) \wedge (\neg Q \vee R)$	Distributive Law

e. Inference by Resolution

Special case :

if $(P \vee Q \vee S) \wedge (\neg P \vee R \vee S)$

then $(Q \vee S \vee R \vee S)$

⚠ need to eliminate duplicate variable

$\Rightarrow (Q \vee R \vee S)$

Special case: if $P \wedge \neg P$ then $()$ (equivalent False)

Inference by resolution Principle.

- To determine $KB \models \alpha$
  - check if $(KB \wedge \neg\alpha)$ is a contradiction
    - if so, then $KB \models \alpha$
    - otherwise, no entailment.

Inference by resolution Algo: to determine if $KB \models \alpha$:

- convert $(KB \wedge \neg\alpha)$ to CNF
- keep checking to see if we can use resolution to produce a new clause.
  - if ever we produce the empty clause (equivalent to false), we have a contradiction and $KB \models \alpha$
  - otherwise, if we can't add new clauses, no entailment

Ex: Does $(A \cup B) \wedge (\neg B \vee C) \wedge (\neg C)$ entail A

- $KB \wedge \neg A = (A \cup B) \wedge \underbrace{(\neg B \vee C)}_{\hookrightarrow \neg B} \wedge \underbrace{(\neg C)}_{\hookrightarrow \neg B} \wedge (\neg A)$

$\rightarrow (\underbrace{A \cup B}_{\hookrightarrow A}) \wedge (\neg B \vee C) \wedge (\neg C) \wedge (\neg A) \wedge (\underbrace{\neg B}_{\hookleftarrow})$

$\rightarrow (A \cup B) \wedge (\neg B \vee C) \wedge (\neg C) \wedge \underline{(A)} \wedge (\neg B) \wedge \underline{(A)}$

$\rightarrow (A \cup B) \wedge (\neg B \vee C) \wedge (\neg C) \wedge (\neg A) \wedge (\neg B) \wedge (A) \wedge C)$

$\rightarrow$ Empty clause $\Rightarrow$ contradiction

3. First-order logic.

Use Constant Symbols and predicate Symbols.

Constant Symbols:
- Ninerva
- Pomona
- Horace
- Gilderoy
- Gryffindor
- Hufflepuff
- Ravenclaw
- Slytherin

Predicate Symbols:
- person
- House
- BelongsTo

Ex. Person(Ninerva) => Ninerva is a Person.
House(Gryffindor) => Gryffindor is a house
¬House(Ninerva) => Ninerva is not a house
BelongsTo(Ninerva, Gryffindor)

=> Minimize number of items needed.

→ Universal Quantification

$\forall x$. BelongsTo(x, Gryffindor) → ¬BelongsTo(x, Hufflepuff)
=> Anyone in Gryffindor is not in Hufflepuff.

→ Existential Quantification.

$\exists x$ House(x) ∧ BelongsTo(Ninerva, x)
=> Ninerva belongs to a house.

More general:

$\forall x$, Person(x) → ($\exists y$, House(y) ∧ BelongsTo(x, y))

Chapter 2 : Uncertainty

a. Probability

Possible world : ω $\quad$ Ω : all possible worlds.
Represent proba : $P(\omega)$

rules : • $\forall \omega, \quad 0 \leq P(\omega) \leq 1$
↓ impossible event ↳ certain event

• $\sum_{\omega \in \Omega} P(\omega) = 1$

Def : **Unconditional probability** : degree of belief in a proposit. in the absence of other any evidence.

Def **Conditional probability** : degree of belief in a proposit. given some evidence that has already been revealed.
⇒ Notation $P(a|b)$: "proba of a given b."

property : $P(a|b) = \frac{P(a \wedge b)}{P(b)}$ or $P(a \wedge b) = P(b) P(a|b)$

Def **Random variable** : a variable in proba theory with a domain of possible values it can take on

Ex : roll a dice $\quad$ Roll : $\{1, 2, 3, 4, 5, 6\}$
↳ random variable.

Def **Probability distribution** : take all values of a random variable and return the proba of each

Ex: Notation $P(Roll) = \langle \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6} \rangle$

⚠ This notation implies that we know the order of values of Roll.

Def Independance: the knowledge that one event occurs does not affect the proba of the other event

Consequence: $P(a \wedge b) = P(a) P(b|a)$

$P(a \wedge b) = P(a) P(b)$ → This a def for a being independent.

$P(a \wedge b) = P(b) P(a|b) = P(a) P(b|a)$

Bayes' Rule: $P(b|a) = \frac{P(a|b) P(b)}{P(a)}$

Def Joint probability: probability of a combination of two values from two random variable.

Def Joint probability distribution: all proba of all possible combination of two random variable.

rule: Negation: $P(\neg a) = 1 - P(a)$

rule: Inclusion - Exclusion: $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$

rule: Marginalization: $P(a) = P(a \wedge b) + P(a \wedge \neg b)$
b for random variable.

$P(X = x_i) = \sum_j P(X = x_i, Y = y_j)$

Marginalization is useful if I have joint proba.

rule: Conditioning: $P(a) = P(a|b)P(b) + P(a|\neg b)P(\neg b)$

Generalize:

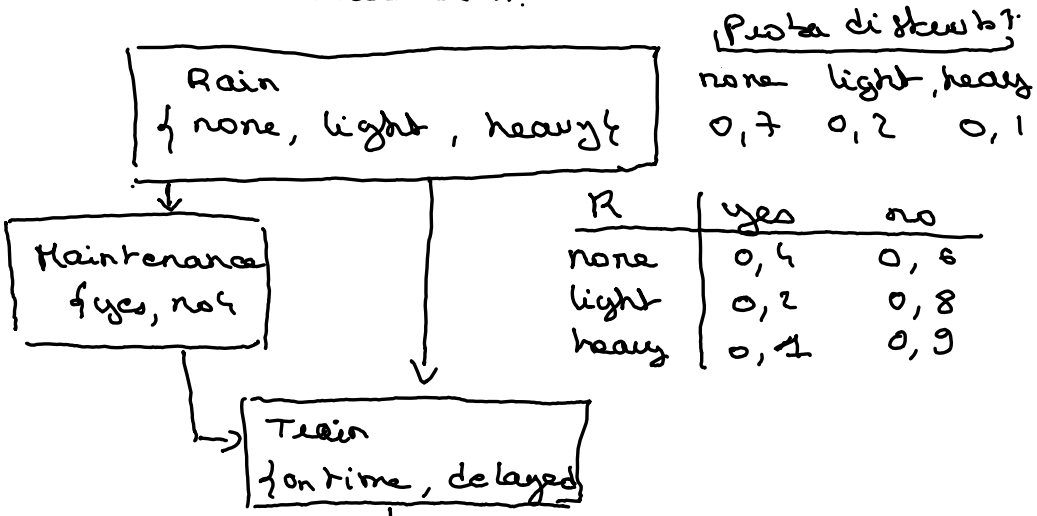
$$P(X = x_i) = \sum_j P(X = x_i | Y = y_j) P(Y = y_j)$$

Def **Bayesian Network**: data structure that represents the dependencies among random variables.

→ directed graph

- each node represents a random variable
- arrow from X to Y means X is a parent of Y
- each node X has probability distribution $P(X | \text{Parents}(X))$.

Ex:



R	M	onTim	delay
none	yes	0,8	0,2
none	no	0,9	0,1
light	yes	0,6	0,4
light	no	0,7	0,3
heavy	yes	0,4	0,6
heavy	no	0,5	0,5

T	attend	miss
on time	0,9	0,1
delayed	0,6	0,4

→ Proba distrib.

Rq: it is easier usually to create the dataset with dependency relationship from a node to another.

=> Computing joint probabilities with Bayes Rule.

Ex: $P(\text{light} \wedge \text{no} \wedge \text{delayed} \wedge \text{miss})$
$= P(\text{light})\, P(\text{no} \mid \text{light})\, P(\text{delayed} \mid \text{light} \wedge \text{no})\, P(\text{miss} \mid \text{delayed})$

b. Inference

Can we get Proba of new variable with what we have. . Same idea as knowledge chapter.

Inference Problem:

- Query X : variable for which to compute distribution
- Evidence : variables E : observed variables for event e
- Hidden variables Y : non-evidence, non-query variable.

=> Goal : calculate $P(X \mid e)$

Ex: Same as previous page.
$P(\text{appointment} \mid \text{light} \wedge \text{no})$.

Query : Appointment = A
Evidence : light, no = L, N
hidden : train. = T

$P(\text{Appointment} \mid \text{light} \wedge \text{no}) = \alpha\, P(\text{Appointment} \wedge \text{light} \wedge \text{no})$

iterate over all values of hidden variable

$P(A \wedge L \wedge N) = \alpha\, P(A \wedge L \wedge N \wedge \text{onTime}) + \alpha\, P(A \wedge L \wedge N \wedge \text{delayed})$

$P(X, e) = P(X \wedge e)$

formula : Inference by Enumeration

$$P(X|e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)$$

X : query variable
e : evidence
y : ranges over values of hidden variables
α : normalise the result.

Rq : There is library that already implement this function.
ex : pomegranate library
→ you have to describe all the node saying if it is conditional proba or not.
→ create Bayesian network
→ creat links between nodes.

⚠ Not particularly efficient.

c. Approximate Inference

→ Sampling : Create samples from possible value of each Node.
→ Then when I search the proba of a value, I search the number of samples that has this value. Same if I look for a combination. If I want conditional, I only search amongst samples where there is the known value.

⚠ only works if you have a great amount of sample.

⚠ less precise but faster.
(not 100 % precise but if enough samples, good enough.

Proba : $P(a) = \frac{\text{Number Samples with a}}{\text{Number total Samples}}$

$P(a \wedge b) = \frac{\text{Number Samples with a and b}}{\text{Number total Samples}}$

$P(a|b) = \frac{\text{Number Samples with a and b}}{\text{Number Samples with b}}$

d. Likelihood Weighting.

Principle :

- Start by fixing the values for evidence variables
- Sample the non-evidence variables using conditional probabilities in the Bayesian Network
- Weight each sample by its likelihood : the probability of all the evidence.

Ex : Same as previous.
We want to calculate $P(R = light \mid T = on\ time)$?

→ Fix evidence variable
  → R = ?
    M = ?
    T = onTime
    A = ?

→ Sample non evidence : R = light
M = yes
T = onTime
A = miss

→ After getting a sample, weight it
here $P(\text{OnTime} \mid \text{light} \wedge \text{yes}) = 0,6$.

⇒ Repeat procedure to have lot of sample.

⇒ Now same principle of sampling to find proba.
↳ Approximate.

e. Prediction on Time.

Def **Markov assumption**: The assumption that the current state depends on only a finite fixed number of previous State

Def **Markov chain** : a sequence of random variable where the distribution of each variable follows the Markov assumption.

Ex: predict if sunny or raining
S R

X_t : weather at time t (today)
X_{t+i} : weather at time $t+i$

Today (X_t) \ Tomorrow (X_{t+1})	S	R
S	0,8	0,2
R	0,3	0,7

⇒ Transition model.
→ create Markov Chain
$S \rightarrow S \rightarrow R \rightarrow R \rightarrow R \ \ldots$
$X_0 \quad X_1 \quad X_2 \quad X_3 \quad X_4$

Application : Find Hidden State from observat.

ex:	Hidden State	observat.
	robot's position words spoken user engagement weather (S or R)	robot's sensor data audio waveforms website or app analytics umbrella (U)

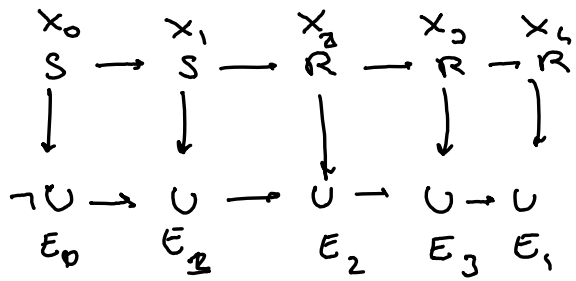
Def **Hidden Markov Model** : a Markov model for a system with hidden states that generate some observed event.

Ex :

	observation (E_t)	
State (X_t)	U	¬U
S	0,2	0,8
R	0,9	0,1

Def **Sensor Markov Assumption** : the assumption that the evidence variable depends only the corresponding state.

Ex : Chain



Ex of task done:

Task	Definition
filtering	given observat- from start until now calculate distribut- for current state
prediction	given observat- from start until now calculate distribut- for future state
smoothing	given observat- from start until now calculate distribut- for past state
most likely explanation	given observat- from start until now calculate most likely sequences of state.

Chapter 4 : optimization.

Def **optimization**: choosing the best option from a set of options

a. local search.

Def **local search** : search algo that maintain a single node and searches by moving to a neighbor node.

Def **state-space landscape**: has graph where each has represents a possible state and its value.

Then we can search:
- → global max / min
- → local min

⇒ for that we define an objective funct (for max)
cost function (for min)

⇒ **Hill climbing Algorithm principle**.

```
function Hill-Climbing (problem):
    current = initial state of problem.
    repeat:
        neighbor = highest valued neighbor of current
        if neighbor not better than current
            return current
        current neighbor
```

⚠ It will not always give the optimal solut. (global) but a local min / max

Hill climbing variants:

variant	definition
steepest-ascent	choose the highest-valued neighbor
stochastic	choose randomly from higher-valued neighbors
first-choice	choose the first higher-valued neighbor
random-start	conduct hill climbing multiple times
local beam search	choose the k highest-valued neighbors

b. Simulated annealing

Principle

- Early on, higher "temperature": more likely to accept neighbors that are worse than current state.
- Later on, lower "temperature": less likely to accept neighbors that are worse than current state.

Pseudo code

```
function simulated-annealing(problem, max):
    current = initial state of problem
    for t = 1 to max
        T = temperature(t)
        neighbor = random neighbor of current
        ΔE = how much better neighbor is than current
        if ΔE > 0: (neighbor is better)
            current = neighbor
        with probability exp(ΔE / T) set current = neighbor
    return current
```

Ex: Traveling salesman Pb:
visit cities and get back home by minimizing the distance.

C. Linear Programming.

Problem:

- Minimize a cost function $c_1x_1 + c_2x_2 + \cdots + c_nx_n$
- with constraints of form $a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$ or of form $a_1x_1 + \cdots + a_nx_n = b$
- with bounds for each variables $l_i \leq x_i \leq u_i$

Ex: Two machines x_1 and x_2 costs 50 \$/h and 80 \$/h to run. goal is to minimise cost

- x_1 requires 5 units of labor per hour. x_2 requires 2 units of labor per hour. Total of 20 units of labour to spend.

- x_1 produces 10 units of output/h. x_2 produces 12 units of output per hour. company needs 90 units of output.

Cost function: $50x_1 + 80x_2$ x_i: num hour.

constraints: $5x_1 + 2x_2 \leq 20$
$10x_1 + 12x_2 \geq 90 \Rightarrow (-10x_1) + (-12x_2) \leq -90$ (we need $\leq$ or $=$ not $\geq$)

Algo possible: Simplex

d. Constraint satisfaction.

=> Constraint satisfaction Problem

- Set of variables $\{X_1, \dots, X_n\}$
- Set of domains for each variable $\{D_1, D_2 \dots, D_N$
- Set of constraint C

Ex: Sudoku.

Variables: empty cells.

Domains: values between 1 and 9

Constraints: cells ≠ in row, column and square.

Def **hard constraints**: constraints that must be satisfied in a correct solution.

Def **Soft constraints**: constraints that express some notion of which solutions are preferred over others.

Def **unary constraint**: single variable not equal to a given value (ex: A ≠ Monday)

Def **binary constraint**: constraint involving two variables (ex: A ≠ B)

Def **node consistency**: when all the values in a variable's domain satisfy the variable's unary constraints

Ex: A ——— B

{Mon, Tue, Wed} {Mon, Tue, Wed}

Constraint: {A ≠ Mon, B ≠ Tue, B ≠ Mon, A ≠ B}

=> A ——— B node consistency. pb now.

{Tue, Wed} {Wed}

Def arc consistency: when all the values in a variable's domain satisfy the variable's binary constraints.
To make X arc-consistent with respect to Y, remove elements from X's domain until every choice for X has a possible choice for Y.

Ex: A ⟶ B
{Tue, Wed} {Wed}

Same constraints

⇒ A — B arc consistency pb.
{Tue} {Wed}

Arc consistency pseudo-code.

```
function revise (csp, X, Y):
    revised = false (if no change in X domain, True if change)
    for x in X.domain:
        if no y in Y.domain satisfies constraint for (X,Y):
            remove x from X.domain
            revised = True.
    return revised
```

⚠ This function makes one variable arc-consistency not all.

To make an entire pb arc consistant, we use the following function:

```
function AC-3(csp):          csp: constraint satisfact. pb
    queue = all arcs in csp
    while queue non empty:
        (X,Y) = Dequeue(queue)
        if revise(csp, X, Y):
            if size.of X.domain == 0:
                return false   (no way to solve pb)
            for each Z in X.neighbors - {Y}:
                Enqueue(queue, (Z,X)).
    return True.
```

This algo reduce the domain for each variable but do not always solve the pb. Some variable can still have multiple values.

=> CSP as search problem:

- Initial state: empty assignment (no variable)
- actions: add a {variable = value} to assignment
- transition model: shows how adding an assignment changes the assignment
- goal test: check if all variables assigned and constraints all satisfied
- path cost function: all paths have same cost

He is a exemple of an algo pseudo code.

Pseudo-code.

```
function Backtrack (assignment, csp):
    if assignment complete:
        return assignment.
    var = Select-Unassigned-Var (assignment, csp)
    for value in Domain-Values (var, assignment, csp):
        if value consistent with assignment:
            add {var = value} to assignment
            result = Backtrack (assignment, csp).
            if result ≠ failure: return result.
        remove {var = value} from assignment
    return failure. (no value to solve issue)
```

Rq: constraint python library has all this function already implemented.

e. Inference.

Add arc consistency to backtrack

Def maintaining arc-consistency: algorithm for enforcing arc-consistency every-time we make a new assignment

Every time we make a new assignment to X, calls AC-3, starting with a queue of all arcs (Y, X) where Y is a neighbor of X

New pseudo code:

Pseudo - code.

```
function BacktTrack(assignment, csp)
    if assignment is complete: return assignment
    var = select-Unassigned-Value(assignment, csp)
    for value in Domain-Values(var, assignment, csp):
        if value consistant with assignment:
            add {var = value} to assignment.
            inferences = Inference(assignment, csp)
            if inferences ≠ failure: add inferences to assignment
            result = Backtrack(assignment, csp)
            if result ≠ failure: return result.
        remove {var = value} and inferences from assignment
    return failure.
```

1. Select-Unassigned-Value.

To be more efficient, it is important to select well this variable.

Def **minimum remaining values (MRV)** heuristic: select the variable that has the smaller domain.

Def **degree heuristic**: select the variable that has the highest degree.

Def **node degree**: number of node that are attached to a node. number of node that are constraint by this node

9. Domain values

Choosing well the order of the value in the variable selected is also a way to go faster.

Def **least-constraining value heuristic**: return variables values in order by number of choices that are ruled out of neighboring values.

- Try least-constraining value first

Chapter 5: Learning

I. Supervised Learning.

Def **Supervised learning** : given a data set of input-output pairs, learn a function to map inputs to outputs

a. Classification

Def **Classification** : supervised learning task of learning a function mapping an input point to a discrete category

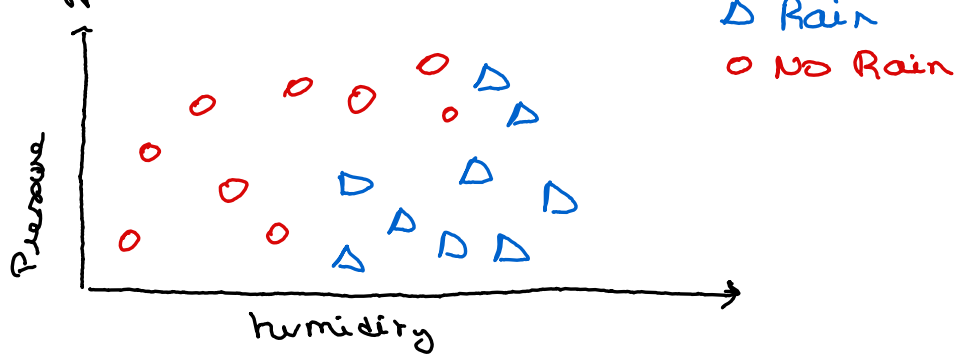
Ex: predict weather. Supervised.

Data :

Date	Humidity	Pressure	Rain
Jan 1	93	999,7	Yes
Jan 2	49	1015,5	No
Jan 3	79	1031,5	No
Jan 4	65	984,9	Yes
Jan 5	90	975,2	Yes

f(humidity, pressure) = Rain

↳ approximate with h.



Neighbor approach

Def **nearest-neighbor classification**: algorithm that, given an input, chooses the class of the nearest data point to that input.

⚠ Tricky if you are close to the frontier. Where nearest point can be not good solution.

Def **k-nearest-neighbor classification**: algorithm that, given an input, chooses the most common class out of the k nearest data points to that input.

⚠ can be slow.

One solution is to draw a linear regression to be faster.

linear regression approach

Here: x_1 = humidity, x_2 = pressure.

$h(x_1, x_2)$ = Rain if $w_0 + w_1 x_1 + w_2 x_2 \geq 0$
No Rain otherwise.

weight vector w : (w_0, w_1, w_2)
input vector x : $(1, x_1, x_2)$

goal: find weight vector to do prediction.

Def **dot product**: $x \cdot w = w_0 + w_1 x_1 + w_2 x_2$

$$\Rightarrow \quad h_w(x) = \begin{cases} 1 & \text{if } w \cdot x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Def perceptron learning rule: given data point (x, y) update each weight according to:

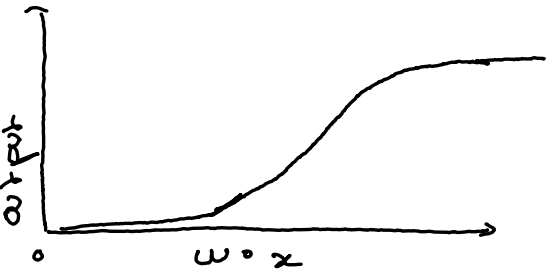
$$w_i = w_i + \alpha(y - h_w(x))\, x_i$$

y: actual value; $h_w(x)$: estimate; α: learning rate

Def hard threshold: does not allow uncertainty give 1 or 0 always

$$\begin{cases} 1 \text{ if } w \cdot x \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

Def soft threshold: can give every value between 1 and 0. give info about how confident we are



$$g(x) = \frac{e^x}{e^x + 1}$$

5. Support vector machines.

Def maximum margin separator: boundary that maximize the distance between any of the data points.

The Support vector machines (SVC) will find this one.
=> good with data that are not linearly separable.

Def **regression** : supervised learning task of learning a function mapping an input point to a continuous value.

Ex : Impact of advertising

f(advertising) = sales

↓ money spend ↑ money make

⇒ linear regression approach possible

c. Evaluating Hypotheses.

Def **loss function** : function that expresses how poorly an hypothesis perform

⇒ For classification (discrete)

Def **0-1 loss function** :

$$L(\text{actual}, \text{predicted}) = \begin{cases} 0 \text{ if actual} = \text{predicted} \\ 1 \text{ otherwise} \end{cases}$$

⇒ For regression (continue)

Def **L_1 loss function**

$$L(\text{actual}, \text{predicted}) = |\text{actual} - \text{predicted}|$$

Def **L_2 loss function** :

$$L(\text{actual}, \text{predicted}) = (\text{actual} - \text{predicted})^2$$

→ penalize more value that are too away.

Def **overfitting** : a model that fits too closely to a particular data set and therefore may fail to generalise to future data.

If we only want to reduce loss, we might have overfitting.
To avoid that, the best is to minimize:
cost(h) = loss(h) + λcomplexity(h)

- simpler sol can be better.
- if λ big : penalize complexity
- if λ small : penalize loss

Def regularization : penalizing hypotheses that are more complex to favor simpler, more general hypotheses

Def holdout cross-validation : splitting data into a training set and a test set, such that learning happens on the training set and is evaluated on the test set.

Def k-fold cross-validation : splitting data into k sets, and experimenting k times, using each set as a test set once and using remaining data as training set

d. Python library.

=> Scikit-Learn : - linear models - fit
- neighbors - predict
- SVC. ... - split data

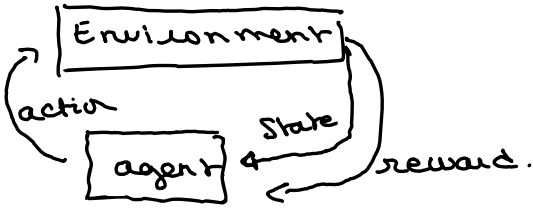
=> CSV, pandas : read csv file.

=> random : randomize data (shuffle, select randomly...)

=> cf bank notes 1.py

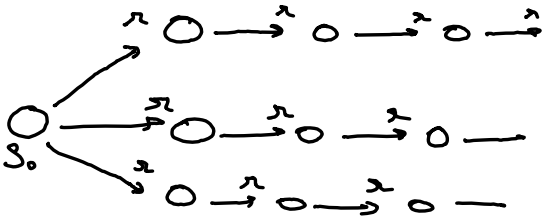
II. Reinforcement learning

Def reinforcement learning: given a set of rewards or punishments, Learn what actions to take in the future.



Def Markov decision process: model for decision-making, representing states, actions and their rewards.

Rappel: Markov chain chap. 3.



Markov model:
- Set of state S
- Set of actions(S)
- Transition model $P(S' \mid S \wedge a)$
- Reward function $R(S, a, S')$

Def Q-Learning: method for learning a function $Q(S, a)$, estimate of the value of performing action a in state S.
(How much reward will I get)

Principle

- Start with $Q(s,a) = 0$ for all s, a.
- When we taken an action and receive a reward:
  - estimate the value of $Q(s,a)$ based on current reward and expected future rewards
  - update $Q(s,a)$ to take into account old estimate as well as our new estimate.

Pseudo-code.

- Start with $Q(s,a) = 0$ for all s, a
- Every time we take an action a in a state s and observe a reward r, we update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left((r + \gamma \max_{a'} Q(s',a')) - Q(s,a)\right)$$

α: Learning rate ⇒ importance of new value compare to old (if $\alpha = 1$ only new value count)

$r + \gamma \max_{a'} Q(s',a')$: future rewards estimate; new value estimate

$Q(s,a)$: old value estimate

Def Greedy Decision making: when in state s, choose action a with highest $Q(s,a)$

⚠ Greedy can lead to solution that are not optimize.

Exploration vs Exploitation
↳ explore ↳ use rewards to make decision

Def ε-greedy:

- Set ε equal to how often we want to move randomly
- with probability $1 - \varepsilon$, choose estimate best move.
- with probability ε, choose a random move.

you can decrease ϵ over time.

Ex: play game. nim (strategy : AI to learn how to win)

Def function approximation: approximating $Q(s,a)$, often by a function combining various features, rather than storing one value for every state-action pair

III. Unsupervised learning

Def unsupervised learning: given input data without any additional feedback, learn patterns.

Def clustering: organizing a set of objects into groups in such a way that similar objects tend to be in the same group.

Ex application:
- genetic research
- Image segmentation
- Market research
- Medical imaging
- Social network analysis

Def k-means clustering: algorithm for clustering data based on repeatedly assigning points to clusters and updating those clusters' centers.

Chapter 6 : Neural Networks.

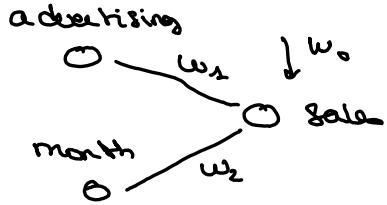
Principle :

- Neurons are connected to and receive electrical signals from other neurons
- Neurons process input signal and can be activated.

Def artificial neural network : mathematical model for learning inspired by biological neural networks.

- Model mathematical function from inputs to outputs based on the structure and parameters of the networks
- allows for learning the network's parameters based on data.

Ex :



a. One layer

Def gradient descent : algorithm for minimizing loss when training neural network.

Principle :

- Start with a random choice of weights
- Repeat :
  - Calculate the gradient based on all data points : direction that lead to decreasing loss
  - update weights according to the gradient

⚠ calculate on all data points is expensive.

→ alternative : • Stochastic gradient descent.

⇒ calculate the gradient based on one data points chose randomly

(faster but less accurate)

• Mini batch gradient descent

⇒ calculate the gradient on a mini batch of points.

If multiple output we can see that as several neural Networks and we can train them separately.

limitation of linear combination:

⇒ only for linear separat

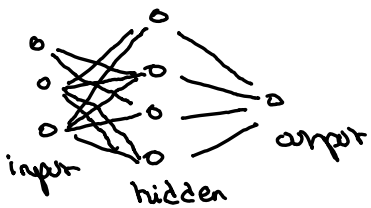
Def Perceptron: only capable of learning linearly separable decision boundary.

b. multilayer.

Def multilayer neural network: artificial neural network with an input layer, an output layer and at least one hidden layer.

Not like this:

but



⇒ ability to model complex situation by adding layer

Def **backpropagation**: algorithm for training neural networks with hidden layers

Principle:

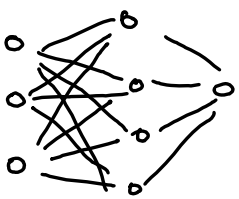
- start with a random choice of weights:
- repeat:
  - calculate error for output layer
  - for each layer, starting with output layer and moving inward towards earliest hidden layer
    - propagate error back one layer
    - update weights.

Def **deep neural networks**: neural networks with multiple hidden layers

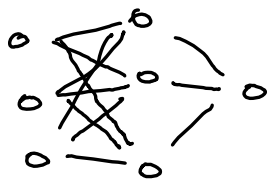
c. overfitting

Def **dropout**: temporarily removing units, selected at random, from a neural network to prevent over-reliance on certain units.

Ex:



sometime become:



d. library.

=> **Tensorflow** (develop by google)

playground.Tensorflow.org : to understand principle

keras: API.

e. Computer vision

Def **Computer vision**: computational methods for analyzing and understanding digital images

Def **image convolution**: applying a filter that adds each pixel value of an image to its neighbors, weighted according to a kernel matrix

Ex:

kernel : $\begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix}$

image :

10	20	30	40
10	20	30	40
20	30	40	50
20	30	40	50

step 1 × kernel

First image [3×3] × kernel (multiply each value by its corresponding in kernel.

→ $0 \times 10 - 20 + 30 \times 0 + \cdots = 10$

→ Slide kernel :

10	20
40	50

Def **pooling**: reducing the size of an input by sampling from regions in the input.

Def **max-pooling** : pooling by choosing the maximum value in each region

Ex : image :

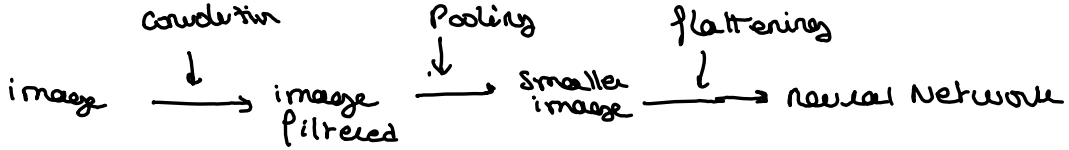
30	40	80	90
20	50	100	110
0	10	20	30
10	20	40	30

2×2 pooling →

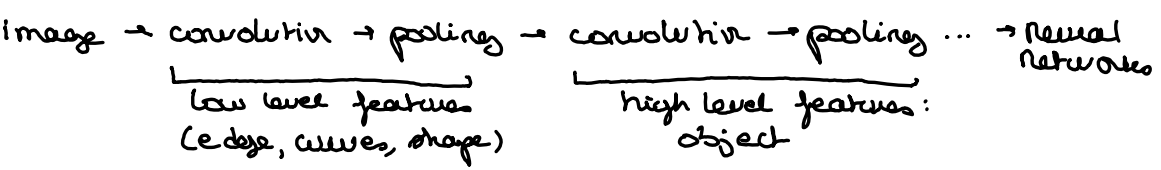
50	110
20	40

Def **convolutional neural network (CNN)**: neural networks that use convolution, usually for analysing images

=> CNN can learn what filter use.

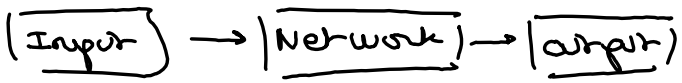


You can use it several time



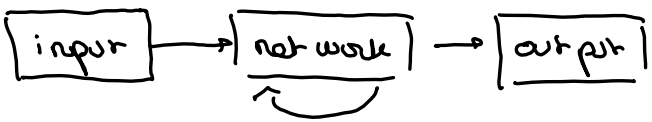
tf.keras.layers.Dense(activation="softmax")
↳ proba distribution

Representation:



Def **feed-forward neural network**: neural network that has connection only in one direct.

Def **recurrente neural network**: neural network that generate output that feeds back into its own inputs



Input
network
output
network
output
network
output

Chapter 7 : Language.

I. Natural language processing

- automatic summarization
- Info extraction
- language identification
- machine translation ...
- named entity recognition
- speech recognition
- text classification
- word sens .disambiguation. ...

⇒ need to focus on
  - → syntax (grammatically correct ...)
  - → semantic (order of words ..)

Def **formal grammar** : a system of rules for generating sentences in a language.

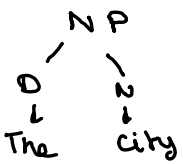
⇒ **context free meaning**

She	saw	the	city
N	V	D	N

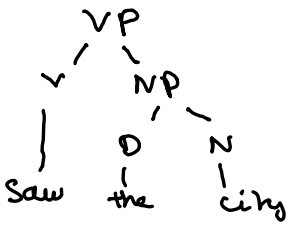
N : noun
V : verb
D : determinant
P : preposition

a word = terminal symbol
type : non terminal symbol

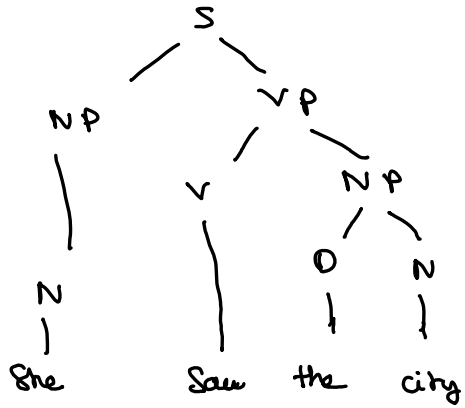
NP → N | D N (noun phrase) (noun or determinant + noun)



VP → V | V NP (verb phrase)



S → NP VP (sentence)



If a sentence is not grammatically correct it will not be possible to create such a tree.

good if not that much rules.

⇒ Python library : nltk.

Def n-gram : a contiguous sequence of n items from a sample of text.

Def character n-gram : a contiguous sequence of n characters from a sample of text

Def word n-gram : a contiguous sequence of n words from a sample of text

Def unigram : a contiguous sequence of 1 item from a sample of text.

Def bigrams : a contiguous sequence of 2 item from a sample of text.

Def trigrams : a contiguous sequence of 3 item from a sample of text.

can be extract n gram as our AI might have seen them already in other text.

Def **tokenization** : the task of splitting a sequence of characters into pieces (tokens)

Def **word tokenization** : the task of splitting a sequence of characters into words.

=> any time see a space create an item.

⚠ pb ponctuation that still there.

II Naïve model.

Once we know the most common n-gram, we can deduce from a word from other.

Ex: if " it was a " is the most common trigram, if we have " it was " we can imagine having " a " after.

=> python library : markovify + nltk.

III. Text categorization

Ex : mail : spam or not
sentiment analysing.

Def **bag of word model** : model that represents text as an unordered collection of words.

=> **Naive Bayes approach**.

p : positive, n : negative.

Ex : Text : "my grandson loved it"

$c = [\text{"my"}, \text{"grandson"}, \text{"loved"}, \text{"it"}] = c_1, c_2, c_3, c_4$

$$P(p|c) = \frac{P(c|p)\,P(p)}{P(c)} \sim P(c|p)\,P(p) = P(c \wedge p)$$

↑ Bayes' rule. ↑ proportionnal to

$$P(c \wedge p) = P(c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge p)$$

$$= P(p)\ P(c_1|p)\ P(c_2|p)\ P(c_3|p)\ P(c_4|p)$$

We assume that words are independent if we know it is positive message (Naive part)

$$P(p) = \frac{\text{num positive sample}}{\text{num sample.}}$$

$$P(c_1|p) = \frac{\text{num positive sample with } c_1}{\text{num positive sample.}}$$

p	n
0,49	0,51

$\Rightarrow P(c \wedge p) = 0,000\ 14112$

$P(c \wedge n) = 0,\ 0000\ 6528$

	p	n
c_1	0,3	0,2
c_2	0,01	0,02
c_3	0,32	0,08
c_4	0,3	0,4

$\Rightarrow$ Treat value as probability distribution and normalized them

$\Rightarrow P(p|c) = 0.6837$

$P(n|c) = 0,3169$

$\Rightarrow$ we can guess the message is positive

⚠ if 0 is in the data set : ex if $p(c_2|p) = 0$. (c_2 never happen) $\Rightarrow P(c \wedge p) = 0$ (problem because c_2 never happen in positive sentence).

Def **additive smoothing** : adding a value α to each value in our distribution to smooth the data.

Def **laplace smoothing** : adding 1 to each value in our distribution : pretending we've seen each value one more time that we actually have.

$\Rightarrow$ avoid multiply by 0.

=> python library : nltk.

IV. Information retrieval.

Def **information retrieval** : the task of finding relevant documents in response to a user query

Def **topic modeling** : models for discovering the topics for a set of documents.

Def **term frequency** : the number of times a term appears in a document.

Def **function words** : words that have little meaning on their own, but are used to grammatically connect other words.

Def **content words** : words that carry meaning independently.

=> usually when we look a document we are interested in the term frequency of content word.
=> When I do Tf, I ignore function words

Def **inverse document frequency** : measure of how common or rare a word is accross documents

=> give better idea of specific topic of a document compare to others. (how rare is a word in the documents)

=> inverse document frequency (word) = $\log\left(\frac{\text{Total documents}}{\text{Num documents containing (word)}}\right)$

Def **Tf-idf** : ranking of what words are important in a document by multiplying term frequency (Tf) by inverse document frequency (idf)

V. Semantico.

Def **Information exctraction** : the task of exctracting knowledge from documents.

=> give AI template to search.
ex : when {company} was founded in {year}
→ will Learn name of company and year of creation
=> ⚠ needs to have template → long to write.

→ If we give data instead of template, AI can search and Learn the template by looking at sentence that relate data. Then use it for other data.

⚠ only works if we find template.

=> **word net** : dictionnary that give all definition of a word and its relation with others.
⚠ Issue with word that change

a. word representation.

Representing word with number to use them in a Neural Network for exemple.

Ex : "He wrote a book"
He : [1, 0, 0, 0], wrote : [0, 1, 0, 0] a : [0, 0, 1, 0] book: [0, 0, 0, 1]

Def **one-hot representation** : representation of meaning as a vector with a single 1, and with other values as 0.

⚠ if dictionnary have 10^5 words, each vecta is 10^5 big.

⚠ each word are disconnected :
ex: work and author can be totally ≠ (nothing to do)
(we want vector that are similar)

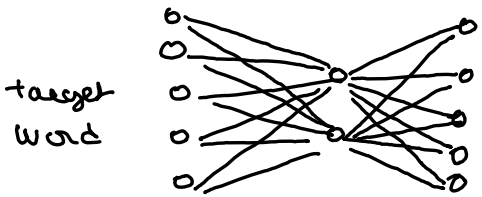
Def **distribution representation** : representation of meaning distributed across multiple values.

Ex : he : [-0.34, -0,08, 0.02, ...]
work : [-0.24, 0.40, 0.00 ....]
⋮

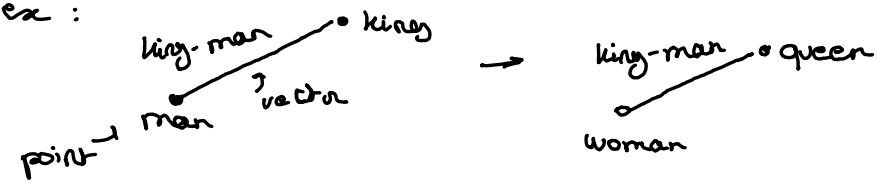
we can hope that similar word have similar vector.

Def **word 2 vec** : model for generating word vectors

Def **skip-gram architecture** : neural network architecture for predicting context words given a target word



if you have vector you can do some calculation :
ex :



⇒ can help to find new word with other.
queen = woman + king - man