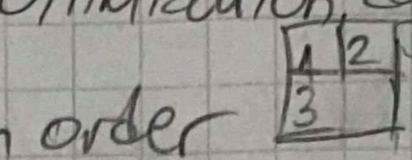
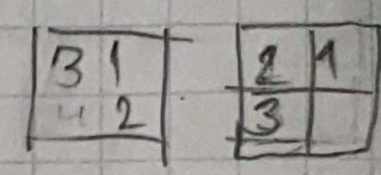
Introduction to AI with Python

- Search, Knowledge, Uncertainty, Optimization, Learning, Neural Networks, Lang

Search Problem: numbers line up in order  : Agent → Solver (AI or person)

Agent - entity that perceives its environment and acts upon that envir.

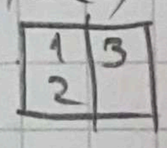
Driving directions → Car

State - a configuration of the agent and its environment. 

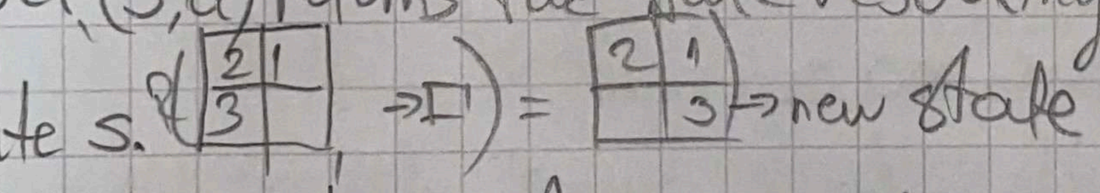
Initial State - the state in which the agent begins.

Actions - choices that can be made in any given state.

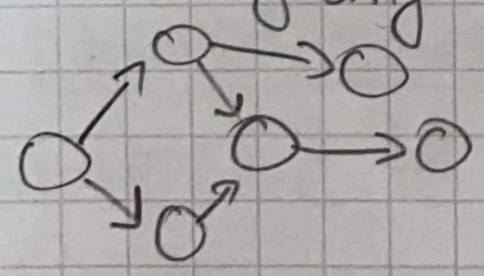
f: Actions(s) returns the set of actions that can be executed in state s

Actions  ↑↓→←

Transition model - a description of what state results from performing any applicable action in any state

f: Result(s,a) returns the state resulting from performing action a in state s. 

State space - the set of all states reachable (dostupan) from the initial state by any sequence of actions.

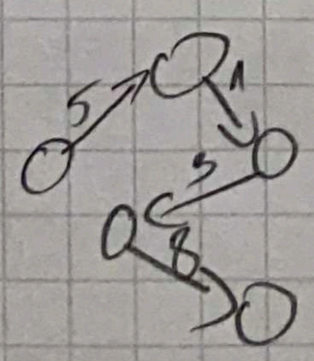


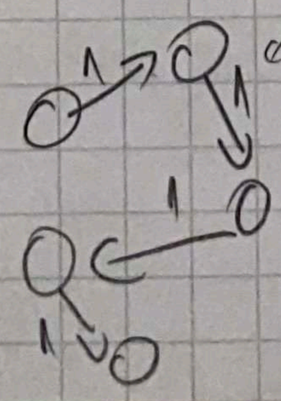
Node - represent one of the states inside of our problem

Arrows - represent action that we can take in any particular state

Goal test - way to determine whether a given state is a goal state.

Path Cost - numerical cost associated with a given path





(puzzle)

Search Problems: initial state, actions, transition model, goal test, path cost function

Solution - a sequence of actions that leads from the initial state to a goal state.

Optimal solution - a solution that has lowest path cost among all solutions.

Node - a data structure that keeps track of:

- a state
- a parent (node that generated this node)
- an action (action applied to parent to get node)
- a path cost (from initial state to node)

Approach

Frontier (granica) - DS that store all available options.

- Start with a frontier that contains the initial state (only we know)

Repeat:

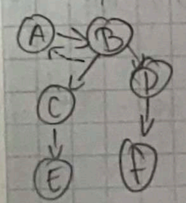
- If the frontier is empty, then no solution.

Remove a node from the frontier.

- If node contains goal state, return the solution

Expand node, add resulting nodes to the frontier.

Find path from A to E



frontier: A -> Start ... B ... C D ... E D ... D

Keep track of what we explored to avoid loop.

Revised Approach

-||-

- Start with an empty explored set

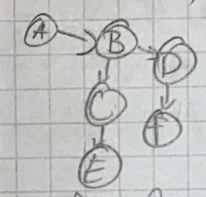
Repeat:

-||- 1, 2, 3

- Add the node to the explored set.

-||- If they aren't already in the frontier or the explored set

Stack (Lifo) data type



frontier: A ... B ... C D F E

Explored set: A B D F C

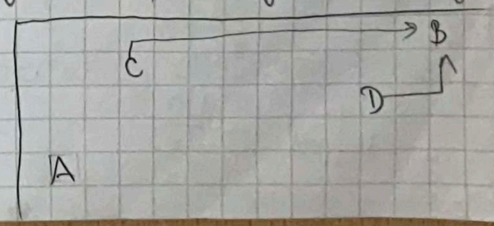
DFS

BFS - Breadth-first search Queue (fifo)

2 Types of search alg:

- Uninformed search - search strategy that uses no problem-specific knowledge. (DFS, BFS)
- Informed search - search strategy that uses problem-specific knowledge to find solutions more efficiently (Greedy best-first search)
- Greedy best-first search - search alg that expands the node that is closest to the goal, as estimated by a heuristic function h(n)

Manhattan distance (Heuristic f) (istraživačka) - How many squares, vertically and horizontally, takes to get to the goal (ignoring walls)



A* search (A star) - search alg that expands node with lowest value of $g(n)+h(n)$.

$g(n)$ - cost to reach node $h(n)$ = estimated cost to goal

- A star going to make a choice based on the sum of how many steps it took to get to my current position and then how far I estimate I am from the goal

Optimal if:

- $h(n)$ is admissible (never overestimates the true cost) (dopustiv)
- $h(n)$ is consistent (for every node n and successor n' with step cost c, $h(n) \leq h(n')+c$)

- Minimax Alg (Tic-Toc)

Max (x) aims to maximize score. Min (O) aims to minimize score.

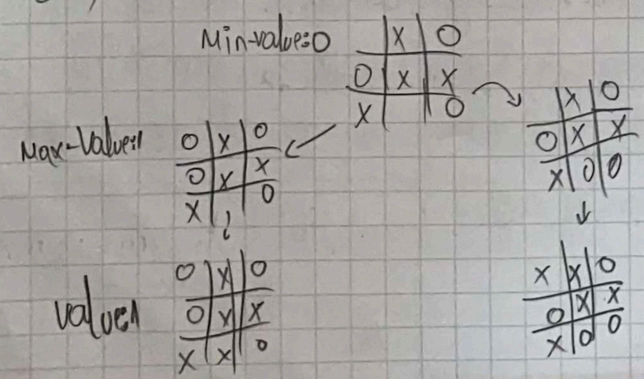
Game:

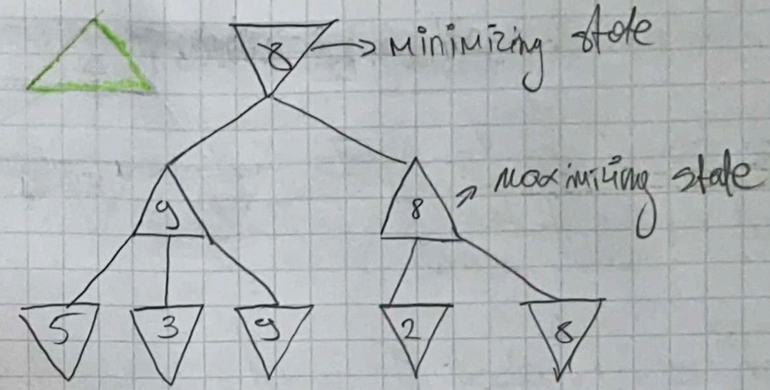
- S_0: initial state
- Player(s): returns which player to move in state s.

Actions(s): returns legal moves in state s

Result(s,a): returns state after action a taken in state s

- Terminal(s): checks if state s is a terminal state s
- Utility(s): final numerical value for terminal state s, win-1 L-1 D-0





- Given state s:
  - Max picks action a in Actions(s) that produces highest value of Min-Value(Result(s,a))
  - Min picks action a 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
```

- Alpha-Beta Pruning (orezivanje) - 2 values, best/worst you can do so far

Depth limited Minimax -

evaluation f-n that estimates the expected utility of the game from a given state

Knowledge

Knowledge-based agents - agents that reason by operating on internal representations of knowledge

Logic

Sentence - an assertion about the world in a knowledge representation language

Model (tvrdnja) - assignment of a truth value to every propositional symbol ("a possible world"). P: It is raining T Q: It is a Monday F

Knowledge base - a set of sentences known by a KB agent

Entailment $\alpha \models \beta$ - In every model in which sentence α is true, sentence β is also true

Inference (zaključivanje) - the process of deriving new sentences from old ones

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

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

Inference algorithm - Does $KB \models \alpha$?

Model checking - enumerate all possible models, and see if there is true

To determin 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 entail α

P: It is a Tuesday. Q: It is raining

R: Harry will go for a run

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

P	Q	R	KB
⊥	⊥	⊥	⊤
⊥	⊥	⊤	⊤
⊥	⊤	⊥	⊤
⊥	⊤	⊤	⊤
⊤	⊥	⊥	⊤
⊤	⊥	⊤	⊤
⊤	⊤	⊥	⊥
⊤	⊤	⊤	⊤

Knowledge Engineering - Figure out how to distill problem down into knowledge that is representable by a computer.

Clue: Propositional Symbols

mustard	ballroom	knife
plum	kitchen	revolver
scarlet	library	wrench

(mustard ∨ plum ∨ scarlet)

(ballroom ∨ kitchen ∨ library)

(knife ∨ revolver ∨ wrench)

¬plum ¬mustard ∨ ¬library ∨ ¬revolver

Logic puzzles - (Pomona Slytherin ⇒ ¬Pomona Hufflepuff)....

- Mastermind

Interence Rules - rules that apply to take knowledge that already exists and translate it into new forms of knowledge

Modus Ponens, And Elimination, Double Negation Elimination, Implication Elimination

Biconditional Elim, De Morgan's Law, Distributive Property

Theorem Proving

initial state: Starting knowledge base

actions: inference rules

transition model: new knowledge base after inference

goal test: check statement we're trying to prove

path cost function: number of steps in proof

Interence by Resolution: $P \vee Q$ $\neg P \vee R$ $Q \vee R$

To determine if $KB \models \alpha$:

- Check if $(KB \wedge \neg\alpha)$ is a contradiction?

If so, then $KB \models \alpha$

Otherwise, no entailment

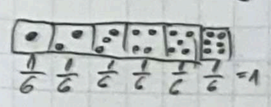
DS1 PR

Uncertainty 4

Probability

Possible worlds ω (dice has 6),

$P(\omega)$ - Probability of world, $0 \le P(\omega) \le 1$, $\sum_{\omega \in \Omega} P(\omega) = 1$ (all possible worlds)



$\frac{1}{6}\ \frac{1}{6}\ \frac{1}{6}\ \frac{1}{6}\ \frac{1}{6}\ \frac{1}{6} = 1$ $\quad P(\boxed{\cdot}) = \frac{1}{6}$

2 dices $P(\text{sum to } 12) = \frac{1}{36}$, $P(\text{sum to } 7) = \frac{6}{36} = \frac{1}{6}$

- Unconditional Probability - degree of belief in a proposition in the absence of any other evidence. (dice/dices)

Conditional Probability - degree of belief in a proposition given some evidence that has already been revealed

$P(a|b)$ - Probability of a given b, a is true given b we know true (evidence)

$P(\text{rain today} | \text{rain yesterday})$, $P(\text{rate change} | \text{traffic conditions})$, $P(\text{disease} | \text{test})$

$P(a|b) = \frac{P(a \wedge b)}{P(b)}$ → all way both can be true; → only world where b is true

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

$P(\text{sum } 12 | \boxed{\cdot\cdot}) = \frac{\frac{1}{36}}{\frac{1}{6}} = \frac{1}{6}$

$P(\boxed{\cdot\cdot}) = \frac{1}{6}$, $P(\text{sum } 12 \wedge \boxed{\cdot\cdot}) = \frac{1}{36}$ one in all

Random variable - a variable in probability theory with a domain of possible values it can take on. ex: Roll $\{1,2,3,4,5,6\}$

Weather $\{$sun, cloud, rain, wind, snow$\}$, Trafic $\{$none, light, heavy$\}$

Probability distribution - Prob for each of the possible values in its domain.

$P(\text{Flight} = \text{on time}) = 0{,}6$ (60%) $\quad P(\text{Flight}) = \langle 0{,}6, 0{,}3, 0{,}1 \rangle$ - representation using vector

$P(\text{Flight} = \text{delayed}) = 0{,}3$ (30%)

$P(\text{Flight} = \text{cancelled}) = 0{,}1$ (10%)

Independence - the knowledge that one event occurs does not affect the probability of the other event. $P(a \wedge b) = P(a)P(b)$

$P(RD\,RD) \neq P(RD)P(RD)$

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

Given clouds in the morning, what's the probability of rain in the afternoon?

- 80% of rainy afternoons start with cloudy mornings
- 40% of days have cloudy mornings
- 10% of days have rainy afternoons.

$P(\text{rain}|\text{clouds}) = \frac{P(\text{clouds}|\text{rain})P(\text{rain})}{P(\text{clouds})} = \frac{(0.8)\cdot(0.1)}{0{,}4} = 0{,}2$

Knowing $P(\text{cloudy morning} | \text{rainy afternoon})$ (visible effect, unknown cause) we can calc $P(\text{rainy afternoon} | \text{cloudy morn})$ (U.C., V.E)

Joint Probability - considering the likelihood of multiple different events simultaneously

AM		PM	
C = cloud	C = ¬cloud	R = rain	R = ¬rain
0,4	0,6	0,1	0,9

Combinations Joint Dist

	R	¬R
C	0,08	0,32
¬C	0,02	0,58

$P(C|\text{rain}) = \frac{P(C, \text{rain})}{P(\text{rain})} = \alpha P(C, \text{rain}) = \alpha \langle 0{,}08, 0{,}02 \rangle = \langle 0{,}8, 0{,}2 \rangle$

var

Conditional Prob

Probability rules:

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

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

Marginalization: $P(a) = P(a \wedge b) + P(a \wedge \neg b) \Leftrightarrow P(X = x_i) = \sum_j P(X = x_i, Y = y_j)$

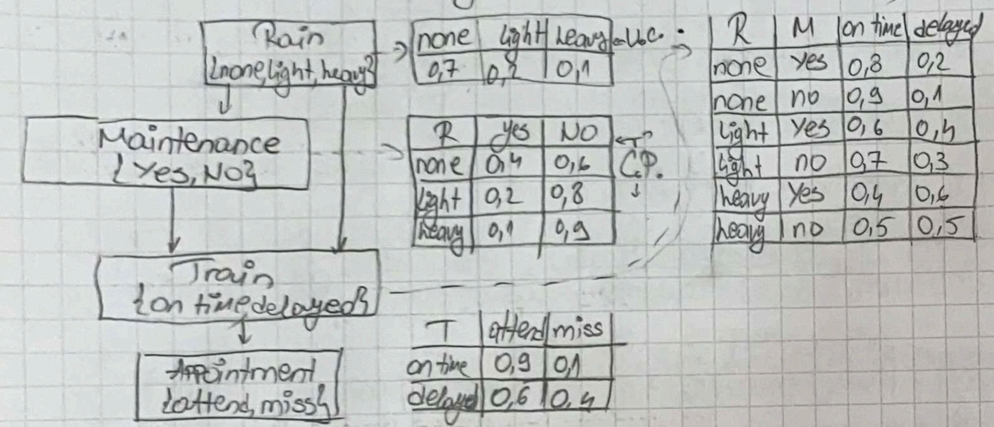
$P(C = \text{cloudy}) = P(C = \text{cloud}, R = \text{rain}) + P(C = \text{cloud}, R = \neg\text{rain}) = 0{,}08 + 0{,}32 = 0{,}40$ ← U.P.

from joint dist to some individual prob

Conditioning $P(a) = P(a|b)P(b) + P(a|\neg b)P(\neg b)$ $\quad P(X = x_i) = \sum_j P(X = x_i | Y = y_j)P(Y = y_j)$

Bayesian network - data struc. 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 $P(x \mid Parents(x))$



none	light	heavy
0,7	0,2	0,1

R	yes	NO
none	0,4	0,6
light	0,2	0,8
heavy	0,1	0,9

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

R	M	on time	delayed
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

Computing Joint Prob: $P(light, no) = P(light)P(no \mid light)$

$P(light, no, delayed, miss) = P(light)P(no \mid light)P(delayed \mid light, no)P(miss \mid delayed)$

Inference (Problem) (zaključivanje)

- Query-variable for which to compute distribution

Evidence variables E: observed variables for event e

- Hidden variables Y: non-evidence, non-query variable (train)
- Goal: Calculate $P(X \mid e)$

$P(Appointment \mid light, no) = \alpha P(App, light, no) = \alpha \underbrace{[P(A, light, no, on\ time) + P(\neg A, delayed)]}_{J.P}$ (normal)

Inference by Enumeration (zaključivanje nabrajanjem): $P(X \mid e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)$

X - query var e - evidence y - ranges over values of hidden var α - normalization of the J.P.

= C.P. is proportional to the J.P.

Approximate Inference

Sampling - one of the values from each of these nodes according to their probability distribution.

R = none (randomly picked) M = yes T = on time A = attend

Likelihood Weighting - Only sampling evidence where is fixed var.

- 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 of the evidence

Markov assumption - Current state depends on only a finite fixed number of previous states

Markov chain - a sequence of random variables where the distribution of each variable follows the Markov assumption

Transitional Model - Construct Markov chain by transitioning from one state to the next state

tommorow (X_{t+1})

	SUN	RAIN
SUN	0,8	0,2
RAIN	0,3	0,7

Today (X_t)

$S \to S \to R \to R \to \ldots$

$X_0 \quad X_1 \quad X_2 \quad X_3$

Sensor Model

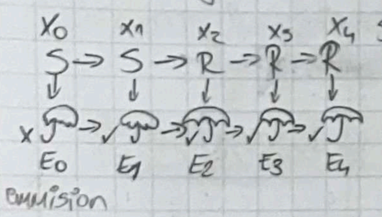
Hidden State	Observation
Robot's position	robot's sensor data
words spoken	audio waveforms
user engagement	website or app analytics
weather	umbrella

Hidden Markov Model - a M.M. for a sys with hidden states that generate some observed event.

		observation (E_t) ✓	observation (E_t) ✗
State(X_t)	Sun	0,2	0,8
	Rain	0,9	0,1

=> Sensor model

Sensor Markov Assumption - the assumption that the evidence variable depends only the corresponding state



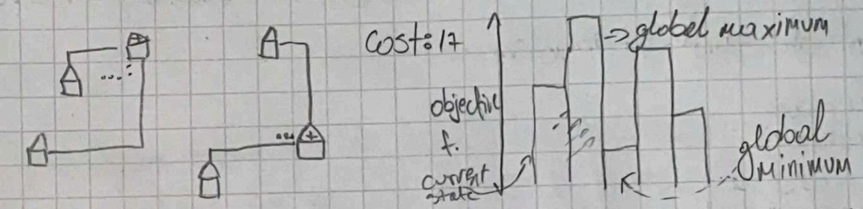
Task	Definition
Filtering	Given observations from start until now, Calculate distribution for current state
Prediction	-||- for a future state
Smoothing	-||- for a past state
Most likely explanation	-||- calculate most likely sequence of states

Optimization L3

Choosing the best option from a set of options

Local search - search algorithms that maintain a single node and searches by moving to a neighboring node. what the goal is?

Put 2 hospitals and minimize distance from each (Manhattan dist)



State-space landscape - vertical bar => config of 2 hospitals.

height - represent function of that state, some value of state (in this case)

Hill Climbing - Choosing optimal solution according to neighbour value

```
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
```

Shoulder - same height values

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-restart	conduct hill climbing multiple times
local beam search	chooses the k highest-valued neighbors

Simulated Annealing (žarenje)

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

- later on, lower "temp": less likely to acc neigh that are worse than curr.

```
function Simulated_Annealing(Problem, max):   -> max numb. of times to run proc
  current = initial state of Problem
  for t = 1 to max:
    T = temperature(t)   -> higher more likely to move to the worst neighbor
    neighbor = random neighbor of current
    ΔE = how much better neighbor is then current
    If (ΔE > 0):
      current = neighbor
    with probability e^(ΔE/T) set current = neighbor
  return current
```

Traveling Salesman Problem - min distance between nodes $\pi - N!!$

Linear Programming types of problems - optimize for mathematical function

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

Ex: Two machines X_1 and X_2. X_1 cost \$50/hour to run, X_2 \$80/hour. Goal is to minimize cost.

- X_1 requires 5 units per hour. X_2 2 per h. Total: 20
- X_1 produces 10 units output per hour. X_2 12. Company needs 90 units of output
- Cost f: $50x_1 + 80x_2$ x_1/x_2 -> hours running

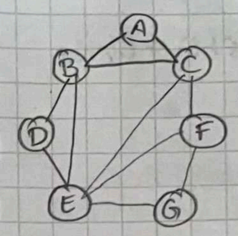
Constraint $5x_1 + 2x_2 \le 20$

Constraint $10x_1 + 12x_2 \ge 90$ (·-1) $-10x_1 - 12x_2 \le -90$

Linear P. Algorithms: Simplex, Interior-Point

Constraint Satisfaction

Student: 1,2,3,4 Taking classes: 1: A,B,C 2: BDE Exam slots: Mon, Tue, Wed.



Can not have exams on same day

-> Constraint graph Variables {A, B, C ... G}
Domains {Mon, Tue, Wed} for each var
Constraints {A≠B, A≠C, B≠C ...}

C.S. Problem

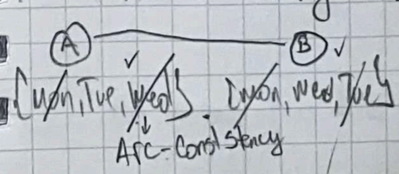
- Set of variables $\{X_1, X_2, \dots X_n\}$
- Set of domains for each variable $\{D_1, D_2, \dots, D_n\}$
- Set of constraints C

Ex: Sudoku

- Variables: empty squares {(0,2), (1,1)..} - represented as coordinate
- Domains {1,2,3,4 ... 9} for each variable
- Constraints - {(0,2) ≠ (1,1) ≠ (1,2)} - enforce what values are allowed
- Hard Constraints - constraints that be satisfied in a correct solution
- Soft Constraints - constraints that express some notion of which solutions are preferred over others

Categories of constraints:

- Unary Constraint - Constraint involving only one variable. {A ≠ Mon}
- Binary Constraint - -||- two var. {A ≠ B}
- Node Consistency - when all the values in a variable's domain satisfy the variable's unary constraints



binary Con.

{A ≠ Mon, B ≠ Tue, B ≠ Mon, A ≠ B}

not node consistent

Arc Consistency

- when all the values in a variable's domain satisfy the variable's binary constraints

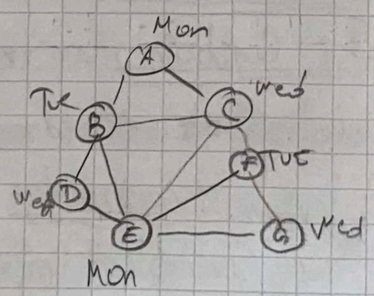
- To make X arc-consi. with respect to Y, remove elements from X's domain until every choice for X has a possible choice for Y.

```
function Revise(CSP, X, Y): #CSP Constraint satisfaction problem
  revised=false
  for x in X.domain:
    if no y in Y.domain satisfies constraint for (X,Y):
      delete x from X.domain
      revised=true
  return revised
```

```
function AC-3 (csp):        Alg. for enforcing arc consistency
  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
      for Z in X.neighbors-{Y}:
        ENQUEUE(queue, (Z,X))
  return true
```

Backtracking Search

```
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
```



Maintaining arc-consistency (inference)

alg. for enforcing (sprovajanje) arc-consistency every time we make a new assignment.

When we make a new assig. to X, calls AC-3, starting with a queue of all arcs(Y,X) where Y is a neighbor of X.

```
function BACKTRACK(assignment, csp):
  if assignment complete: return assignment
  var=SELECTED-UNASSIGNED-VAR(assignment, csp)
  for value in DOMAIN-VALUES(var, assignment, csp):
    if value consistent 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
```

SELECT-UNASSIGNED-VAR

- Minimum remaining values (MRV) heuristic: select the variable that has the smallest domain

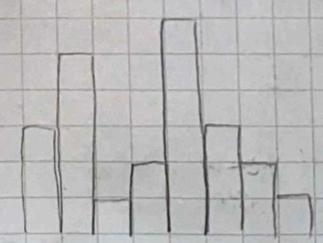
- degree heuristic: select the variable that has the highest degree

DOMAIN-VALUES

- least-constr. values heuristic: return var. in order by number of choices that are ruled out for neighbouring variables

  - try least-constr. values first

Problem formulation



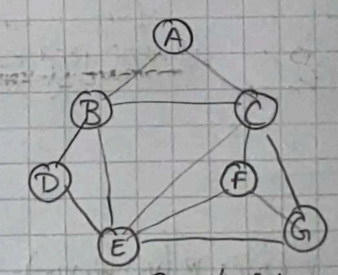
Local search

$50x_1+80x_2$

$5x_1+2x_2 \leq 20$

$-10x_1-12x_2 \leq -90$

Linear Programming



Constraint Satisfaction

Learning

Supervised Learning - given a data set of input-output pairs, learn a function to map inputs to outputs

- **Classification** - s.l. task of learning a f. mapping an input point to a discrete category.

Date	Humidity (relative H.)	Pressure (sea lev, mb)	Rain
Jan 1	93%	999.7	Rain
Jan 2	49%	1015.5	No Rain
-"- 3	79%	1031.1	No Rain
-"- 4	65%	984.9	Rain
-"- 5	90%	975.2	Rain

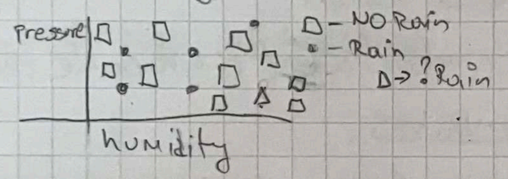
f(humidity, pressure)

f(93, 999.7) = Rain

f(49, 1015.5) = No Rain

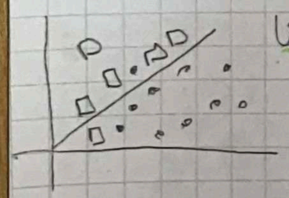
hypothesis function

h(humidity, pressure) ⇒ approx. of f



Nearest-neighbor classification - alg. that, given an input, chooses the class (Rain/No) of the nearest data point to that input. (k=1)

K-nearest-neighbor classification - alg. that, given an input, chooses the most common class out of the k nearest data points to that input



Linear regression

x_1 = Humidity

x_2 = Pressure

Line → $h(x_1, x_2) = \begin{cases} 1 & \text{Rain if } w_0 + w_1x_1 + w_2x_2 \geq 0 \\ 0 & \text{No Rain otherwise} \end{cases}$

Weight Vector: $w\ (w_0, w_1, w_2)$

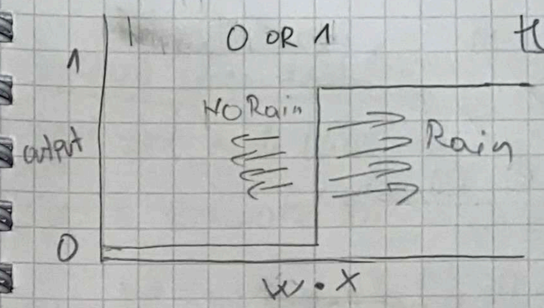
Input Vector: $X\ (1, x_1, x_2)$

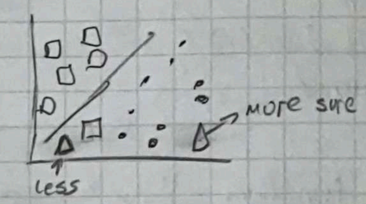
Dot product - taking each of the terms and the vectors and multiplying them together: $w \cdot x = w_0 + w_1x_1 + w_2x_2$

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

Perceptron Learning rule - Given data point (x, y) (0/1), update each weight according to: $w_i = w_i + \alpha(y - h_w(x)) \times x_i$

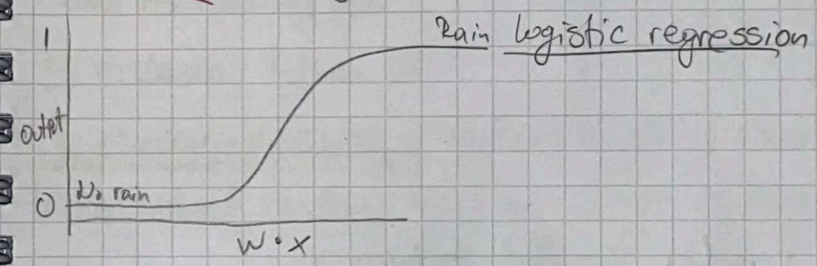
α - learning rate - number for how quickly updating weight values.





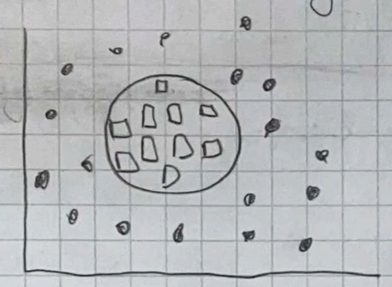
Hard threshold - doesn't allow for notion of less sure or more sure

Soft threshold - any real number of values between 0 and 1.



Support Vector Machines

- **Maximum margin separator** - boundary that maximizes the distance between any of the data points

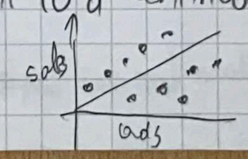


Regression - supervised learning task of learning a function mapping an input point to a continuous value. (output: real number) ads → sales

f(advertising)

f(1200) = 5800

h(advertising)

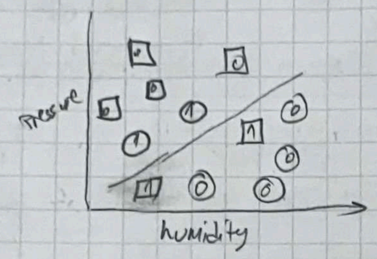


Evaluating Hypotheses

Loss function - f. that expresses how poorly our hypoth. performs (loss of utility)

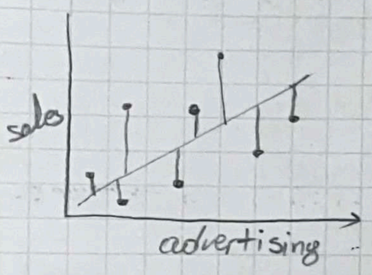
- **0-1 Loss f.**

L(actual, predicted) =
0 if actual = predicted,
1 otherwise



L1 loss function - How far apart were the actual v. and predicted

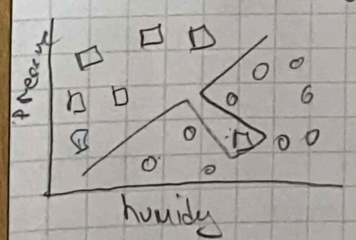
L(actual, predicted) = |actual - predicted| minimize loss



L2 loss function - harshly penalize one that is far away

L(actual, Predicted) = (actual - Predicted)^2

- **Overfitting** - a model that fits too closely to a particular data set and therefore may fail to generalize to future data



No loss but bad generalization

- cost(h) = loss(h) - minimizing loss, but overfitting

cost(h) = loss(h) + complexity(h)

cost(h) = loss(h) + λ complexity(h) λ - parameter to balans loss and complexity

Regularization - Penalizing hypotheses that are more complex to favor simpler, more general hypotheses.

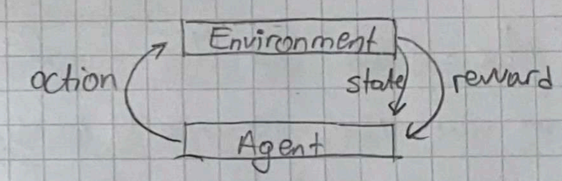
- cost(h) = loss(h) + λ complexity(h)

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

- **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.

scikit-learn - Python lib.

- **Reinforcement learning** - given a set of rewards or punishments learn what actions to take in the future



- **Markov Decision Process** - model for decision-making, representing states, actions, and their rewards
  - Set of states S
  - Set of actions ACTIONS(s)
  - Transition model P(s'|s,a)
  - Reward function R(s,a,s')

Q-learning - method for learning a function $Q(s,a)$ estimate of the value of performing action a in state s.

Overview

- 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

Formula:

Start with $Q(s,a)=0$ for all s,a

Every time we take an action a in state s and observe a reward r we update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(\text{new value estimate} - \overbrace{\text{old value estimate}}^{Q(s,a)})$$

α - learning rate - how much we value new info compared to old one.

If $\alpha=1$ we consider only new info. $\alpha=0$ - ignor new info

$$Q(s,a) \leftarrow Q(s,a) + \alpha((r + \underbrace{\text{future reward estimate}}_{\max_{a'} Q(s',a')}) - Q(s,a))$$

$\max_{a'} Q(s',a')$ - of all possible action take one with highest reward

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

Every time we experience some new reward, we take into account.

- Update our estimate of how good is this action and in the future we can make decisions based on that algorithm

Greedy Decision-Making

- When in state s, choose action a with highest $Q(s,a)$

Explore vs Exploit

Exploitation - Using knowledge that the AI already has.

Exploration - Exploring other actions that we may not have explored temeljito thoroughly before.

ε-greedy algorithm

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

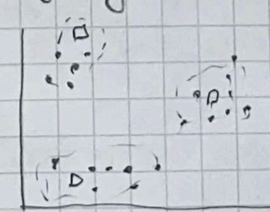
Function approximation - approx. $Q(s,a)$, often by a function combining various features, rather than storing one value for every state-action pair.

- **Unsupervised Learning**
- Given input data without any additional feedback, learn patterns

Clustering - organizing a set of objects into groups in such a way that similar objects tend to be in the same group

Ex of clustering apps: Genetic research, Image segmentation, Market research, Medical imaging, Social network analysis.

K-means clustering - alg. for clustering data based on repeatedly assigning points to clusters and updating those clusters' centers.



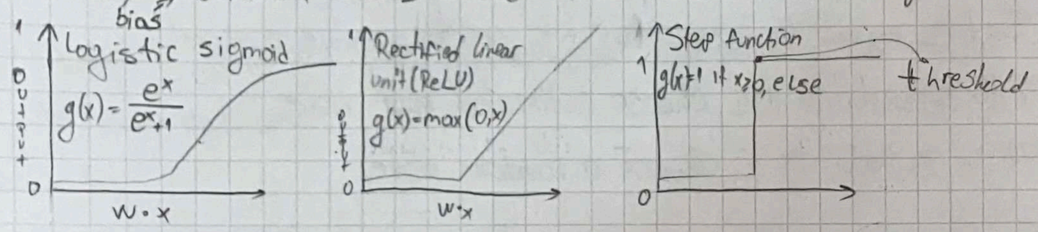
Learning: Supervised, Reinforcement, Unsupervised

Neural Networks

Artificial neural network (ANN) - mathematical model for learning inspired by biological neural networks

- **Model math. f.** from inputs to outputs based on the structure and parameters of the network.
- Allows for learning the network's parameters based on data

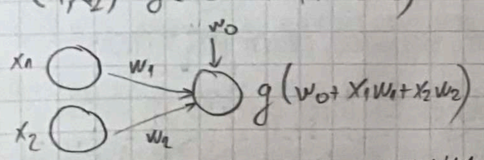
$h(x_1, x_2) = \underset{\text{bias}}{w_0} + w_1 x_1 + w_2 x_2$ $x_1, x_2 \rightarrow$ input variables w_0, w_1, w_2 - weight

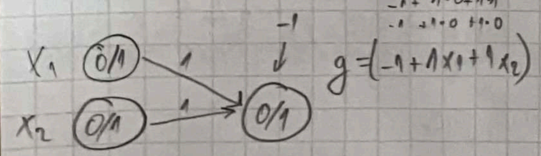


Activation f. - f. that determines when it is that this output becomes active

changes to a 1 instead of being a 0.

$h(x_1, x_2) \overset{\text{Activ}}{=} g(w_0 + x_1 w_1 + x_2 w_2)$





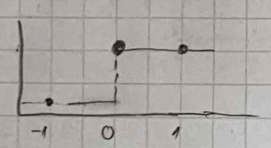
$-1 + 1 \cdot 0 + 1 \cdot 1$
$-1 + 1 \cdot 0 + 1 \cdot 1$
$-1 + 1 \cdot 1 + 1 \cdot 0$
$-1 + 1 \cdot 1 + 1 \cdot 1$

$g = (-1 + 1x_1 + 1x_2)$

$g\left(\sum_{i=1}^{n} x_i w_i + w_0\right)$

OR

X	y	f(x,y)
0	0	0
0	1	1
1	0	1
1	1	1



AND

X	y	f(x,y)
0	0	0
0	1	0
1	0	0
1	1	1

Gradient descent - alg. for minimizing loss when training neural network

- Start with random choice of weights

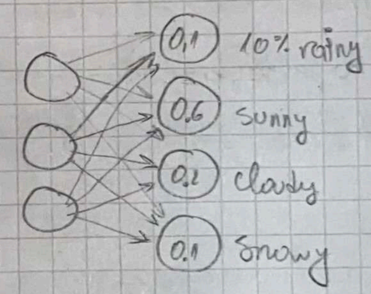
Repeat:

- Calculate the gradient based on **all data points** (expensive) - direction that will lead to decreasing loss
- Update weights according to the gradient

Gradient - slope (nagib) - direction the loss f. is moving at any particular point

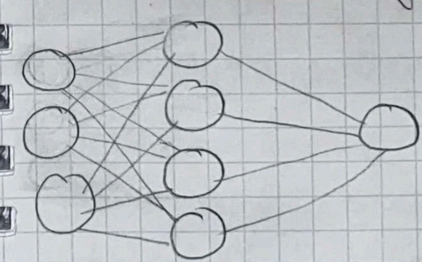
In which direction should we be moving these weights in order to minimize loss.

- **Stochastic Gradient Descent**: -||- **One data point** - less accurate
- **Mini-Batch Gradient Descent**: -||- **one small batch** (gamica)



- **Perceptron** - Only capable of learning linearly separable decision boundary.

Multilayer neural network - Ann with an input layer, an output layer, and at least one hidden layer



- Based on the error from the output i can backpropagate the error and estimate for what the error is for each of hidden layer nodes.

Backpropagation - alg. for training nn with hidden layers.

- Start with random choice of weights
- Repeat:
- Calculate error for output layer
- for each layer, starting with output layer, and moving inwards towards earliest hidden layer:
- Propagate error back one layer
- Update weights.

Deep n.n. - n.n. with multiple hidden layers.

Overfitting - over-reliant on certain nodes to calculate things just purely based on the input data that doesn't allow us to generalize well to the output

Dropout - temporarily removing units - selected at random - from a n.n. to prevent over-reliance on certain units.

TensorFlow - library developed by Google for quickly creating, modeling... n.n.

playground.tensorflow.org

Keras - API, a set of f. to manipulate n.n. inside of tensorflow.

Dense (gusto) - each of the layer nodes connected to each from the previous layer.

Computer vision - computational methods for analyzing and understanding digital images.

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

kernel

0	-1	0
-1	5	-1
0	-1	0

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

10	20
40	50

20	20	20
20	20	20
20	20	20

-1	-1	-1
-1	8	-1
-1	-1	-1

$20\cdot(-1)+20\cdot(-1) \ldots = 0$

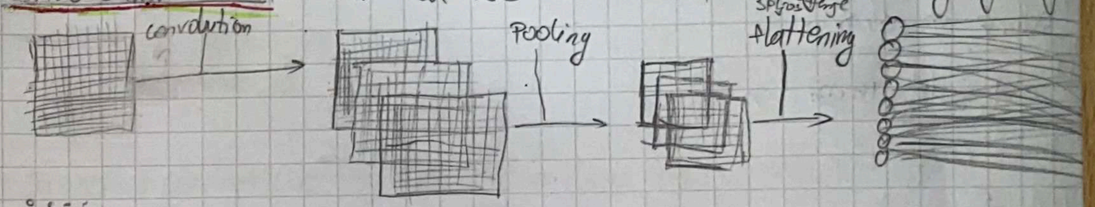
Pooling - reducing the size of an input by sampling from regions in the input

Max-pooling - pooling by choosing the maximum value in each region.

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

50	110
20	40

Convolutional n.n. - n.n. that use convolution, usually for analyzing images

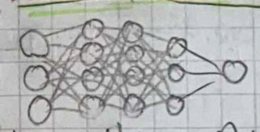


First convolution and pooling - low-lvl. features: edges, curves, shapes

Second -11- : high-lvl. features: objects

mnist - handwriting dataset

Feed-forward n.n. - n.n. that has connections only in one direction.



Input → Network → Output

Recurrent N.N. - n.n. that generates output that feeds back into its own inputs. Ex. Microsoft CaptionBot

Input → Network → Output 1-1

feed its own output back into itself as input for the next time N-N

Input → Network → Output first word

same network dif. input → Network → Output One word at a time

Network → Output

Network → Output

frame: Input → Network Youtube classification of video category

Input → Network

Input → Network → Output

Language

- Natural Language Processing spans all tasks where the AI gets human language as input.
- Automatic summarization - summary of the text as output
- Information extraction - corpus of text extracts data as output
- Language identification -
- Machine translation
- Named entity recognition
- Speech recognition
- Text classification
- Word sense disambiguation (rozjaśnianie) - choose the right meaning of a word

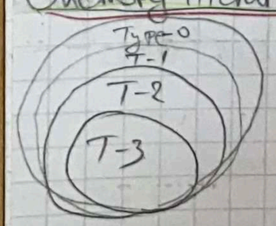
Syntax - sentence structure

Semantics - meaning of words or sentences

Context-Free Grammar - text is abstracted from its meaning to rep. struc. of the sent. using rules

Formal grammar a sys. of rules for generating sentences in a language

Chomsky Hierarchy



T-0 → Unrestricted Grammar (Recognized by Turing Machine)

T-1 → Context Sensitive Grammar (Acc by Linear Bound Automata)

T-2 → Context free Grammar (Acc by Push Down Automata)

T-3 → Regular Grammar (Acc by Finite Automata)

She saw the city → terminal symbols

↓ ↓ ↓ ↓

N V D N → non-terminal symbols

N → she | city | car

D → the | a | an

V → saw | ate | walked

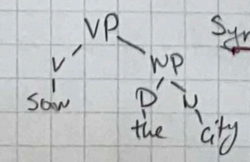
P → to | on | over

ADJ → blue | busy | old

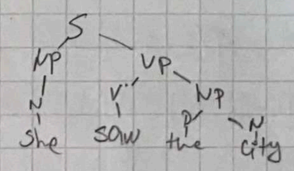
NP → N | DN Noun phrase → Noun | determiner followed by a noun

VP → V | V NP Verb Phrase

Syntax tree



S → NP VP sentence



nltk - Python lib for NL

n-grams - a contiguous sequence of n items from a sample of text

Character n-gram → -||- of n characters from a sample of text

Word n-gram -||- of n words from a sample of text

Unigram -||- of 1 item from a sample of text

bigram / trigrams - 2/3 items from a sample of text

Tokenization - the task of splitting a sequence of characters into pieces (tokens)

Word / Sentence tokenization -||- of words / sentences into tokens

Markov Models - Train the model on text, and then establish probabilities for every n-th token in an n-gram based on the n words proceding it.

Bag-of-words Model - Model that represents text as an unordered collection of words. Model ignors syntax and considers only the meanings of the words in the sentences.

Text categorization - email Inbox or Spam

1. My grandson loved it!
2. Product broke after a few days

Based on words 1. is positive, 2. is negative

Naive Bayes - technique that's can be used in sentiment analysis with bag-of-words model. In sentiment analysis, we are asking "what is probability that the sentence is positive/negative given the words in the sentence

$$P(\text{Positive} \mid \text{"my"}, \text{"grandson"}, \text{"loved"}, \text{"it"}) = \frac{P(\text{"my"}, \text{"grandson"}, \text{"loved"}, \text{"it"} \mid \text{Positive})\, P(☺)}{P(\text{"my"}, \text{"grandson"}, \text{"loved"}, \text{"it"})}$$

↳ proportional to $P(\text{"my"}, \text{"grandson"}, \text{"loved"}, \text{"it"} \mid \text{Positive})\, P(\text{Positive}) = P(☺, \text{"sentence"})$

↳ naively prop. to $P(☺)P(\text{"my"} \mid ☺)P(\text{"grandson"} \mid ☺)P(\text{"loved"} \mid ☺)P(\text{"it"} \mid ☺)$

$$P(☺) = \frac{\text{number of positive samples}}{\text{number of total samples}}$$

☺	☹
0.49	0.51

	☺	☹
my	0.30	0.20
grandson	0.01	0.02
loved	0.32	0.08
it	0.30	0.40

☺ = 0.00014112 = 0.49 · 0.30 · ...
☹ = 0.00006528 = 0.51 · 0.20 · ...

Additive smoothing - adding value α to each value in our distribution to smooth the data. To prevent multiplying the whole prob. by 0.

Laplace smoothing - adding 1 to each value in our distribution: pretending we've seen each value one more time than we actually have

Information retrieval - the task of finding relevant documents in response to a user query

Topic modeling - models for discovering the topics for a set of documents

Term frequency - number of times a term appears in a document

Function words - (am, by, is) - words that have little meaning on their own, but are used to grammatically connect other words

Content words - (alg., category, computer, ..) - words that carry meaning independently

Inverse document frequency - measure of how common or rare a word is across doc.

$$IDF: \log \frac{\text{Total Documents}}{\text{Num Doc Containing (word)}}$$

tf-idf - Ranking of what words are important in a doc. by multiplying term frequency (TF) by IDF

- **Semantics**

Information extraction - the task of extracting knowledge from documents

WordNet - is database similar to a dictionary where words are given definitions as well as broader categories.

One-Hot Representation - Each word is represented with a vector that consists of as many values as we have words.

[1,0,0,0] (he) [0,1,0,0] (wrote) [0,0,1,0] (a) [0,0,0,1] (book)

- Incredibly inefficient

Distributed Representation - repres. of meaning distributed across multiple values

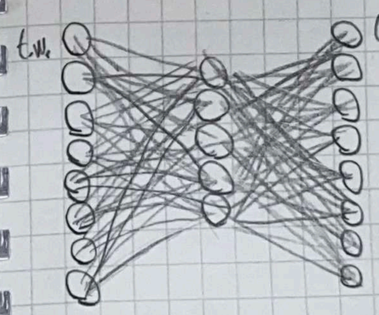
[-0.34, -0.08, 0.02, -0.18, 0.22, ...] he [-0.27, 0.40, 0.00...] wrote [-0.12, -0.25...] a [0.23, -0.16...] book

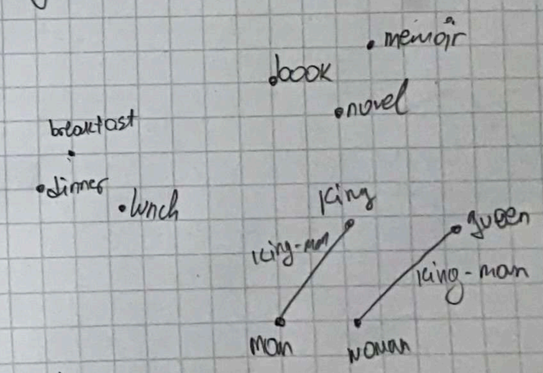
Skip-gram architecture - n.n. arch. for predicting context words given a target word.

for [ ] he ate
lunch / dinner / breakfast

Word2Vec - alg. for generating distributed representations of words. It does by SGA

N.N has an input unit for every target word.





- Training dataset using the backpropagation alg.