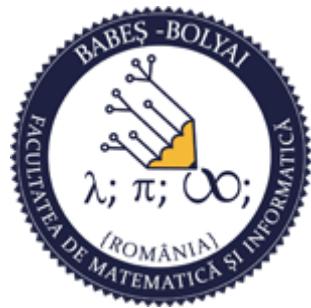




UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

C. Sisteme inteligente

■ Sisteme care învață singure

- Arbori de decizie
- Rețele neuronale artificiale
- Mașini cu suport vectorial
- Algoritmi evolutivi

■ Sisteme bazate pe reguli

■ Sisteme hibride

B. Rezolvarea problemelor prin căutare

■ Definirea problemelor de căutare

■ Strategii de căutare

- Strategii de căutare neinformate
- Strategii de căutare informate
- Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
- Strategii de căutare adversială

Materiale de citit și legături utile

- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 și 11 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997* [link](#)
- capitolul 1 din *C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006* [link](#)
- capitolul 1 din *S. Guido, A. C. Müller, Introduction to Machine Learning with Python, O'Reilly Media, 2016* [link](#)
- capitolele 1 și 2 din *A. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly Media, 2019* [link](#)

Conținut

❑ Sisteme inteligente

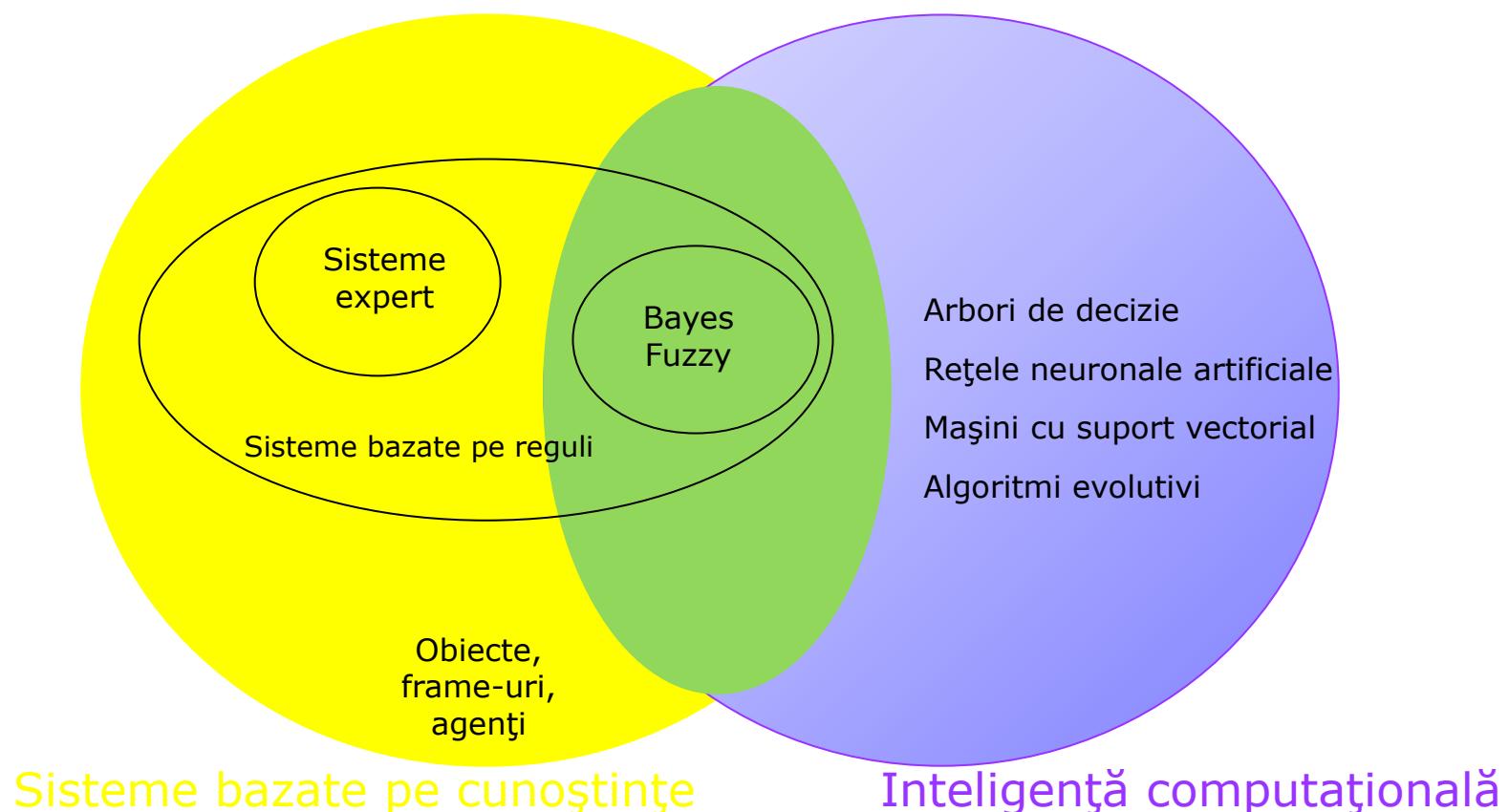
■ Sisteme care învață singure (SIS)

❑ Instruire (învățare) automata (Machine Learning - ML)

- Problematică
- Proiectarea unui sistem de învățare automată
- Tipologie
 - Învățare supervizată
 - Învățare nesupervizată
 - Învățare cu întărire
 - Teoria învățării

❑ Exemple de sisteme

Sisteme inteligente



Sisteme inteligente – SIS – Învățare automată

- **Problematica**
 - “How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?”

- **Aplicații**
 - Recunoaștere de imagini și semnal vocal
 - Recunoașterea scrisului de mână
 - Detecția fețelor
 - Înțelegerea limbajului vorbit
 - Computer vision
 - Detecția obstacolelor
 - Recunoașterea amprentelor
 - Supraveghere bio
 - Controlul roboților
 - Predicția vremii
 - Diagnosticare medicală
 - Detectia fraudelor

Sisteme inteligețe – SIS – Învățare automată

- Definire
 - Arthur Samuel (1959)
 - “field of study that gives computers the ability to learn without being explicitly programmed”
 - Înșestrarea computerelor cu abilitatea de a învăța pe baza experienței
 - Herbert Simon (1970)
 - “Learning is any process by which a system improves performance from experience.”
 - Tom Mitchell (1998)
 - “a well-posed learning problem is defined as follows: He says that a computer program is set to learn from an experience E with respect to some task T and some performance measure P if its performance on T as measured by P improves with experience E”
 - Ethem Alpaydin (2010)
 - Programming computers to optimize a performance criterion using example data or past experience.
 - John L. Hennessy, President of Stanford (2000–2016)
 - Machine learning is the hot new thing
 - Bill Gates (Microsoft co-founder)
 - A breakthrough in machine learning would be worth ten Microsofts
- Necesitate
 - Sisteme computaționale mai bune
 - Sisteme dificil sau prea costisitor de construit manual
 - Sisteme care se adaptează automat
 - Filtre de spam
 - Sisteme care descoperă informații în baze de date mari → data mining
 - Analize financiare
 - Analize de text/imagini
 - Înțelegerea organismelor biologice



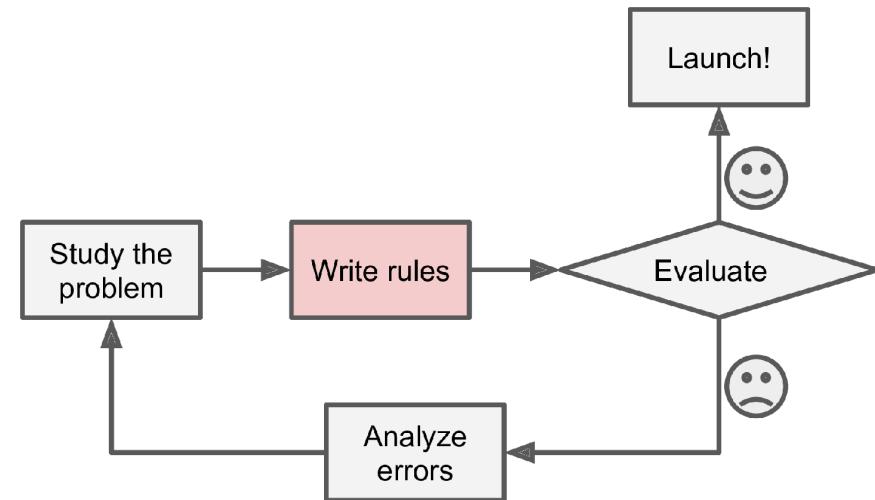
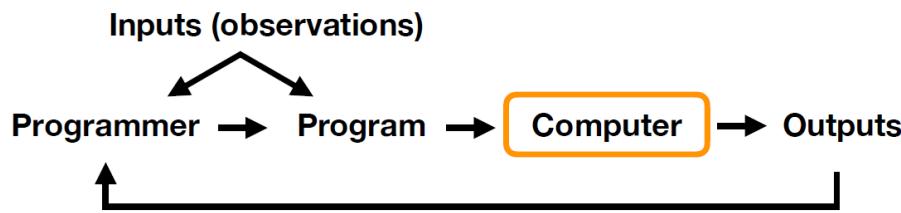
Sisteme inteligente – SIS – Învățare automată

❑ Persoane importante și/sau interesante

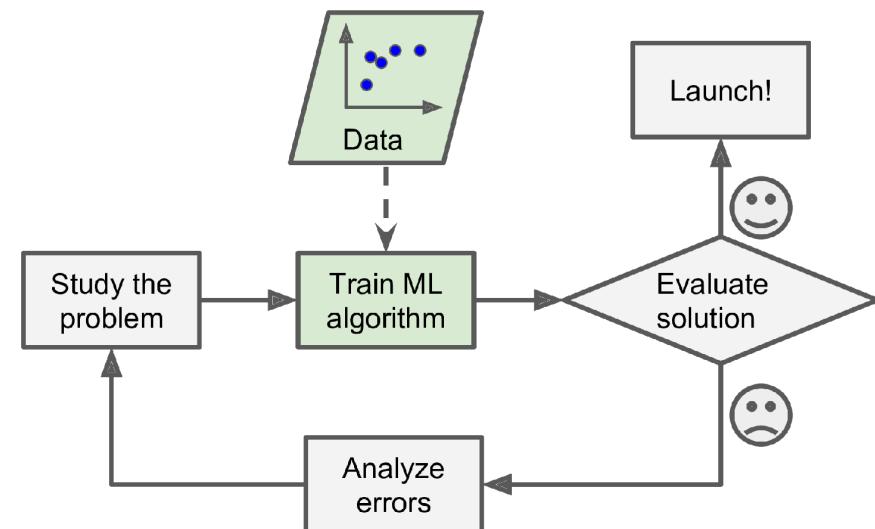
- Peter Norvig
- Stuart Russell
- Michael Jordan (Bayesian Nets), Andrew Ng
- Elon Musk, Andrej Karpathy (TESLA)
- Fei Fei Li (AI for social good)
- Richard Sutton (Reinforcement Learning)
- Jurgen Schmidhuber (LSTM)
- Geoffrey Hinton , Yann LeCun, and Yoshua Bengio (CNN and deep CNN)
- John Koza (Genetic Programming)
- Rana el Kaliouby (Affectiva)
- ...alții...

Sisteme inteligente – SIS – Învățare automată

□ Programarea tradițională



□ Machine Learning



Sisteme inteligente – SIS – Învățare automată

□ Proiectare

- Îmbunătățirea task-ului T
 - Stabilirea scopului (ceea ce trebuie învățat) - funcției obiectiv – și reprezentarea sa
 - Alegerea unui algoritm de învățare care să realizeze inferență (previziunea) scopului pe baza experienței
- respectând o metrică de performanță P
 - Evaluarea performanțelor algoritmului ales
- bazându-se pe experiența E
 - Alegerea bazei de experiență
- Exemplu
 - T: jucarea jocului de dame
 - P: procentul de jocuri câștigate împotriva unui oponent oarecare
 - E: exersarea jocului împotriva lui însuși
 - T: recunoașterea scrisului de mână
 - P: procentul de cuvinte recunoscute corect
 - E: baze de date cu imagini cu cuvinte corect adnotate
 - T: separarea spam-urilor de mesajele obișnuite
 - P: procentul de email-uri corect clasificate (spam sau normal)
 - E: baze de date cu email-uri adnotate

Sisteme inteligente – SIS – Învățare automată

■ Stabilirea scopului (ceea ce trebuie învățat)

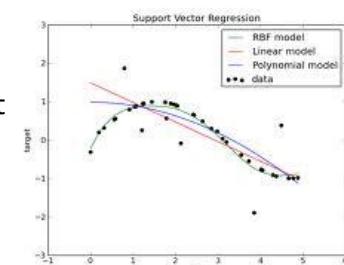
■ SI pentru predicții / regresii

- Scop: predicția ieșirii pentru o intrare nouă folosind un model învățat anterior
- Ex.: predicția vânzărilor dintr-un produs pentru un moment de timp viitor în funcție de preț, lună calendaristică, regiune, venit mediu pe economie



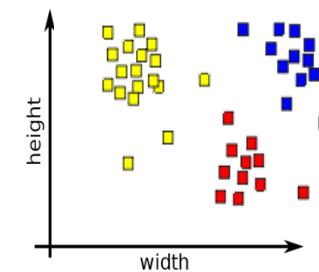
■ SI pentru regresii simbolice

- Scop: estimarea formei unei funcții uni sau multivariată folosind un model învățat anterior
- Ex.: estimarea funcției care modelează conturul unei suprafete



■ SI pentru clasificare

- Scop: clasificarea unui obiect într-una sau mai multe categorii (clase) – cunoscute anterior sau nu - pe baza caracteristicilor (atributelor, proprietăților) lui
- Ex.: sistem de diagnoză pentru un pacient cu tumoare: nevasculară, vasculară, angiogenă



■ SI pentru planificare

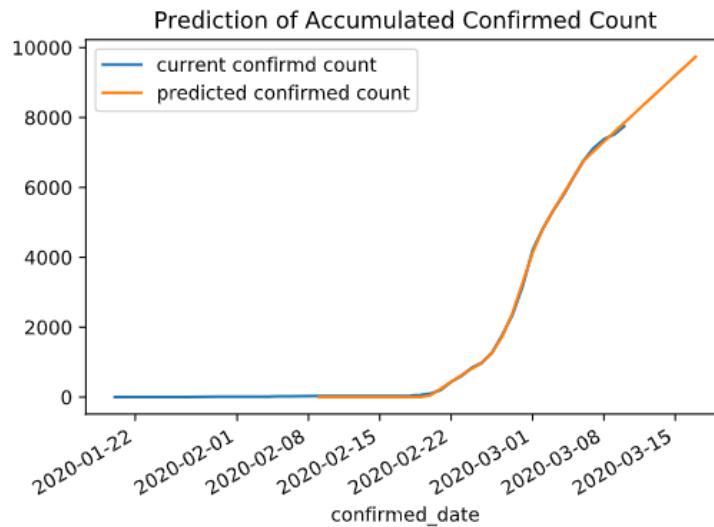
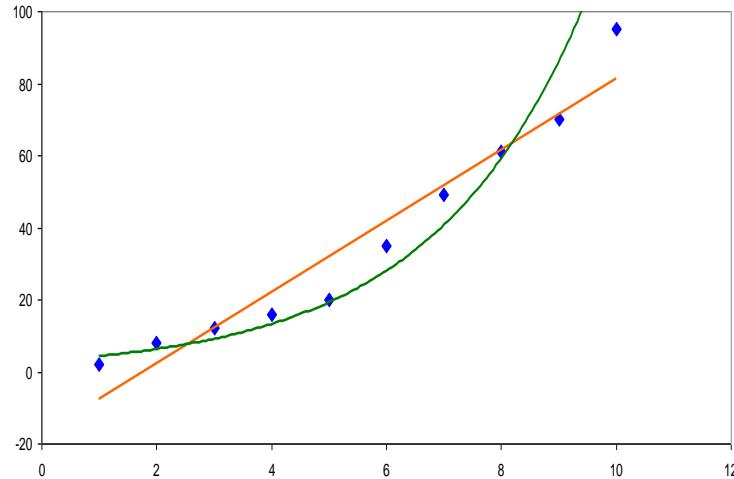
- Scop: generarea unei succesiuni optime de acțiuni pentru efectuarea unei sarcini
- Ex.: planificarea deplasării unui robot de la o poziție dată până la o sursă de energie (pentru alimentare)



Sisteme inteligente – SIS – Învățare automată

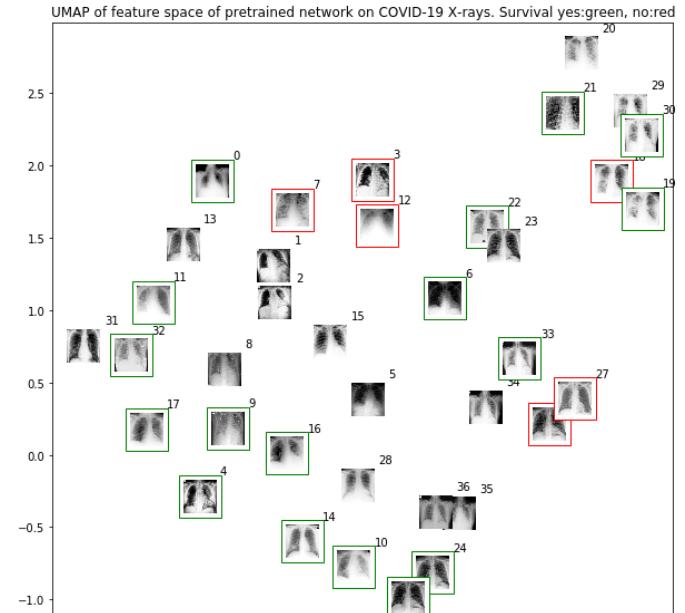
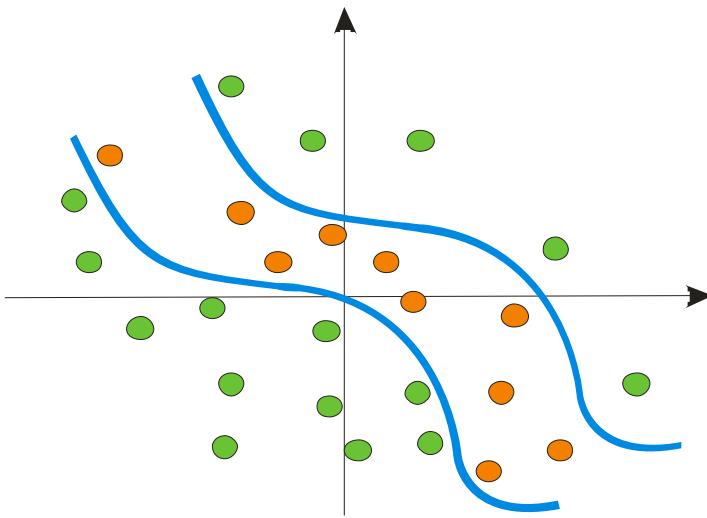
□ Stabilirea scopului (ceea ce trebuie învățat)

- Probleme de predicție / regresie
 - Se dau date (de intrare și ieșire) trecute
 - Numărul de persoane infectate cu SARS-CoV-2 pentru ultimele 3 luni
 - Se cer predicții viitoare (pentru anumite date de intrare)
 - Numărul de persoane care se vor infecta cu SARS-CoV-2 în urmatoarele 7 zile / 4 săptamani / 2 luni



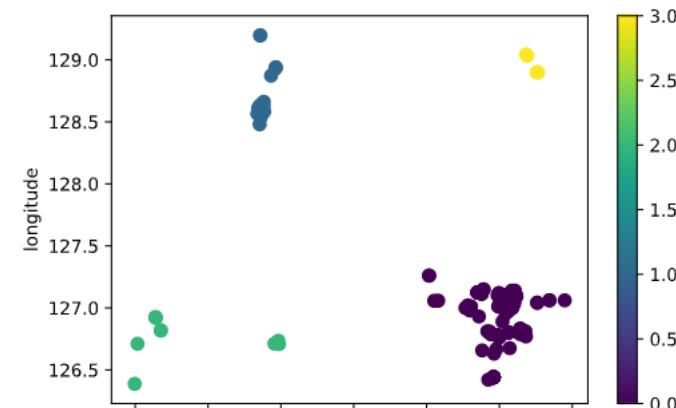
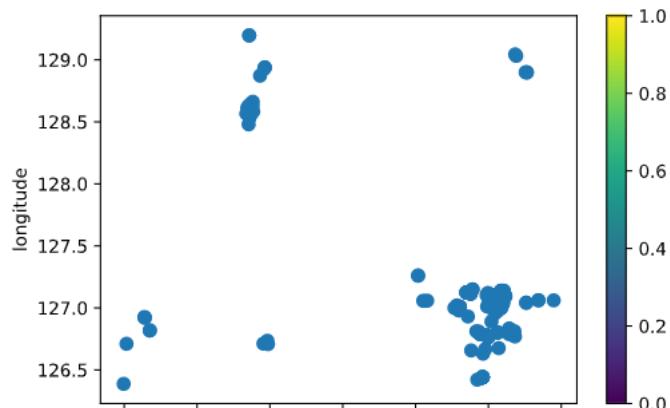
Sisteme inteligente – SIS – Învățare automată

- Stabilirea scopului (ceea ce trebuie învățat)
 - Probleme de clasificare
 - Se dau date (de intrare și ieșire) trecute
 - Imagini RMN de la pacienți infectați cu SARS-CoV-2 și de la martori (sănătoși)
 - Se cer predicții viitoare (pentru anumite date de intrare)
 - Să se prezică, pe baza RMN-ului, dacă o persoană este infectată sau nu cu SARS-CoV-2



Sisteme inteligente – SIS – Învățare automată

- ❑ Stabilirea scopului (ceea ce trebuie învățat)
 - Probleme de clusterizare
 - ❑ Se dau date (de intrare)
 - Localizarea geografică a unor persoane infectate cu SARS-CoV-2
 - ❑ Se cere identificarea anumitor structuri în aceste date
 - Modul de grupare a celor infectați pe regiuni (Densitatea acestor regiuni)



Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea funcției obiectiv
 - Care este funcția care trebuie învățată?
 - Ex.: pentru jocul de dame → funcție care:
 - alege următoarea mutare
 - evaluează o mutare
 - obiectivul fiind alegerea celei mai bune mutări
 - Reprezentarea funcției obiectiv
 - Diferite reprezentări
 - Tablou (tabel)
 - Reguli simbolice
 - Funcție numerică
 - Funcții probabilistice
 - Ex. Jocul de dame
 - Combinație liniară a nr. de piese albe, nr. de piese negre, nr. de piese albe compromise la următoarea mutare, r. de piese albe compromise la următoarea mutare
 - Există un compromis între
 - expresivitatea reprezentării și
 - ușurința învățării
 - Calculul funcției obiectiv
 - Timp polinomial
 - Timp non-polinomial

Sisteme inteligente – SIS – Învățare automată

□ Proiectare → Alegerea unui algoritm de învățare

■ Algoritmul

- folosind datele de antrenament
- induce definirea unor ipoteze care
 - să se potrivească cu acestea și
 - să generalizeze cât mai bine datele ne-văzute (datele de test)

■ Principiul de lucru de bază

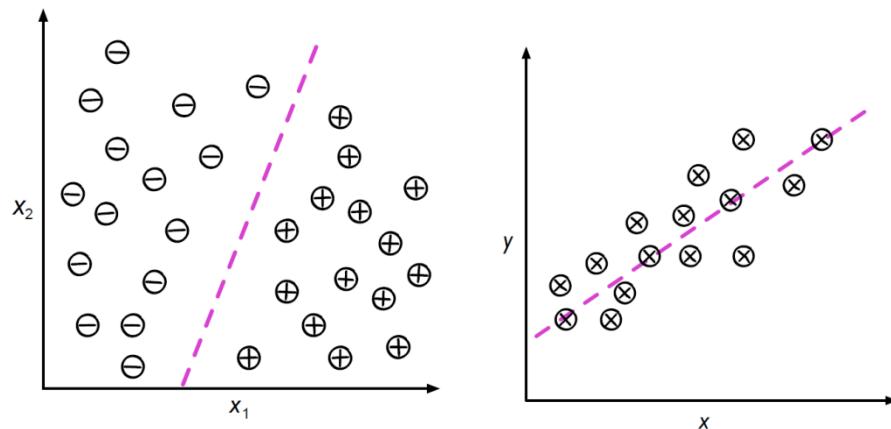
- Minimizarea unei erori (funcție de cost – loss function)

■ Tipuri de algoritmi după metodologia de învățare automată

- Învățare supervizată
 - Ex. regresie, clasificare
- Învățare nesupervizată
 - Ex. clusterizare, reducerea numărului de dimensiuni
- Învățare prin întărire
 - Ex. planning, gaming

Sisteme inteligente – SIS – Învățare automată

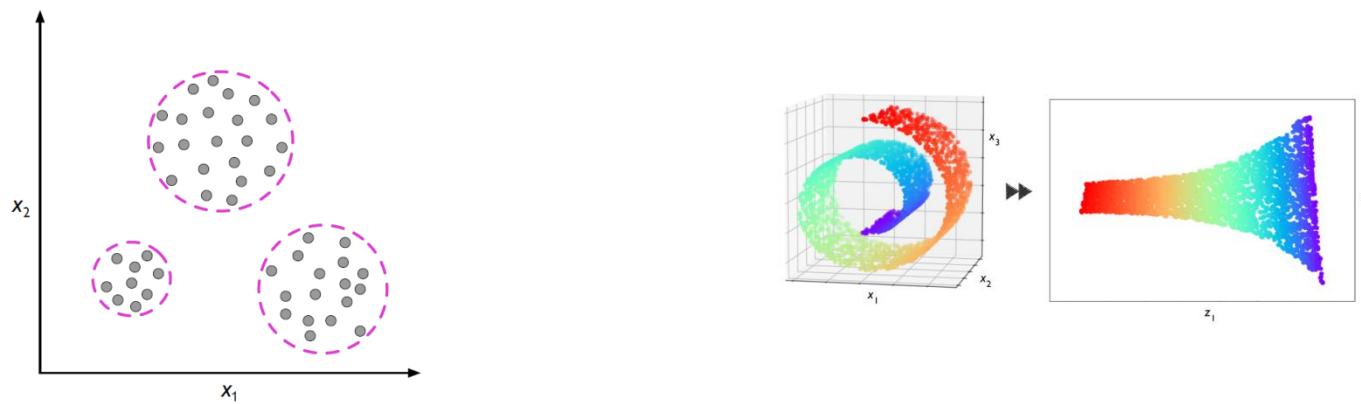
- Proiectare → Alegerea unui algoritm de învățare → Tipuri de algoritmi după metodologia de învățare automată
 - Învățare supervizată
 - Ex. regresie, clasificare
 - Caracteristici
 - Date etichetate (se cunosc o parte din datele de intrare și ieșire *)
 - Feedback direct în timpul învățării – algoritmul se adaptează la datele de intrare și ieșire
 - Predicție a datelor de ieșire (fiind cunoscute niște date de intrare diferite de cele din *)



)

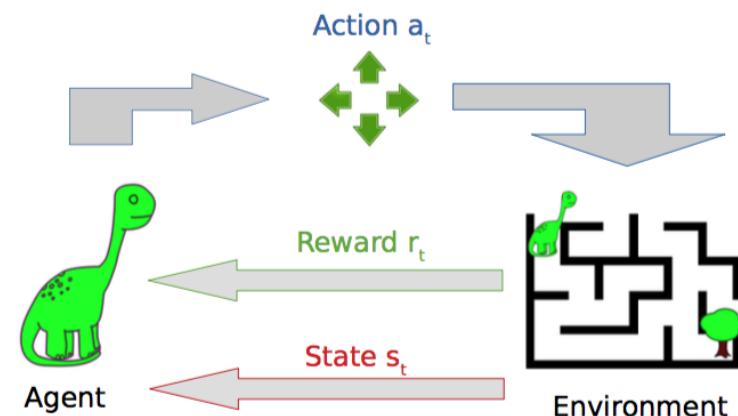
Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învățare → Tipuri de algoritmi după metodologia de învățare automată
 - Învățare supervizată
 - Învățare nesupervizată
 - Ex. clusterizare, reducerea numărului de dimensiuni
 - Carcteristici
 - Date neetichetate (se cunosc o parte din datele de intrare**)
 - Fără feedback direct în timpul învățării – pentru că nu se cunosc datele de ieșire
 - Identificarea unor structuri în date (generarea de date de ieșire pentru datele de intrare din **)



Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învățare → Tipuri de algoritmi după metodologia de învățare automată
 - Învățare supervizată
 - Învățare nesupervizată
 - Învățare prin întărire
 - Ex.
 - Caracteristici
 - Predicția unor secvențe de decizii / de acțiuni
 - Sistem de recompense (pentru fiecare decizie / acțiune)
 - Se învață un model de acțiune (o serie de acțiuni care trebuie efectuate)



Sisteme inteligente – SIS – Învățare automată

- Proiectare → Alegerea unui algoritm de învățare
 - Algoritmul
 - folosind datele de antrenament
 - induce definirea unor ipoteze care
 - să se potrivească cu acestea și
 - să generalizeze cât mai bine datele ne-văzute (datele de test)
 - Principiul de lucru de bază
 - Minimizarea unei erori (funcție de cost – loss function) pentru datele de antrenament
 - Eroarea de predicție (cât de departe sunt valorile prezise față de valorile reale)
 - Eroarea de clasificare (câte exemple au fost clasificate corect)
 - Eroarea creării unor structuri (cât de ne-omogene sunt structurile produse)
- Proiectare → Evaluarea unui sistem de învățare
 - Experimental
 - Compararea diferitelor metode pe diferite date (cross-validation)
 - Colectarea datelor pe baza performanței
 - Acuratețe, timp antrenare, timp testare
 - Aprecierea diferențelor dpdv statistic
 - Teoretic
 - Analiza matematică a algoritmilor și demonstrarea de teoreme
 - Complexitatea computațională
 - Abilitatea de a se potrivi cu datele de antrenament
 - Complexitatea eșantionului relevant pentru o învățare corectă

Sisteme inteligente – SIS – Învățare automată

□ Proiectare → Evaluarea unui sistem de învățare

■ Compararea performanțelor a 2 algoritmi în rezolvarea unei probleme

□ Indicatori de performanță

- Parametrii ai unei serii statistice (ex. media)
- Proporție calculată pentru serie statistică (ex. acuratețea)

□ Comparare pe baza intervalelor de încredere

- Pe o problemă și 2 algoritmi care o rezolvă
- Performanțele algoritmilor: p_1 și p_2
- Intervalele de încredere corespunzătoare celor 2 performanțe $I_1 = [p_1 - \Delta_1, p_1 + \Delta_1]$ și $I_2 = [p_2 - \Delta_2, p_2 + \Delta_2]$
- Dacă $I_1 \cap I_2 = \emptyset \rightarrow$ algoritmul 1 este mai bun decât algoritmul 2 (pt problema dată)
- Dacă $I_1 \cap I_2 \neq \emptyset \rightarrow$ nu se poate spune care algoritm este mai bun

□ Interval de încredere pentru medie

- Pentru o serie statistică de volum n , cu media (calculată) m și dispersia σ să se determine intervalul de încredere al valorii medii μ
- $P(-z \leq (m-\mu)/(\sigma/\sqrt{n}) \leq z) = 1 - \alpha \rightarrow \mu \in [m - z\sigma/\sqrt{n}, m + z\sigma/\sqrt{n}]$
- $P = 95\% \rightarrow z = 1.96$
- Ex. Problema rucsacului rezolvată cu ajutorul algoritmilor evolutivi

□ Interval de încredere pentru acuratețe

- Pentru o performanță p (acuratețe) calculată pentru n date să se determine intervalul de încredere
- $P \in [p - z(p(1-p)/n)^{1/2}, p + z(p(1-p)/n)^{1/2}]$
- $P = 95\% \rightarrow z = 1.96$
- Ex. Problemă de clasificare rezolvată cu ajutorul Mașinilor cu suport vectorial

$P=1-\alpha$	z
99.9%	3.3
99.0%	2.577
98.5%	2.43
97.5%	2.243
95.0%	1.96
90.0%	1.645
85.0%	1.439
75.0%	1.151

Sisteme inteligente – SIS – Învățare automată

□ Proiectare → Alegerea bazei de experiență

■ Bazată pe

□ Experiență directă

- Perechi (intrare, ieșire) utile pt. funcția obiectiv
- Ex. Jocul de dame → table de joc etichetată cu mutare corectă sau incorectă

□ Experiență indirectă

- Feedback util (diferit de perechile I/O) pt funcția obiectiv
- Ex. Jocul de dame → secvențe de mutări și scorul final asociat jocului

■ Surse de date

□ Exemple generate aleator

- Exemple pozitive și negative

□ Exemple pozitive colectate de un "învățător" benevol

□ Exemple reale

■ Compoziție

□ Date de antrenament

□ Date de test

■ Caracteristici

□ Date independente

- Dacă nu → clasificare colectivă

□ Datele de antrenament și de test trebuie să urmeze aceeași lege de distribuție

- Dacă nu → învățare prin transfer (*transfer learning/inductive transfer*)

- recunoașterea mașinilor → recunoașterea camioanelor
- analiza textelor
- filtre de spam

Sisteme inteligente – SIS – Învățare automată

- ❑ Proiectare → Alegerea bazei de experiență
 - Tipuri de atrbute ale datelor
 - ❑ Cantitative → scară nominală sau rațională
 - Valori continue → greutatea
 - Valori discrete → numărul de computere
 - Valori de tip interval → durata unor evenimente
 - ❑ Calitative
 - Nominale → culoarea
 - Ordinale → intensitatea sunetului (joasă, medie, înaltă)
 - ❑ Structurate
 - Arbori – rădăcina e o generalizare a copiilor (vehicol → mașină, autobus, tractor, camion)
 - Transformări asupra datelor
 - ❑ Standardizare → atrbute numerice
 - Înlăturarea efectelor de scară (scări și unități de măsură diferite)
 - Valorile brute se transformă în scoruri z
 - $Z_{ij} = (x_{ij} - \mu_j)/\sigma_j$, unde x_{ij} – valoarea atrbutului al j -lea al instanței i , μ_j (σ_j) este media (abaterea) atrbutelor j pt. toate instanțele
 - ❑ Selectarea anumitor atrbute

Sisteme inteligente – SIS – Învățare automată

❑ Învățare supervizată

- Definire
- Exemple
- Proces
- Calitatea învățării
 - ❑ Metode de evaluare
 - ❑ Măsuri de performanță
- Tipologie

Sisteme inteligente – SIS – Învățare automată

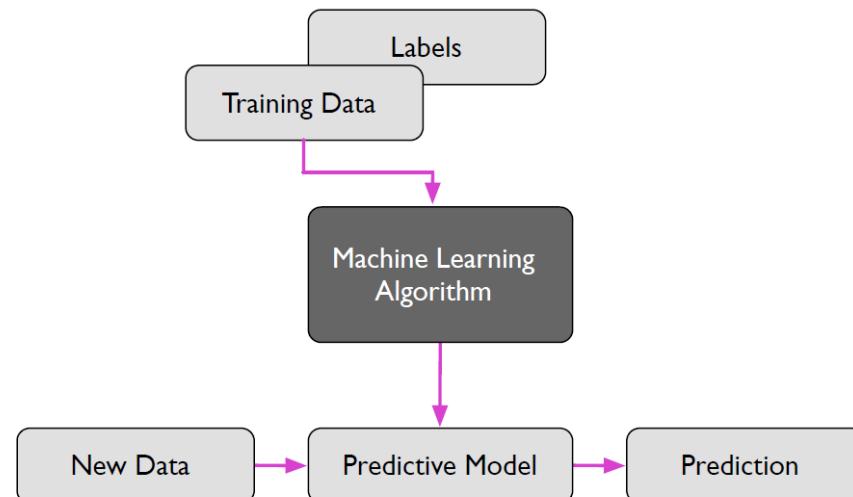
Învățare supervizată

- Scop
 - Furnizarea unei ieșiri corecte pentru o nouă intrare
- Definire
 - Se dă un set de date (exemple, instanțe, cazuri)
 - date de antrenament – sub forma unor perechi (atribute_data_i , ieșire_i), unde
 - $i = 1, N$ ($N = \text{nr datelor de antrenament}$)
 - $\text{atribute_data}_i = (\text{atr}_{i1}, \text{atr}_{i2}, \dots, \text{atr}_{im})$, $m = \text{nr atributelor (caracteristicilor, proprietăților) unei date}$
 - ieșire_i
 - o categorie dintr-o mulțime dată (predefinită) cu k elemente ($k = \text{nr de clase}$) → problemă de clasificare
 - un număr real → problemă de regresie
 - date de test - sub forma (atribute_data_i), $i = 1, n$ ($n = \text{nr datelor de test}$).
 - Să se determine
 - o funcție (necunoscută) care realizează corespondența atribut – ieșire pe datele de antrenament
 - ieșirea (clasa/valoarea) asociată unei date (noi) de test folosind funcția învățată pe datele de antrenament
 - Alte denumiri
 - Clasificare (regresie), învățare inductivă

Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

- Proces → 2 etape
 - Antrenarea
 - Învățarea, cu ajutorul unui algoritm, a modelului de predicție
 - Testarea
 - Testarea modelului folosind date de test noi (*unseen data*)



- Caracteristic
 - BD experimentală adnotată (pt. învățare)

Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

□ Tip de probleme

- regresie
 - Scop: predicția output-ului pentru un input nou
 - Output continuu (nr real)
 - Ex.: predicția ratei șomajului în funcție de produsul intern brut și rata inflației
- clasificare
 - Scop: clasificarea (etichetarea) unui nou input
 - Output discret (etichetă dintr-o mulțime predefinită)
 - Ex.: detectarea tumorilor maligne în imagini RMN

□ Exemple de probleme

- Recunoașterea scrisului de mână
- Recunoașterea pietonilor în imagini
- Previziunea vremii
- Detectia spam-urilor

Sisteme inteligente – SIS – Învățare automată

- Învățare supervizată
 - Terminologie – e.g. Problema predicției consumului de înghețată pe baza temperaturii de afară și a sumei de bani avută la dispoziție
 - Exemplu (example, observation, instance, record)
 - o observație a datelor care trebuie procesate
 - dacă datele de intrare sunt tabelare, un exemplu este asociat cu o linie din tabel
 - format din proprietăți a datelor care trebuie procesate (de intrare și de ieșire)
 - Caracteristică (feature, property, attribute)
 - Proprietate cunoscută a unui exemplu, folosită drept dată de intrare pentru algoritmul de ML (variabilele independente din modelul de predicție)
 - dacă datele de intrare sunt tabelare, un o proprietate are asociate valorile dintr-o coloană a tabelului (pentru toate exemplele)
 - E.g. Temperatura, banii
 - Valoare țintă (target or real value/label, ground-truth)
 - Proprietate a unui exemplu folosită ca variabilă dependentă
 - Cunoscută pentru exemplele de antrenament
 - Ne-cunoscută pentru exemplele de testare
 - E.g. Nr de inghetate
 - Valoare calculată (computed value/label)
 - Proprietate a unui exemplu estimată cu ajutorul algoritmului de ML
 - Se dorește a fi cât mai aproape de valoarea target

Exemplu	Temperatura	Banii	Nr de inghetate
Ex1	30	25	2
Ex2	5	100	0
Ex3	19	55	2
Ex4	35	75	4

Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

□ Calitatea învățării

■ Definire

- o măsură de performanță a algoritmului de ML
 - ex. acuratețea ($\text{Acc} = \text{nr de exemple corect clasificate} / \text{nr total de exemple}$)

- Posibile măsuri:

- Măsuri statistice
 - Eroarea de predicție
 - acuratețea
 - Precizia
 - Rapelul
 - Scorul F1

Sisteme inteligente – SIS – Învățare automată

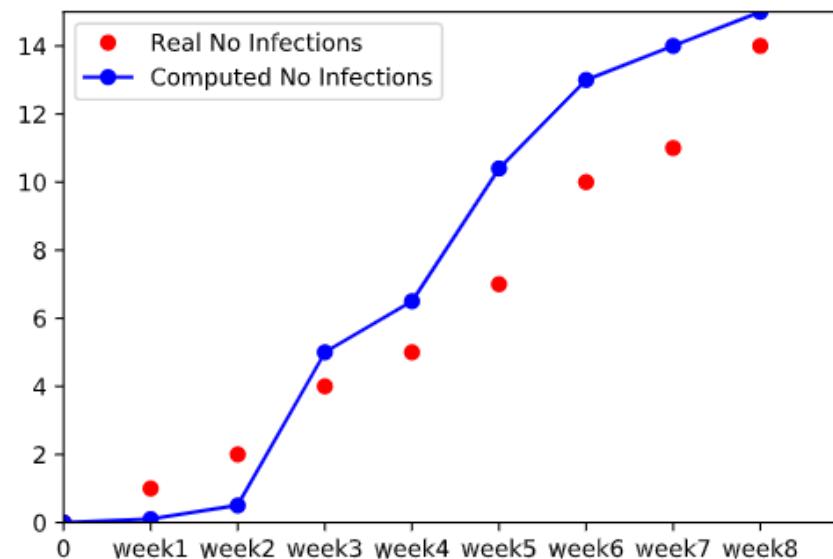
Învățare supervizată

- Calitatea învățării → Măsuri de performanță → Măsuri statistice
 - Eroarea de predicție
 - Suma diferențelor absolute între valorile reale și cele calculate
 - Suma pătratelor diferențelor între valorile reale și cele calculate

$$Err = \frac{1}{noSamples} \sum_{i=1}^{noSamples} abs(real_i - computed_i)$$

- Suma pătratelor diferențelor între valorile reale și cele calculate

$$Err = \sqrt{\frac{1}{noSamples} \sum_{i=1}^{noSamples} (real_i - computed_i)^2}$$



Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

- Calitatea învățării → Măsuri de performanță → Măsuri statistice

- Acuratețea

- Nr de exemple corect clasificate / nr total de exemple
 - Opusul erorii
 - Calculată pe
 - Setul de validare
 - Setul de test
 - Uneori
 - Analiză de text
 - Detectarea intrușilor într-o rețea
 - Analize financiare

este importantă doar o singură clasă (clasă pozitivă) → restul claselor sunt negative

- Precizia (P)

- nr. de exemple pozitive corect clasificate / nr. total de exemple clasificate ca pozitive
 - probabilitatea ca un exemplu clasificat pozitiv să fie relevant
 - $TP / (TP + FP)$

- Rapelul (R)

- nr. de exemple pozitive corect clasificate / nr. total de exemple pozitive
 - Probabilitatea ca un exemplu pozitiv să fie identificat corect de către clasificator
 - $TP / (TP + FN)$
 - Matrice de confuzie → rezultate reale vs. rezultate calculat

- Scorul F1

- Combină precizia și rapelul, facilitând compararea a 2 algoritmi
 - Media armonică a preciziei și rapelului
 - $2PR/(P+R)$

		Rezultate reale	
		Clasa pozitivă	Clasa(ele) negativă(e)
Rezultate calculate	Clasa pozitivă	<i>True positiv (TP)</i>	<i>False positiv (FP)</i>
	Clasa(ele) negativă(e)	<i>False negative (FN)</i>	<i>True negative (TN)</i>

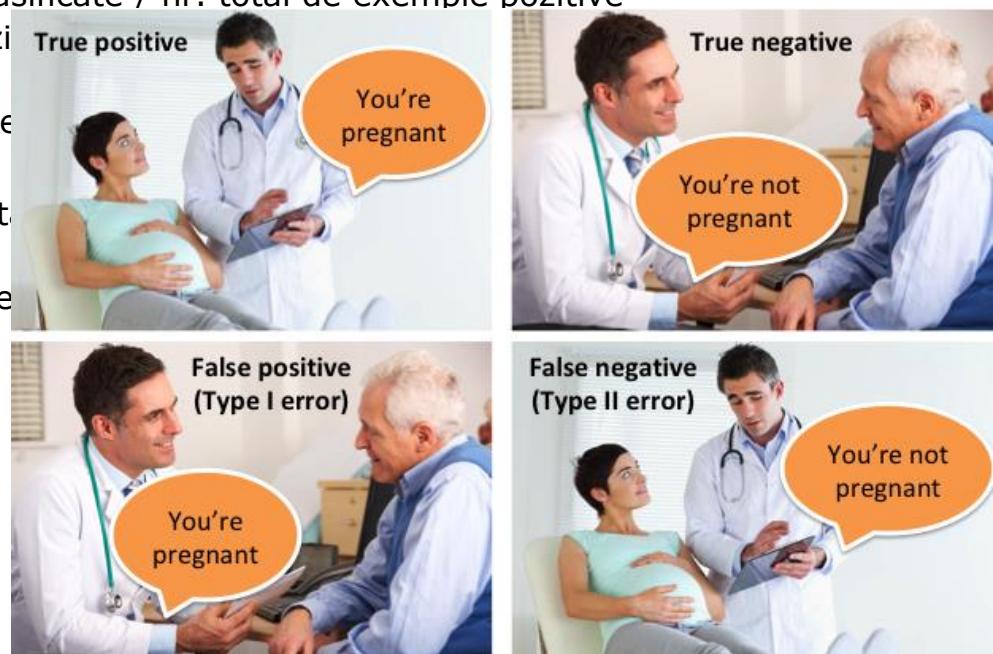
Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

- Calitatea învățării → Măsuri de performanță → Măsuri statistice

- Acuratețea
 - Nr de exemple corect clasificate / nr total de exemple
- Precizia (P)
 - nr. de exemple pozitive corect clasificate / nr. total de exemple clasificate ca pozitive
 - probabilitatea ca un exemplu clasificat pozitiv să fie relevant
 - $TP / (TP + FP)$
- Rapelul (R)
 - nr. de exemple pozitive corect clasificate / nr. total de exemple pozitive
 - Probabilitatea ca un exemplu pozitiv să fie clasificat pozitiv
 - $TP / (TP + FN)$
 - Matrice de confuzie → rezultate reale vs rezultate calculate
- Scorul F1
 - Combină precizia și rapelul, facilită compararea a 2 algoritmi
 - Media armonică a preciziei și rapelului
 - $2PR / (P+R)$

		Rezultate reale	
		Clasa pozitivă	Clasa(ele) negativă(e)
Rezultate calculate	Clasa pozitivă	True positiv (TP)	False positiv (FP)
	Clasa(ele) negativă(e)	False negative (FN)	True negative (TN)



Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

□ Calitatea învățării

■ Definire

- o măsură de performanță a algoritmului
 - ex. acuratețea ($\text{Acc} = \text{nr de exemple corect clasificate} / \text{nr total de exemple}$)

- Posibile măsuri:

- Măsuri statistice
 - acuratețea
 - Precizia
 - Rapelul
 - Scorul F1
 - Eficiența
 - În construirea modelului
 - În testarea modelului
 - Robustețea
 - Tratarea zgomotelor și a valorilor lipsă
 - Scalabilitatea
 - Eficiența gestionării seturilor mari de date
 - Interpretabilitatea
 - Modelului de clasificare
 - Proprietatea modelului de a fi compact
 - Scoruri

Sisteme inteligente – SIS – Învățare automată

Învățare supervizată

□ Calitatea învățării

■ Definire

- o măsură de performanță a algoritmului
 - ex. acuratețea ($\text{Acc} = \text{nr de exemple corect clasificate} / \text{nr total de exemple}$)
- calculată în
 - fază de antrenare
 - fază de testare

■ Metode de evaluare

- Seturi disjuncte de antrenare și testare
 - setul de antrenare poate fi împărțit în date de învățare și date de validare
 - setul de antrenare este folosit pentru estimarea parametrilor modelului (cei mai buni parametri obținuți pe validare vor fi folosiți pentru construcția modelului final)
 - pentru date numeroase
- Validare încrucișată cu mai multe (h) sub-seturi egale ale datelor (de antrenament)
 - separarea datelor de h ori în ($h-1$ sub-seturi pentru învățare și 1 sub-set pt validare)
 - dimensiunea unui sub-set = dimensiunea setului / h
 - performanța este dată de media pe cele h rulări (ex. $h = 5$ sau $h = 10$)
 - pentru date puține
- Leave-one-out cross-validation
 - similar validării încrucișate, dar $h = \text{nr de date} \rightarrow$ un sub-set conține un singur exemplu
 - pentru date foarte puține

■ Dificultăți

- Învățare pe deros (overfitting) \rightarrow performanță bună pe datele de antrenament, dar foarte slabă pe datele de test

Sisteme inteligente – SIS – Învățare automată

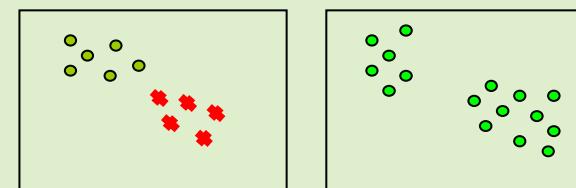
□ Învățare ne-supervizată

- Definire
- Exemple
- Proces
- Metode de evaluare și măsuri de performanță
- Tipologie

Sisteme inteligeante – SIS – Învățare automată

Învățare ne-supervizată

- Scop
 - Găsirea unui model sau a unei structuri utile a datelor
 - Împărțirea unor exemple **neetichetate** în submulțimi disjuncte (clusteri) astfel încât:
 - exemplele din același cluster sunt foarte similare
 - exemplele din clusteri diferiți sunt foarte diferite
- Definire
 - Se dă un set de date (exemple, instanțe, cazuri)
 - Date de antrenament sub forma **atribute_data_i**, unde
 - $i = 1, N$ (N = nr datelor de antrenament)
 - **atribute_data_i** = $(atr_{i1}, atr_{i2}, \dots, atr_{im})$, m – nr atributelor (caracteristicilor, proprietăților) unei date
 - Date de test sub forma (**atribute_data_i**), $i = 1, n$ (n = nr datelor de test)
 - Se determină
 - o funcție (necunoscută) care realizează gruparea datelor de antrenament în mai multe clase
 - Nr de clase poate fi pre-definit (k) sau necunoscut
 - Datele dintr-o clasă sunt asemănătoare
 - clasa asociată unei date (noi) de test folosind gruparea învățată pe datele de antrenament
 - Învățare supervizată vs. Învățare ne-supervizată
- Distanțe între 2 elemente p și $q \in R^m$
 - Euclideană $\rightarrow d(p,q) = \sqrt{\sum_{j=1,2,\dots,m} (p_j - q_j)^2}$
 - Manhattan $\rightarrow d(p,q) = \sum_{j=1,2,\dots,m} |p_j - q_j|$
 - Mahalanobis $\rightarrow d(p,q) = \sqrt{(p-q)'S^{-1}(p-q)}$, unde S este matricea de variație și covariație ($S = E[(p-E[p])(q-E[q])']$)
 - Produsul intern $\rightarrow d(p,q) = \sum_{j=1,2,\dots,m} p_j q_j$
 - Cosine $\rightarrow d(p,q) = \sum_{j=1,2,\dots,m} p_j q_j / (\sqrt{\sum_{j=1,2,\dots,m} p_j^2} * \sqrt{\sum_{j=1,2,\dots,m} q_j^2})$
 - Hamming \rightarrow numărul de diferențe între p și q
 - Levenshtein \rightarrow numărul minim de operații necesare pentru a-l transforma pe p în q
- Distanță vs. Similaritate
 - Distanță \rightarrow min
 - Similaritatea \rightarrow max



Sisteme inteligente – SIS – Învățare automată

Învățare ne-supervizată

- ❑ Alte denumiri
 - Clustering
- ❑ Procesul → 2 pași
 - Antrenarea → Învățarea (determinarea), cu ajutorul unui algoritm, a clusterilor existenți
 - Testarea → Plasarea unei noi date într-unul din clusterii identificați în etapa de antrenament
- ❑ Caracteristic
 - Datele nu sunt adnotate (etichetate)
- ❑ Tip de probleme
 - Identificarea unor grupuri (clusteri)
 - ❑ Analiza genelor
 - ❑ Procesarea imaginilor
 - ❑ Analiza rețelelor sociale
 - ❑ Segmentarea pieței
 - ❑ Analiza datelor astronomice
 - ❑ Clusteri de calculatoare
 - Reducerea dimensiunii
 - Identificarea unor cauze (explicații) ale datelor
 - Modelarea densității datelor
- ❑ Exemple de probleme
 - Gruparea genelor
 - Studii de piață pentru gruparea clienților (segmentarea pieței)
 - news.google.com

Sisteme inteligente – SIS – Învățare automată

Învățare ne-supervizată

□ Calitatea învățării (validarea clusterizări):

- Criterii interne → Similaritate ridicată în interiorul unui cluster și similaritate redusă între clusteri
 - Distanța în interiorul clusterului
 - Distanța între clusteri
 - Indexul Davies-Bouldin
 - Indexul Dunn
- Criterii externe → Folosirea unor benchmark-uri formate din date pre-grupate
 - Compararea cu date cunoscute – în practică este imposibil
 - Precizia
 - Rapelul
 - F-measure

Sisteme inteligețe – SIS – Învățare automată

Învățare ne-supervizată

□ Calitatea învățării → Criterii interne

■ Distanța în interiorul clusterului c_j care conține n_j instanțe

- Distanța medie între instanțe (average distance) $D_a(c_j) = \sum_{x_{i1}, x_{i2} \in c_j} ||x_{i1} - x_{i2}|| / (n_j(n_j-1))$
- Distanța între cei mai apropiati vecini $D_{nn}(c_j) = \sum_{x_{i1} \in c_j} \min_{x_{i2} \notin c_j} ||x_{i1} - x_{i2}|| / n_j$
- Distanța între centroizi $D_c(c_j) = \sum_{x_i \in c_j} ||x_i - \mu_j|| / n_j$, unde $\mu_j = 1/n_j \sum_{x_i \in c_j} x_i$

■ Distanța între 2 clusteri c_{j1} și c_{j2}

- Legătură simplă $d_s(c_{j1}, c_{j2}) = \min_{x_{i1} \in c_{j1}, x_{i2} \in c_{j2}} \{||x_{i1} - x_{i2}||\}$
- Legătură completă $d_{co}(c_{j1}, c_{j2}) = \max_{x_{i1} \in c_{j1}, x_{i2} \in c_{j2}} \{||x_{i1} - x_{i2}||\}$
- Legătură medie $d_m(c_{j1}, c_{j2}) = \sum_{x_{i1} \in c_{j1}, x_{i2} \in c_{j2}} \{||x_{i1} - x_{i2}||\} / (n_{j1} * n_{j2})$
- Legătură între centroizi $d_{ce}(c_{j1}, c_{j2}) = ||\mu_{j1} - \mu_{j2}||$

■ Indexul Davies-Bouldin → min → clusteri compacti

- $DB = 1/nc * \sum_{i=1,2,\dots,nc} \max_{j=1, 2, \dots, nc, j \neq i} ((\sigma_i + \sigma_j)/d(\mu_i, \mu_j))$, unde:
 - nc – numărul de clusteri
 - μ_i – centroidul clusterului i
 - σ_i – media distanțelor între elementele din clusterul i și centroidul μ_i
 - $d(\mu_i, \mu_j)$ – distanța între centroidul μ_i și centroidul μ_j

■ Indexul Dunn

- Identifică clusterii denși și bine separați
- $D = d_{min}/d_{max}$, unde:
 - d_{min} – distanța minimă între 2 obiecte din clusteri diferiți – distanță intra-cluster
 - d_{max} – distanța maximă între 2 obiecte din același cluster – distanță inter-cluster

Sisteme inteligente – SIS – Învățare automată

Învățare ne-supervizată

□ Tipologie

■ După modul de formare al clusterilor

□ Ierarhic

- se crează un arbore taxonomic (dendogramă)
 - crearea clusterilor → recursiv
 - nu se cunoaște k (nr de clusteri)
- aglomerativ (de jos în sus) → clusteri mici spre clusteri mari
- diviziv (de sus în jos) → clusteri mari spre clusteri mici
- Ex. Clustering ierarhic aglomrativ

□ Ne-ierarhic

- Partițional → se determină o împărțire a datelor → toți clusterii deodată
- Optimizează o funcție obiectiv definită local (doar pe anumite atrbute) sau global (pe toate atrbutele) care poate fi:
 - Pătratul erorii – suma patratelor distanțelor între date și centroizii clusterilor → min (ex. K-means)
 - Bazată pe grafuri (ex. Clusterizare bazată pe arborele minim de acoperire)
 - Pe modele probabilistice (ex. Identificarea distribuției datelor → Maximizarea așteptărilor)
 - Pe cel mai apropiat vecin

- Necesită fixarea apriori a lui k → fixarea clusterilor inițiali
 - Algoritmii se rulează de mai multe ori cu diferenți parametri și se alege versiunea cea mai eficientă
- Ex. K-means, ACO

□ bazat pe densitatea datelor

- Densitatea și conectivitatea datelor
 - Formarea clusterilor de bază pe densitatea datelor într-o anumită regiune
 - Formarea clusterilor de bază pe conectivitatea datelor dintr-o anumită regiune
- Funcția de densitate a datelor
 - Se încercă modelarea legii de distribuție a datelor
- Avantaj:
 - Modelarea unor clusteri de orice formă

□ Bazat pe un grid

- Nu e chiar o metodă nouă de lucru
 - Poate fi ierarhic, partițional sau bazat pe densitate
- Pp segmentarea spațiului de date în zone regulate
- Obiectele se plasează pe un grid multi-dimensional
- Ex. ACO

Sisteme inteligețe – SIS – Învățare automată

Învățare ne-supervizată

□ Tipologie

- După modul de lucru al algoritmului
 - Aglomerativ
 1. Fiecare instanță formează inițial un cluster
 2. Se calculează distanțele între oricare 2 clusteri
 3. Se reunesc cei mai apropiati 2 clusteri
 4. Se repetă pașii 2 și 3 până se ajunge la un singur cluster sau la un alt criteriu de stop
 - Diviziv
 1. Se stabilește numărul de clusteri (k)
 2. Se inițializează centrii fiecărui cluster
 3. Se determină o împărțire a datelor
 4. Se recalculează centrii clusterilor
 5. Se repetă pasul 3 și 4 până partitioarea nu se mai schimbă (algoritmul a convergat)
- După atributele considerate
 - Monotetic – atributele se consideră pe rând
 - Politetic – atributele se consideră simultan
- După tipul de apartenență al datelor la clusteri
 - Clustering exact (*hard clustering*)
 - Asociază fiecarei intrări x_i o etichetă (clasă) c_j
 - Clustering fuzzy
 - Asociază fiecarei intrări x_i un grad (probabilitate) de apartenență f_{ij} la o anumită clasă $c_j \rightarrow$ o instanță x_i poate apartine mai multor clusteri

Sisteme inteligente – SIS – Învățare automată

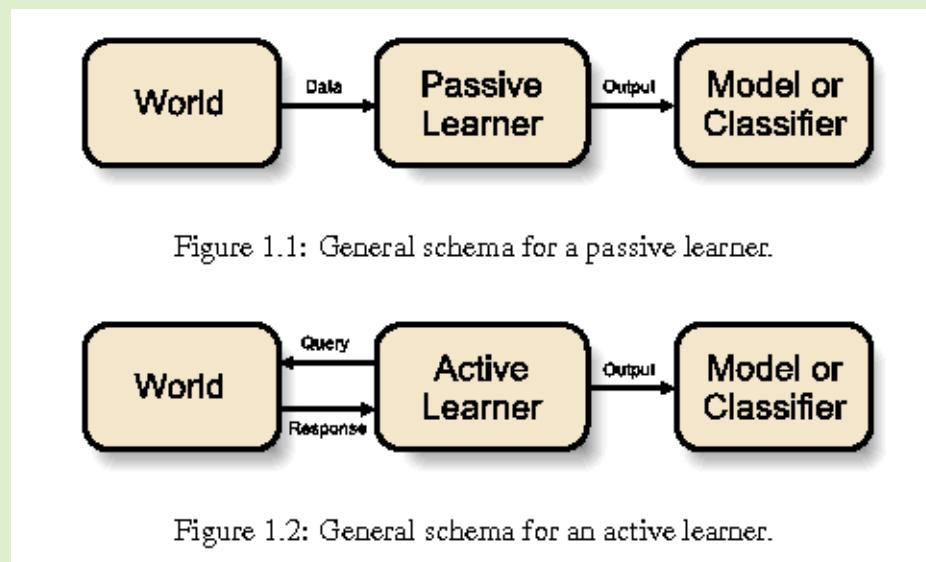
□ Tipologie

- În funcție de experiența acumulată în timpul învățării
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - **SI cu învățare activă**
 - SI cu învățare cu întărire
- În funcție de modelul învățat (algoritmul de învățare)
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Algoritmi evolutivi
 - Mașini cu suport vectorial
 - Modele Markov ascunse

Sisteme inteligente – SIS – Învățare automată

□ Învățare activă

- Algoritmul de învățare poate primi informații suplimentare în timpul învățării pentru a-și îmbunătăți performanța
 - ▣ Ex. pe care din datele de antrenament este mai ușor să se învețe modelul de decizie



Sisteme inteligente – SIS – Învățare automată

□ Tipologie

- În funcție de experiența acumulată în timpul învățării
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - SI cu învățare activă
 - SI cu învățare cu întărire
- În funcție de modelul învățat (algoritmul de învățare)
 - **Metoda celor mai mici pătrate**
 - Metoda gradient descent
 - Algoritmi evolutivi
 - Logisitic regression
 - kNN
 - Arbori de decizie
 - Mașini cu suport vectorial
 - Rețele neuronale artificiale
 - Programare genetică
 - Modele Markov ascunse



Recapitulare

❑ Sisteme care învață singure (SIS)

■ Instruire (învățare) automata (Machine Learning - ML)

- ❑ Învățare supervizată → datele de antrenament sunt deja etichetate cu elemente din E, iar datele de test trebuie etichetate cu una dintre etichetele din E pe baza unui model (învățat pe datele de antrenament) care face corespondența date-etichete
- ❑ Învățare nesupervizată → datele de antrenament NU sunt etichetate, trebuie învățat un model de etichetare, iar apoi datele de test trebuie etichetate cu una dintre etichetele identificate de model

■ Sisteme

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Metoda celor mai mici patrate, Gradient Descent, Logistic regression
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor –

Materiale de citit și legături utile

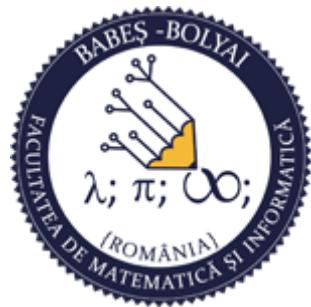
- Capitolul VI (19) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 8 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 12 și 13 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Capitolul 4 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

□ Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:

- Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
- Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan
- Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

■ Sisteme care învață singure

- Metoda celor mai mici patrate, Gradient Descent, Logistic regression
- Arbori de decizie
- Rețele neuronale artificiale
- Mașini cu suport vectorial
- Algoritmi evolutivi

■ Sisteme bazate pe reguli

■ Sisteme hibride

A. Rezolvarea problemelor prin căutare

■ Definirea problemelor de căutare

■ Strategii de căutare

- Strategii de căutare neinformate
- Strategii de căutare informate
- Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
- Strategii de căutare adversială

Materiale de citit și legături utile

- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 și 11 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Conținut

❑ Sisteme inteligente

■ Sisteme care învață singure (SIS)

❑ Instruire (învățare) automata (Machine Learning - ML)

- Problematică
- Proiectarea unui sistem de învățare automată
- Tipologie
 - Învățare supervizată
 - Învățare nesupervizată
 - Învățare cu întărire
 - Teoria învățării

❑ Exemple de sisteme

Conținut

■ Sisteme care învață singure (SIS)

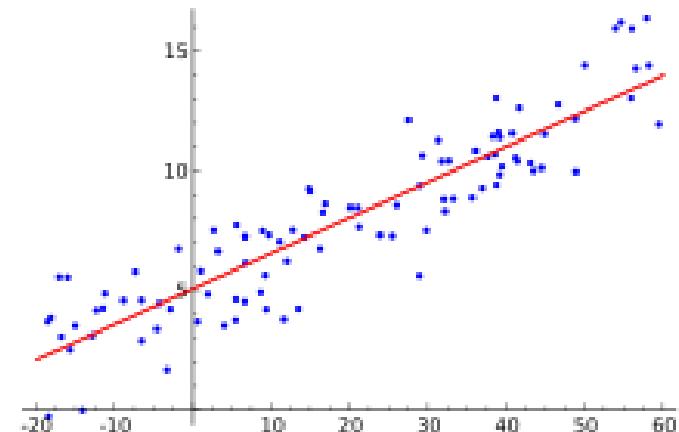


- "Field of study that gives computers the ability to learn without being explicitly programmed." -- Arthur Samuels (1959)

- Invățare
 - Supervizată
 - Nesupervizată
 - Reinforcement

Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
 - Presupunem cazul unei probleme de regresie
 - Date de intrare $x \in \mathbb{R}^d$
 - Date de ieșire $y \in \mathbb{R}$
 - Se cere un model **liniar** f care transformă x în y
 - $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$
 - Învățare supervizată



Sisteme inteligente – SIS – Învățare automată

□ Metoda celor mai mici pătrate

■ Presupunem cazul unei probleme de regresie

- Date de intrare $x^i \in R^d$, $i=1,n$
- Date de ieșire $y^i \in R$
- Se cere un model **liniar** f care transformă orice x^i în y^i , $i=1,n$
- $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$

- Se poate defini o funcție de cost
- $\text{Loss} = \sum_{i=1,n} (y^i - f(x^i))^2$ -- minimizată → valorile optime ale lui β

$$\mathbf{x} = (1, \mathbf{x}) = (1, x_1, x_2, \dots, x_d)^T \in R^{d+1}$$

$$\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \dots, \beta_d)^T \in R^{d+1}$$

$$f(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\beta}$$

$$\text{Loss}(\boldsymbol{\beta}) = \| \mathbf{y} - \mathbf{X} \boldsymbol{\beta} \|^2$$

$$\mathbf{X} = \begin{matrix} 1 & x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,d} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{matrix}$$

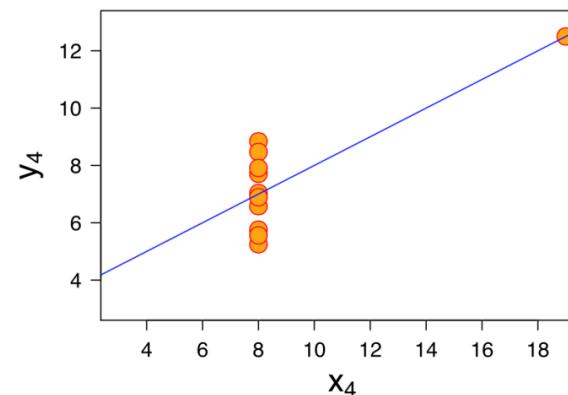
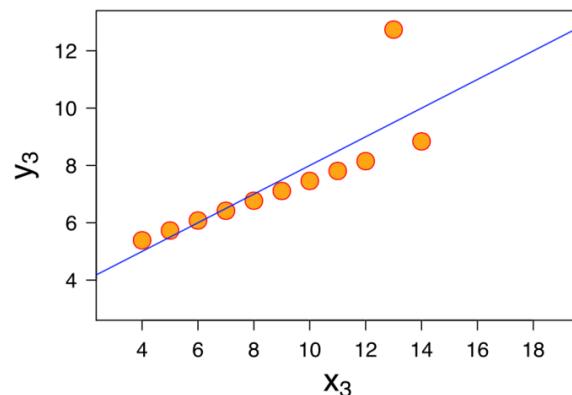
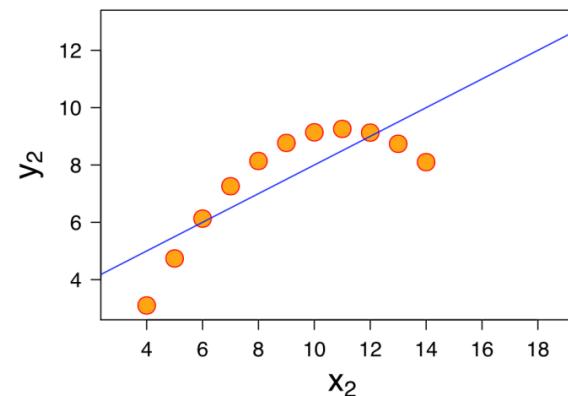
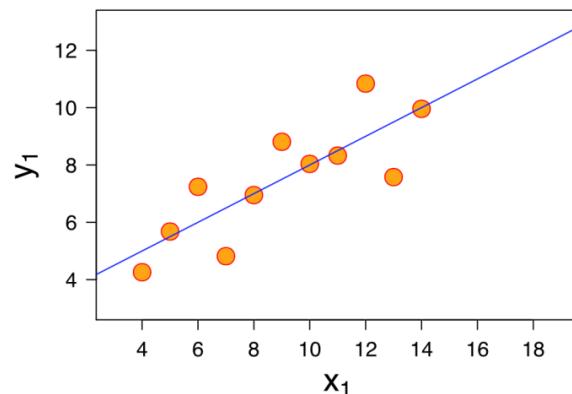
$$1 \ x_{n,1} \ x_{n,2} \ x_{n,3} \ \dots \ x_{n,d}$$

Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
 - Presupunem cazul unei probleme de regresie
 - Date de intrare $x^i \in R^d$, $i=1,n$
 - Date de ieșire $y^i \in R$
 - Se cere un model **liniar** f care transformă orice x^i în y^i , $i=1,n$
 - $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$
 - Se poate defini o funcție de cost
 - Loss = $\sum_{i=1,n} (y^i - f(x^i))^2$ -- minimizată → valorile optime ale lui β
 - Derivarea loss-ului după β : $\beta = (X^T X)^{-1} X^T y$
 - Dacă $d = 1$, $\beta_1 = \text{cov}(x,y)/\text{var}(x)$, $\beta_0 = y - \beta_1 x$

Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
 - Anscombe Quartet

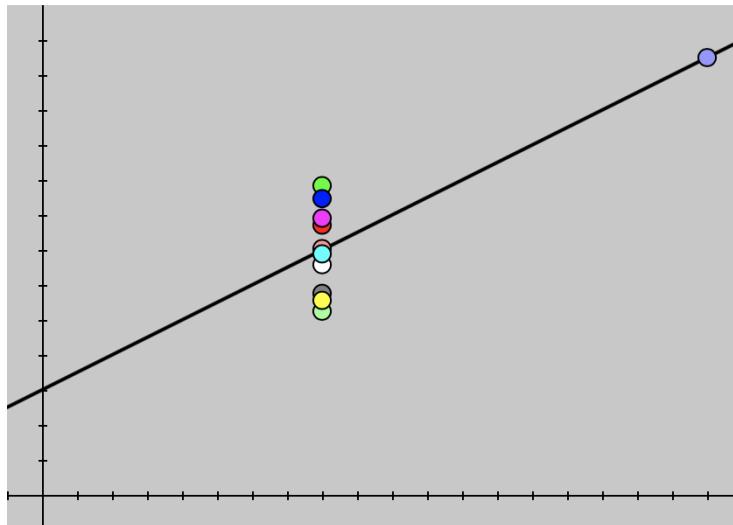


Ref. imagine

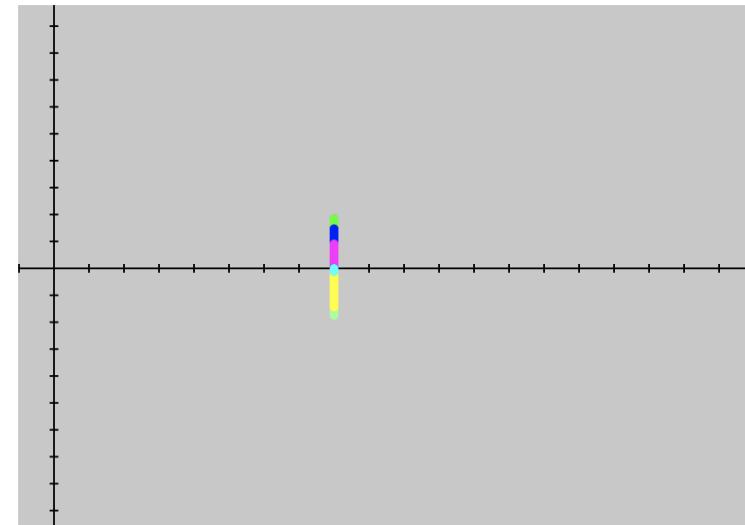
Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
 - Residual plot

Scatter plot



Residual plot

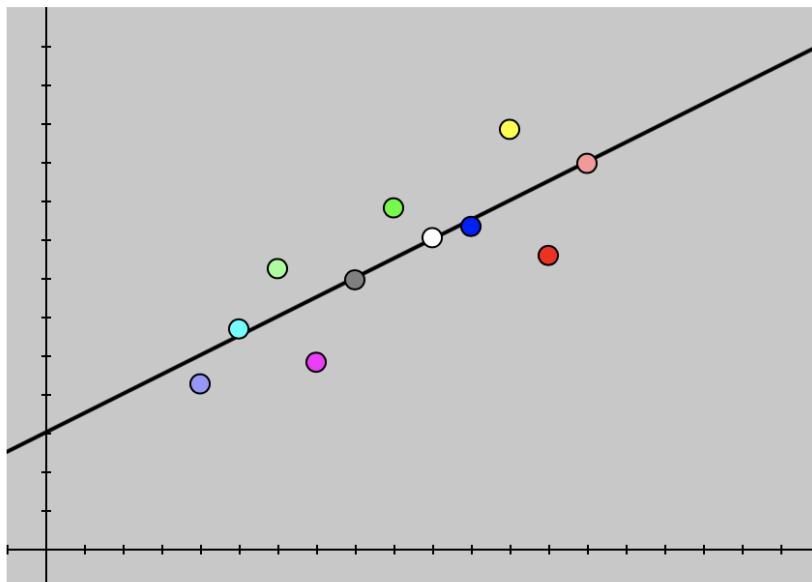


[Ref. imagine](#)

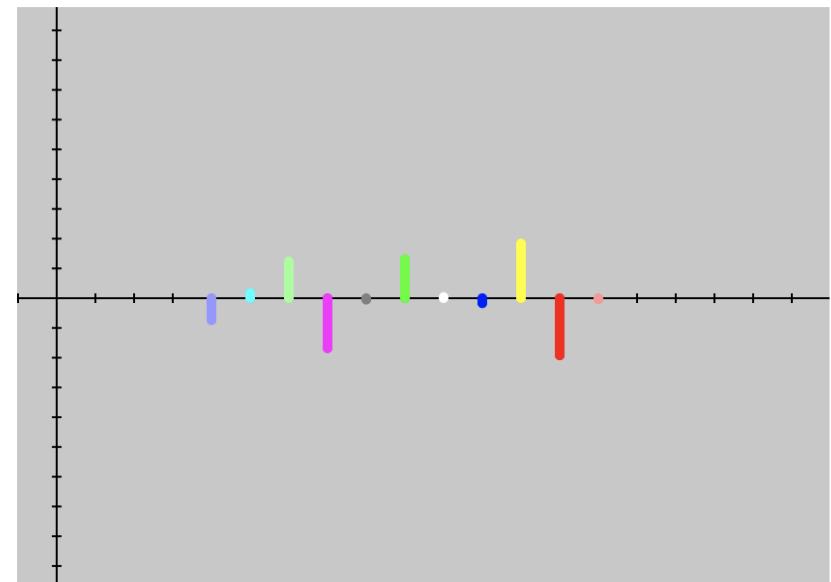
Sisteme inteligente – SIS – Învățare automată

- Metoda celor mai mici pătrate
 - Residual plot

Scatter plot



Residual plot

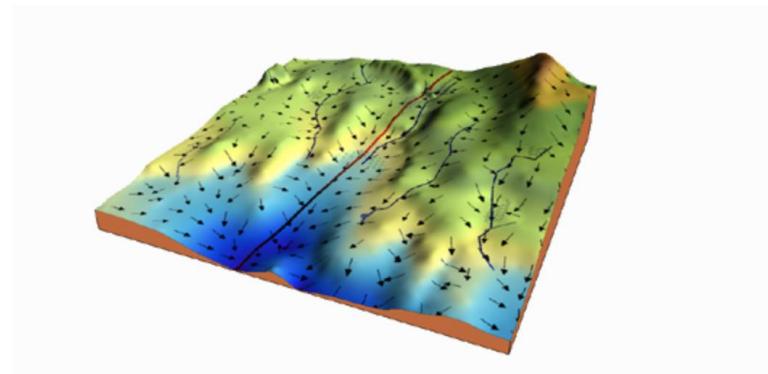


[Ref. imagine](#)

Sisteme inteligente – SIS – Învățare automată

□ Metoda *gradient descent*

- Presupunem cazul unei probleme de regresie
 - Date de intrare $x \in \mathbb{R}^d$
 - Date de ieșire $y \in \mathbb{R}$



- Se cere un model **liniar** f care transformă x în y
- $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$
- Învățare supervizată

[Ref. imagine](#)

Sisteme inteligente – SIS – Învățare automată

□ Metoda *gradient descent*

■ Modelarea coeficienților β :

- la iterația 0: valori random (sau 0)
- la iterația $t + 1$ ($t = 0, 1, 2, \dots$)

$$\beta_k(t+1) = \beta_k(t) - \text{learning_rate} * \text{error}(t) * x_k, k=1,2,\dots,d$$

$$\beta_0(t+1) = \beta_0(t) - \text{learning_rate} * \text{error}(t)$$

- Unde
 - $\text{error}(t) = \text{computed} - \text{realOutput}$
 - $\text{error}(t) = \beta_0(t) + \beta_1(t)*x_1 + \beta_2(t)*x_2 + \dots + \beta_d(t)*x_d - y$

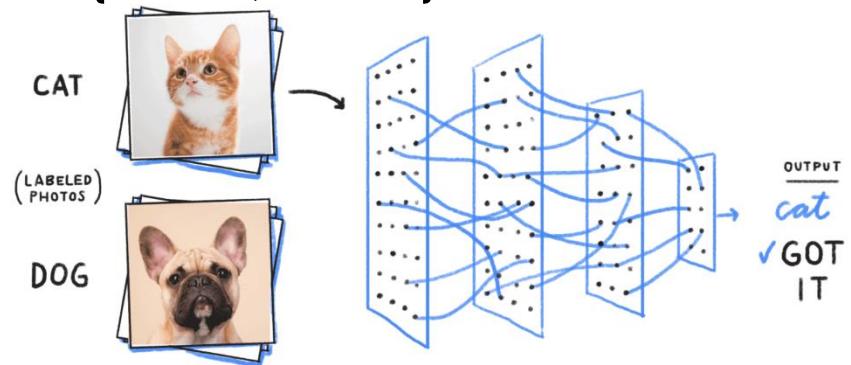
Sisteme inteligente – SIS – Învățare automată

❑ Metoda *gradient descent* – versiuni:

- Stochastic GD
 - ❑ Eroarea se calculează pentru fiecare exemplu de antrenament
 - ❑ Modelul se updatează pentru fiecare exemplu de antrenament (*online learning*)
- Batch GD
 - ❑ Eroarea se calculează pentru fiecare exemplu de antrenament
 - ❑ Modelul se updatează după ce toate exemplele de antrenament au fost evaluate (la finalul unei epoci)
- Mini-batch GD
 - ❑ Combinare a precedentelor două
 - ❑ Setul de date se împarte în mai multe părți (mini-batch-uri)
 - ❑ Eroarea se calculează pentru fiecare exemplu de antrenament dintr-un mini-batch
 - ❑ Modelul se updatează pentru fiecare exemplu de antrenament dintr-un mini-batch

Sisteme inteligente – SIS – Învățare automată

- Regresie Logistică (clasificare)
 - Presupunem cazul unei probleme de clasificare
 - Date de intrare $x^i \in \mathbb{R}^d$, $i=1,n$
 - Date de ieșire $y^i \in \{0,1\}$ sau {label1, label2}
 - Se cere un model **liniar** f care separă orice x^i în 2 clase (etichetate cu 0 și 1)
 - $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_d x_d$
 - Invățare supervizată



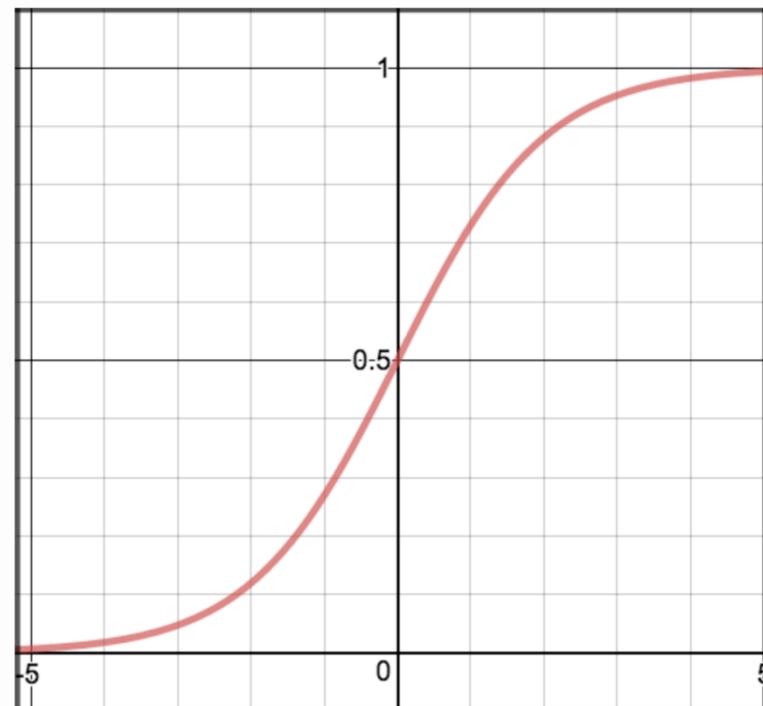
Sisteme inteligente – SIS – Învățare automată

- Regresie Logistică (clasificare)
 - Mapează datele într-un set discret de clase (label-uri)
 - Tipuri:
 - Binar (Pass/Fail, True/False)
 - Multi (Cat,Dog,Panda)
 - Ordinal (Low,Medium,High)
 - Folosește funcția sigmoid pentru a decide clasa de apartenență
 - Putem folosi gradient descent pentru minimizarea erorii

Sisteme inteligente – SIS – Învățare automată

- Regresie Logistică – sigmoid:
 - Mapează orice numar real in intervalul (0,1)

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



[Ref. imagine](#)

Sisteme inteligente – SIS – Învățare automată

□ Regresie Logistica

■ Modelarea coeficienților β :

- la iterația 0: valori random (sau 0)
- la iterația $t + 1$ ($t = 0, 1, 2, \dots$)

$$\beta_k(t+1) = \beta_k(t) - \text{learning_rate} * \text{error}(t) * x_k, k=1,2,\dots,d$$

$$\beta_0(t+1) = \beta_0(t) - \text{learning_rate} * \text{error}(t)$$

- Unde
 - $\text{error}(t) = \text{Sigmoid}(\text{computed}) - \text{realOutput}$
 - $\text{error}(t) = \text{Sigmoid}(\beta_0(t) + \beta_1(t)*x_1 + \beta_2(t)*x_2 + \dots + \beta_d(t)*x_d) - y$

■ Clasificarea rezultatelor

- $(0,1) \rightarrow [\text{label}_0, \text{label}_1, \dots \text{label}_n]$

Recapitulare



□ Sisteme care învață singure (SIS)

■ Instruire (învățare) automata (Machine Learning - ML)

- Învățare supervizată → datele de antrenament sunt deja etichetate cu elemente din E, iar datele de test trebuie etichetate cu una dintre etichetele din E pe baza unui model (învățat pe datele de antrenament) care face corespondență date-etichete
- Învățare nesupervizată → datele de antrenament NU sunt etichetate, trebuie învățat un model de etichetare, iar apoi datele de test trebuie etichetate cu una dintre etichetele identificate de model



Recapitulare

□ Sisteme care învață singure (SIS)

■ Metoda celor mai mici patrate

- Supervizată
- Output continu (vânzări, preț, etc.)
- Panta constantă

■ Gradient descent

- Supervizată
- Output continu (vânzări, preț, etc.)
- Optimizare

■ Regresie Logistică

- CLASIFICARE
- Supervizată
- Output – label-uri (clase) – set discret
- Folosește Gradient descent

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

C. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

Cursul următor – Materiale de citit și legături utile

- Capitolul VI (19) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 8 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 12 și 13 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Capitolul 4 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

□ Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:

- Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
- Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan
- Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop

INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure
– kNN și programare genetică –

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

C. Sisteme inteligente

■ Sisteme care învață singure

- Arbori de decizie
- Rețele neuronale artificiale
- kNN
- Algoritmi evolutivi
- **Mașini cu suport vectorial**

■ Sisteme bazate pe reguli

■ Sisteme hibride

B. Rezolvarea problemelor prin căutare

■ Definirea problemelor de căutare

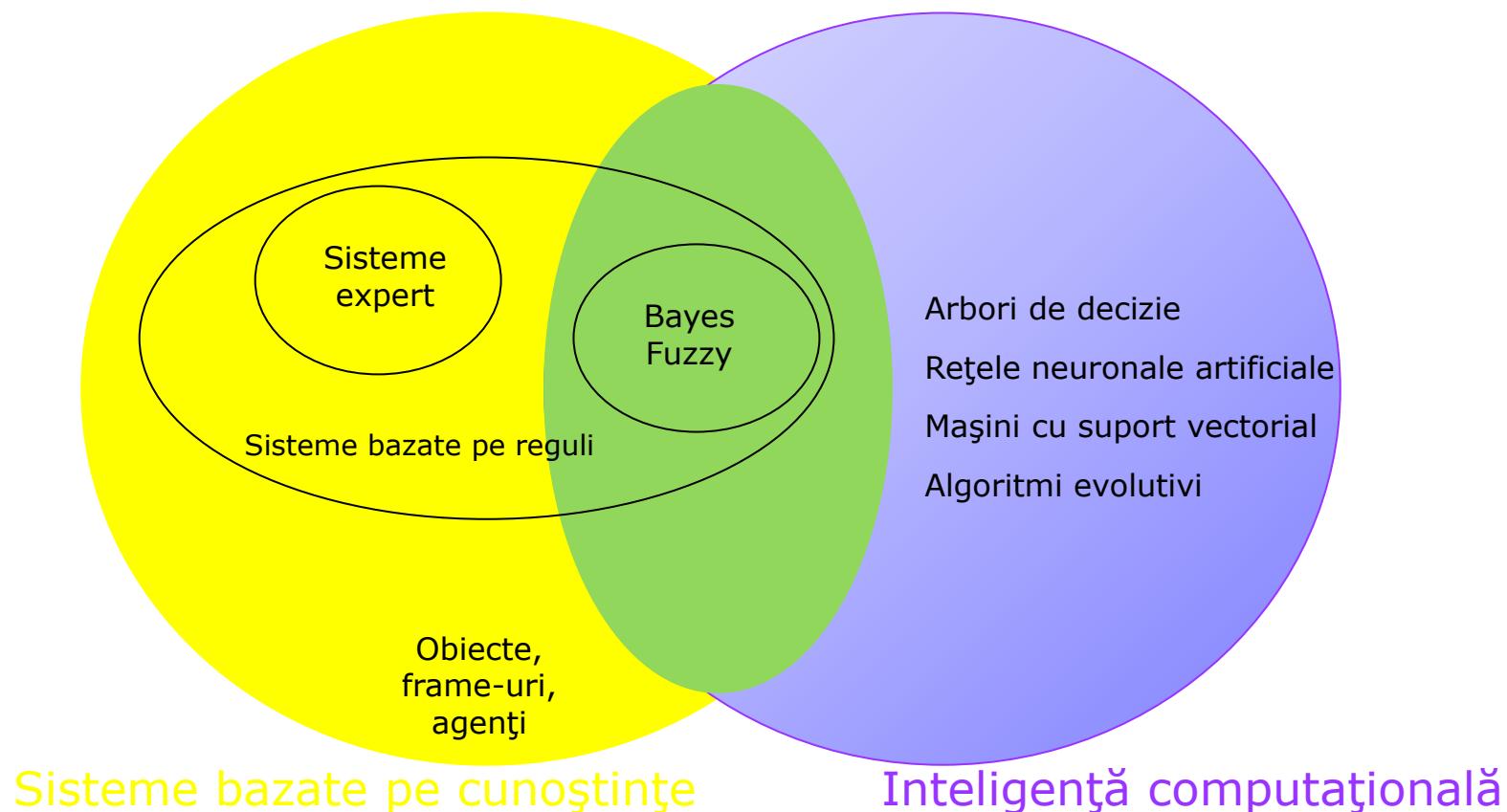
■ Strategii de căutare

- Strategii de căutare neinformate
- Strategii de căutare informate
- Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
- Strategii de căutare adversială

Materiale de citit și legături utile

- capitolul 15 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 și 11 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Sisteme inteligente



Sisteme inteligente – SIS – Învățare automată

□ Tipologie

- În funcție de experiența acumulată în timpul învățării:
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - SI cu învățare activă
 - SI cu învățare cu întărire

- În funcție de modelul învățat (algoritmul de învățare):
 - Rețele neuronale artificiale
 - **Mașini cu suport vectorial (MSV)**
 - Algoritmi evolutivi
 - kNN
 - Arbori de decizie
 - Modele Markov ascunse

Materiale de citit și legături utile

- capitolul VI (18) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 10 și 11 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- capitolul 3 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Sisteme inteligente – SIS – Învățare automată

□ Tipologie

- În funcție de experiența acumulată în timpul învățării:
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - SI cu învățare activă
 - SI cu învățare cu întărire

- În funcție de modelul învățat (algoritmul de învățare):
 - Arbori de decizie
 - Rețele neuronale artificiale
 - **Mașini cu suport vectorial (MSV)**
 - Algoritmi evolutivi
 - Modele Markov ascunse

Sisteme inteligente – SIS – MSV

□ Mașini cu suport vectorial (MSV)

- Definire
- Tipuri de probleme rezolvabile
- Avantaje
- Dificultăți
- Tool-uri

Sisteme inteligente – SIS – MSV

□ Definire

- Dezvoltate de Vapnik în 1970
- Popularizate după 1992
- Clasificatori liniari care identifică un hiperplan de separare a clasei pozitive de clasa negativă
- Au o fundamentare teoretică foarte riguroasă
- Funcționează foarte bine pentru date de volum mare (analiza textelor, analiza imaginilor)

■ Reamintim

- Problemă de învățare supervizată în care avem un set de date de forma:
 - (x^d, t^d) , cu:
 - $x^d \in \mathbb{R}^m \rightarrow x^d = (x_1^d, x_2^d, \dots, x_m^d)$
 - $t^d \in \mathbb{R} \rightarrow t^d \in \{1, -1\}$, 1 → clasă pozitivă, -1 → clasă negativă
 - cu $d = 1, 2, \dots, n, n+1, n+2, \dots, N$
- Primele n date (se cunosc x^d și t^d) vor fi folosite drept bază de antrenament a MSV
- Ultimele $N-n$ date (se cunosc doar x^d , fără t^d) vor fi folosite drept bază de testare a MSV

Sisteme inteligente – SIS – MSV

□ Definire

- MSV găsește o funcție liniară de forma $f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$, (\mathbf{w} -vector pondere) a.î.

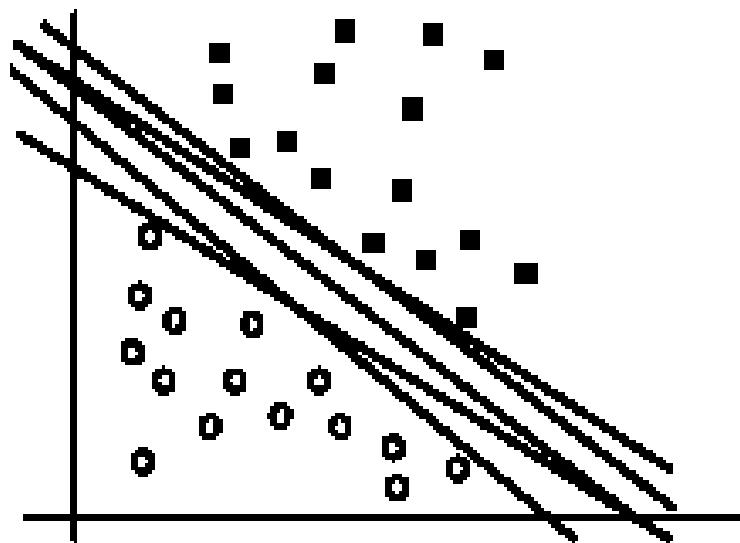
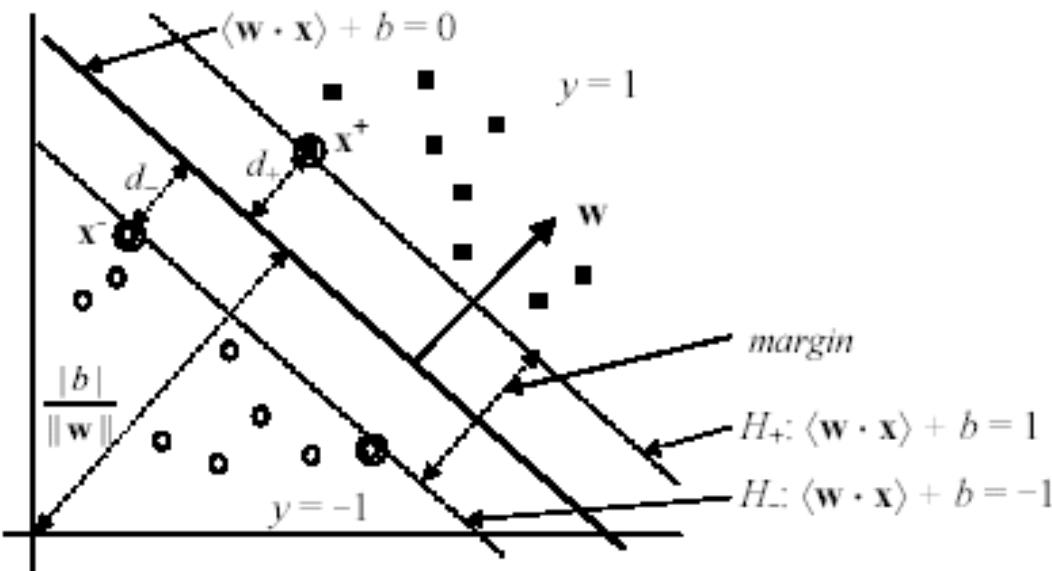
$$y_i = \begin{cases} 1 & \text{if } \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1 & \text{if } \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases}$$

- $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0 \rightarrow$ hiperplanul de decizie care separă cele 2 clase

Sisteme inteligente – SIS – MSV

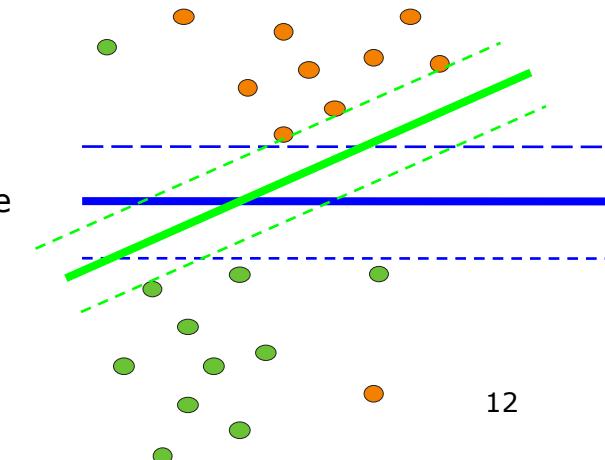
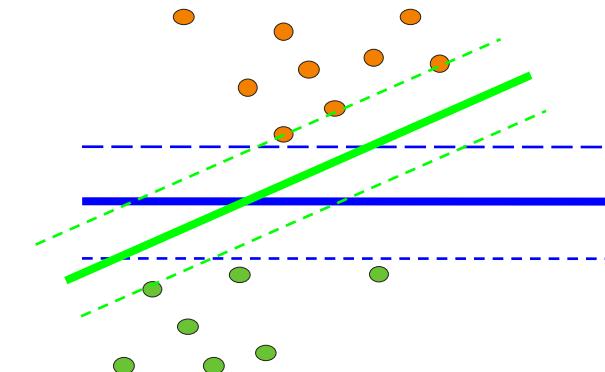
□ Definire

- Pot exista mai multe hiperplane
 - Care este cel mai bun hiperplan?
- MSV caută hiperplanul cu cea mai largă margine (cel care micșorează eroarea de generalizare)
 - Algoritmul SMO (*Sequential minimal optimization*)



Sisteme inteligente – SIS – MSV

- Tipuri de probleme rezolvabile
 - Probleme de clasificare → Cazuri de date
 - Liniar separabile
 - Separabile
 - Eroarea = 0
 - Ne-separabile
 - Se relaxează constrângerile → se permit unele erori
 - C – coeficient de penalizare

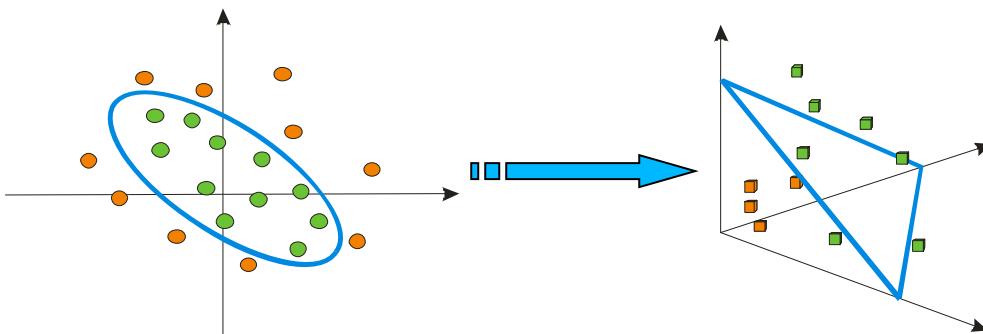


Sisteme inteligente – SIS – MSV

Cazuri de date

■ Non-liniar separabile

- Spațiul de intrare se transformă într-un spațiu cu mai multe dimensiuni (*feature space*), cu ajutorul unei funcții kernel, unde datele devin liniar separabile



■ Kernele posibile

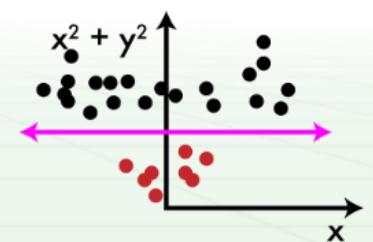
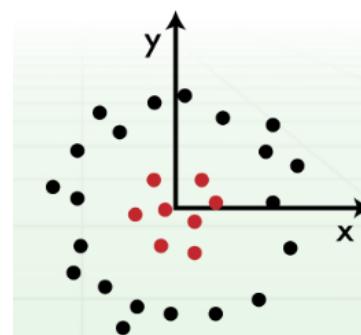
□ Clasice

- Polynomial kernel: $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^d$
- RBF kernel: $K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\sigma |\mathbf{x}_1 - \mathbf{x}_2|^2)$

□ Kernele multiple

- Liniare: $K(\mathbf{x}_1, \mathbf{x}_2) = \sum w_i K_i$
- Ne-liniare

- Fără coeficienți: $K(\mathbf{x}_1, \mathbf{x}_2) = K_1 + K_2 * \exp(K_3)$
- Cu coeficienți: $K(\mathbf{x}_1, \mathbf{x}_2) = K_1 + c_1 * K_2 * \exp(c_2 + K_3)$



Sisteme inteligente – SIS – MSV

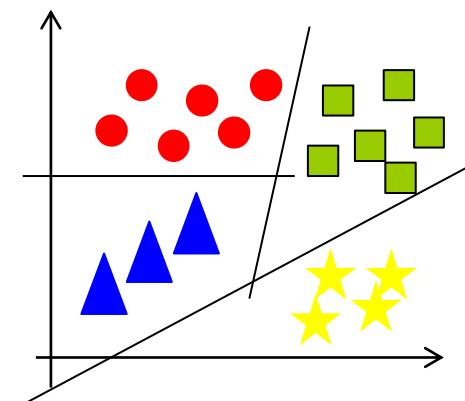
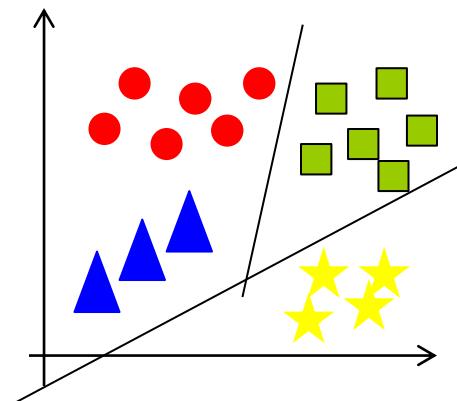
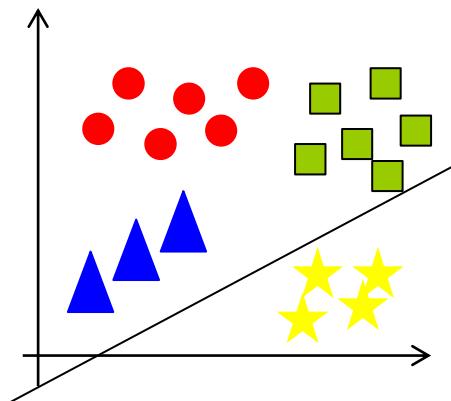
□ Configurarea MSV

■ Parametrii unei MSV

- Coeficientul de penalizare C
 - C – mic → convergență lentă
 - C – mare → convergență rapidă
- Parametrii funcției kernel (care kernel și cu ce parametri)
 - Dacă m (nr de atribute) este mult mai mare decât n (nr de instanțe)
 - MSV cu kernel liniar (MSV fără kernel) $\rightarrow K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = \mathbf{x}^{d1} \cdot \mathbf{x}^{d2}$
 - Dacă m (nr de atribute) este mare, iar n (nr de instanțe) este mediu
 - MSV cu kernel Gaussian $K(\mathbf{x}^{d1}, \mathbf{x}^{d2}) = \exp(-||\mathbf{x}^{d1} - \mathbf{x}^{d2}||^2 / 2\sigma^2)$
 - σ – dispersia datelor de antrenament
 - Atributele instanțelor trebuie normalize (scalate la $(0,1)$)
 - m (nr de atribute) este mic, iar n (nr de instanțe) este mare
 - Se adaugă noi atribute, iar apoi
 - MSV cu kernel liniar

Sisteme inteligente – SIS – MSV

- MSV pentru probleme de clasificare supervizate cu mai mult de 2 clase
 - Una vs. restul (one vs. all)



Sisteme inteligente – SIS – MSV

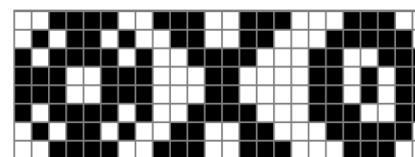
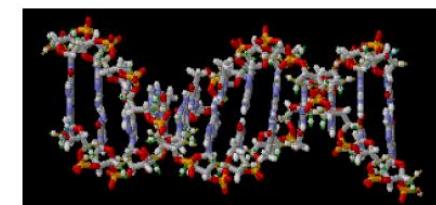
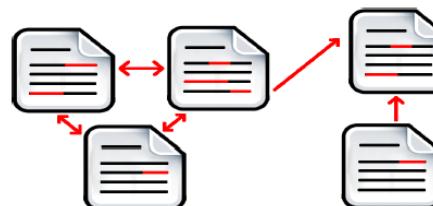
□ MSV structurate

- Învățare automată

- Normală $f: \mathcal{X} \rightarrow \mathbb{R}$
 - Intrări de orice fel
 - Ieșiri numerice (naturale, întregi, reale)
- Structurată: $\mathcal{X} \rightarrow \mathcal{Y}$
 - Intrări de orice fel
 - Ieșiri de orice fel (simple sau structurate)

- Informații structurate

- Texte și hiper-texte
- Molecule și structuri moleculare
- Imagini



Sisteme inteligețe – SIS – MSV

□ MSV structurate

■ Aplicații

- Procesarea limbajului natural
 - Traduceri automate (ieșiri → propoziții)
 - Analiza sintactică și/sau morfologică a propozițiilor (ieșiri → arborele sintactic și/sau morfologic)
- Bioinformatică
 - Predicția unor structuri secundare (ieșirile → grafe bi-partite)
 - Predicția funcționării unor enzime (ieșirile → *path*-uri în arbori)
- Procesarea vorbirii
 - Transcrieri automate (ieșiri → propoziții)
 - Transformarea textelor în voce (ieșiri → semnale audio)
- Robotică
 - Planificare (ieșirile → secvențe de acțiuni)

Sisteme inteligente – SIS – MSV

□ Avantaje

- Pot lucra cu orice fel de date (liniar separabile sau nu, distribuit uniform sau nu, cu distribuție cunoscută sau nu)
 - Funcțiile kernel care crează noi atribute (features) → straturile ascunse dintr-o RNA
- Dacă problema e convexă oferă o soluție unică → optimul global
 - RNA pot asocia mai multe soluții → optime locale
- Selectează automat mărimea modelului învățat (prin vectorii suport)
 - În RNA straturile ascunse trebuie configurate de către utilizator *apriori*
- Nu învăță pe derost datele (overfitting)
 - RNA se confruntă cu problema overfitting-ului chiar și cand modelul se învăță prin validare încruciată

□ Dificultăți

- Doar atribute reale
- Doar clasificare binară
- Background matematic dificil

□ Tool-uri

- LibSVM → <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- Weka → SMO
- SVMLight → <http://svmlight.joachims.org/>
- SVM Torch → <http://www.torch.ch/>
- <http://www.support-vector-machines.org/>

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

■ **Sisteme care învață singure**

- Arbori de decizie
- Rețele neuronale artificiale
- Mașini cu suport vectorial
- Algoritmi evolutivi

- Sisteme bazate pe reguli
- Sisteme hibride

C. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversarială

Cursul următor –

Materiale de citit și legături utile

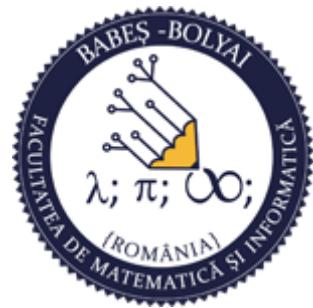
- capitolul 15 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

□ Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:

- Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean
- Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan
- Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure
– rețelele neuronale artificiale –

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

■ Sisteme care învață singure

- Arbori de decizie
- Rețele neuronale artificiale
- Mașini cu suport vectorial
- Algoritmi evolutivi

■ Sisteme bazate pe reguli

■ Sisteme hibride

C. Rezolvarea problemelor prin căutare

■ Definirea problemelor de căutare

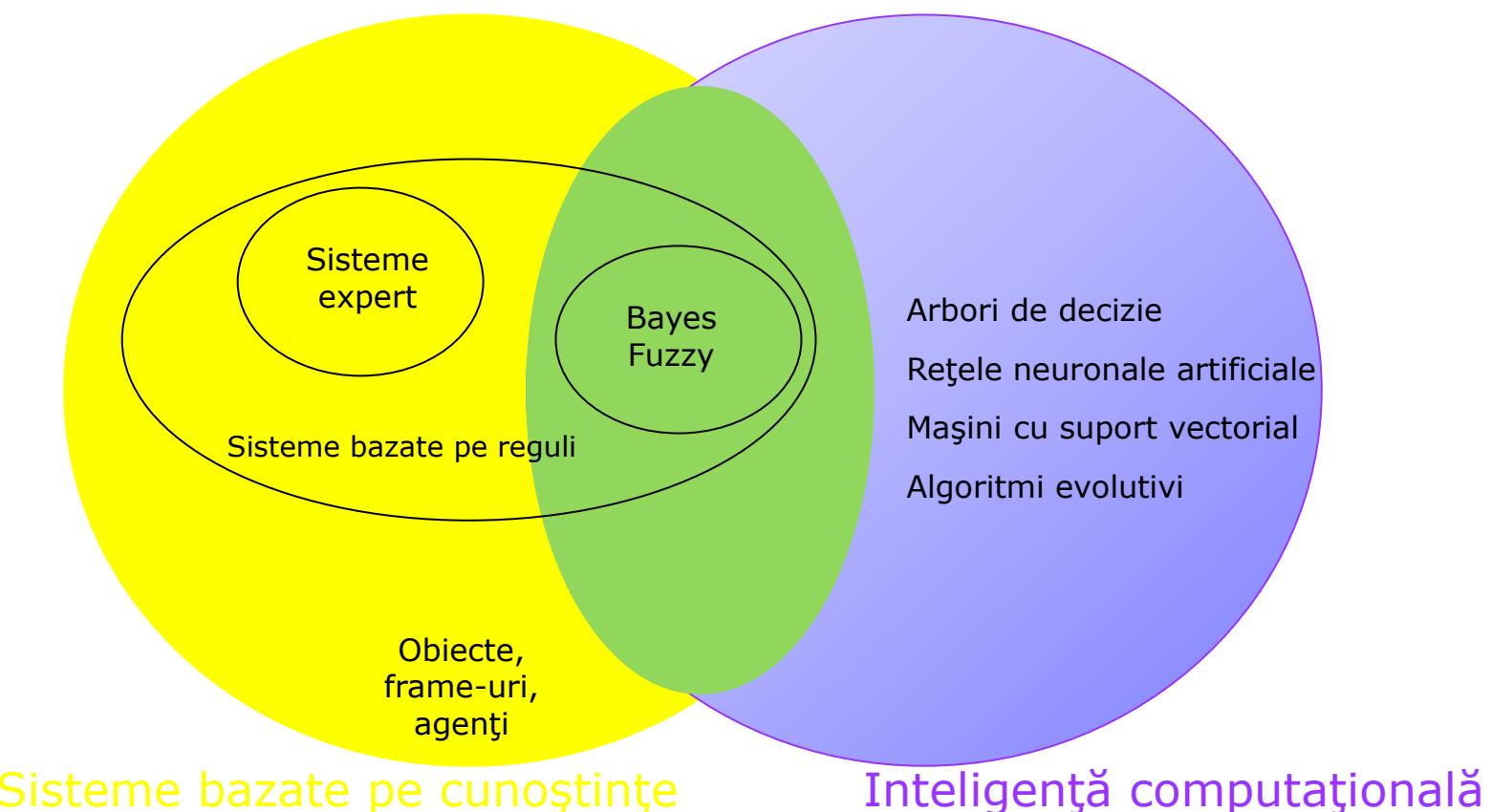
■ Strategii de căutare

- Strategii de căutare neinformate
- Strategii de căutare informate
- Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
- Strategii de căutare adversială

Materiale de citit și legături utile

- Capitolul VI (19) din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 8 din *Adrian A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- capitolul 12 și 13 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul V din *D. J. C. MacKey, Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003*
- Capitolul 4 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Sisteme inteligente



Sisteme inteligente – SIS – Învățare automată

□ Tipologie

- În funcție de experiența acumulată în timpul învățării:
 - SI cu învățare supervizată
 - SI cu învățare nesupervizată
 - SI cu învățare activă
 - SI cu învățare cu întărire
- În funcție de modelul învățat (algoritmul de învățare):
 - Arbori de decizie
 - **Rețele neuronale artificiale**
 - Algoritmi evolutivi
 - Mașini cu suport vectorial
 - Modele Markov ascunse

?
!



Sisteme inteligente – SIS – RNA

❑ Rețele neuronale artificiale (RNA)

- Scop
- Definire
- Tipuri de probleme rezolvabile
- Caracteristici
- Exemplu
- Proiectare
- Evaluare
- Tipologie

Sisteme inteligente – SIS – RNA

□ Scop

- Clasificare binară pentru orice fel de date de intrare (discrete sau continue)

- Datele pot fi separate de:

- o dreaptă $\rightarrow ax + by + c = 0$ (dacă $m = 2$)
 - un plan $\rightarrow ax + by + cz + d = 0$ (dacă $m = 3$)
 - un hiperplan $\sum a_i x_i + b = 0$ (dacă $m > 3$)

- Cum găsim valorile optime pt. a, b, c, d, a_i ?

- Rețele neuronale artificiale (RNA)
 - Mașini cu suport vectorial (MSV)

- De ce RNA?

- Cum învață creierul?

Sisteme inteligente – SIS – RNA

❑ Scop → De ce RNA?

- Unele sarcini pot fi efectuate foarte ușor de către oameni, însă sunt greu de codificat sub forma unor algoritmi
 - ❑ Recunoașterea formelor
 - vechi prieteni
 - caractere scrise de mâna
 - vocea
 - ❑ Diferite raționamente
 - conducerea autovehiculelor
 - cântatul la pian
 - jucarea baschetului
 - înnotul
- Astfel de sarcini sunt dificil de definit formal și este dificilă aplicarea unui proces de raționare pentru efectuarea lor

Sisteme inteligente – SIS – RNA

❑ Scop → Cum învață creierul?

■ Creierul uman - componență

- ❑ Aproximativ 10.000.000.000 de neuroni conectați prin sinapse

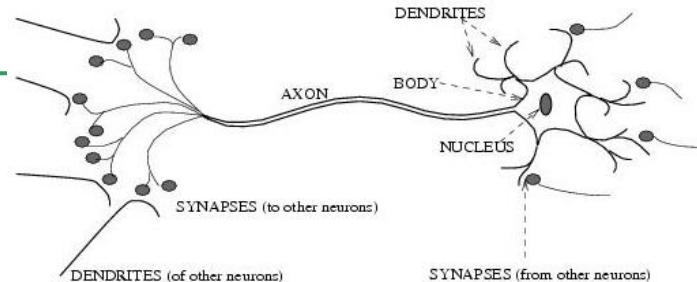
❑ Fiecare neuron

- are un corp (soma), un axon și multe dendrite
- poate fi într-o din 2 stări:
 - activ – dacă informația care intră în neuron depășește un anumit prag de stimulare –
 - pasiv – altfel

❑ Sinapsă

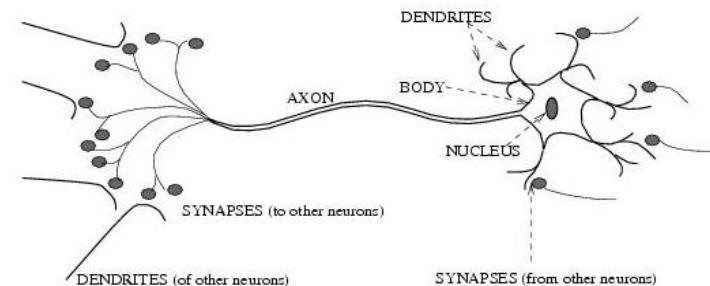
- Legătura între axon-ul unui neuron și dendritele altui neuron
- Are rol în schimbul de informație dintre neuroni
- 5.000 de conexiuni / neuron (în medie)

❑ În timpul vieții pot să apară noi conexiuni între neuroni



Sisteme inteligente – SIS – RNA

- Scop → Cum învață creierul?
 - Cum "învață" (procesează informații)?
 - Conexiunile care de-a lungul trăirii unor experiențe s-au dovedit utile devin permanente (restul sunt eliminate)
 - Creierul este interesat de noutăți
 - Modelul de procesare a informației
 - Învățare
 - Depozitare
 - Amintire
 - Memoria
 - Tipologie
 - De scurtă durată
 - Imediată → 30 sec.
 - De lucru
 - De lungă durată
 - Capacitate
 - Crește odată cu vârsta
 - Limitată → învățarea unei poezii pe strofe
 - Influențată și de stările emoționale
 - Creierul
 - rețea de neuroni
 - sistem foarte complex, ne-liniar și paralel de procesare a informației
 - Informația este depozitată și procesată de întreaga rețea, nu doar de o anumită parte a rețelei → informații și procesare globală
 - Caracteristica fundamentală a unei rețele de neuroni → învățarea → rețele neuronale artificiale (RNA)



Sisteme inteligente – SIS – RNA

- Definire
 - Ce este o RNA?
 - RN biologice vs. RN artificiale
 - Cum învață rețeaua?

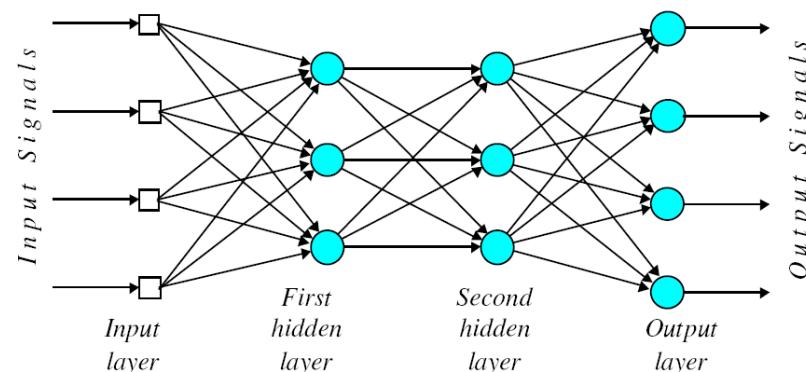
Sisteme inteligente – SIS – RNA

□ Definire → Ce este o RNA?

- O structură similară unei rețele neuronale biologice
- O mulțime de noduri (unități, neuroni, elemente de procesare) dispuse ca într-un graf pe mai multe straturi (*layere*)
 - Nodurile
 - au intrări și ieșiri
 - efectuează un calcul simplu prin intermediul unei funcții asociate → funcție de activare
 - sunt conectate prin legături ponderate
 - Conexiunile între noduri conturează structura (arhitectura) rețelei
 - Conexiunile influențează calculele care se pot efectua

□ Straturile

- Strat de intrare
 - Conține m (nr de atribute al unei date) noduri
- Strat de ieșire
 - Conține r (nr de ieșiri) noduri
- Straturi intermediare (ascunse) – rol în “complicarea” rețelei
 - Diferite structuri
 - Diferite mărimi



Sisteme inteligente – SIS – RNA

□ Definire → RN biologice vs. RN artificiale

RNB	RNA
Soma	Nod
Dendrite	Intrare
Axon	Ieșire
Activare	Procesare
Synapsă	Conexiune ponderată

□ Definire → Cum învață rețeaua?

- Plecând de la un set de n date de antrenament de forma

$$((x_{p1}, x_{p2}, \dots, x_{pm}, y_{p1}, y_{p2}, \dots, y_{pr}))$$

cu $p = 1, 2, \dots, n$, m – nr atributelor, r – nr ieșirilor

- se formează o RNA cu m noduri de intrare, r noduri de ieșire și o anumită structură internă
 - un anumit nr de nivele ascunse, fiecare nivel cu un anumit nr de neuroni
 - cu legături ponderate între oricare 2 noduri
- se caută valorile optime ale ponderilor între oricare 2 noduri ale rețelei prin minimizarea erorii
 - diferența între rezultatul real y și cel calculat de către rețea

Sisteme inteligente – SIS – RNA

- Tipuri de probleme rezolvabile cu RNA
 - Datele problemei se pot reprezenta prin numeroase perechi atribut-valoare
 - Funcția obiectiv poate fi:
 - Unicriterială sau multicriterială
 - Discretă sau cu valori reale
 - Datele de antrenament pot conține erori (zgomot)
 - Timp de rezolvare (antrenare) prelungit

Sisteme inteligente – SIS – RNA

□ Proiectare

- Construirea RNA pentru rezolvarea problemei P
- Inițializarea parametrilor RNA
- Antrenarea RNA
- Testarea RNA

Sisteme inteligente – SIS – RNA

□ Proiectare

- Construirea RNA pentru rezolvarea unei probleme P
 - pp. o problemă de clasificare în care avem un set de date de forma:
 - (x^d, t^d) , cu:
 - $x^d \in \mathbb{R}^m \rightarrow x^d = (x_{1^d}, x_{2^d}, \dots, x_{m^d})$
 - $t^d \in \mathbb{R}^R \rightarrow t^d = (t_{1^d}, t_{2^d}, \dots, t_{R^d})$,
 - cu $d = 1, 2, \dots, n, n+1, n+2, \dots, N$
 - primele n date vor fi folosite drept bază de antrenament a RNA
 - ultimele $N-n$ date vor fi folosite drept bază de testare a RNA
 - se construiește o RNA astfel:
 - stratul de intrare conține exact m noduri (fiecare nod va citi una dintre proprietățile de intrare ale unei instanțe a problemei – $x_{1^d}, x_{2^d}, \dots, x_{m^d}$)
 - stratul de ieșire poate conține R noduri (fiecare nod va furniza una dintre proprietățile de ieșire ale unei instanțe a problemei $t_{1^d}, t_{2^d}, \dots, t_{R^d}$)
 - unul sau mai multe straturi ascunse cu unul sau mai mulți neuroni pe fiecare strat

Sisteme inteligente – SIS – RNA

- Proiectare
 - Construirea RNA pentru rezolvarea problemei P
 - **Inițializarea parametrilor RNA**
 - **Antrenarea RNA**
 - Testarea RNA

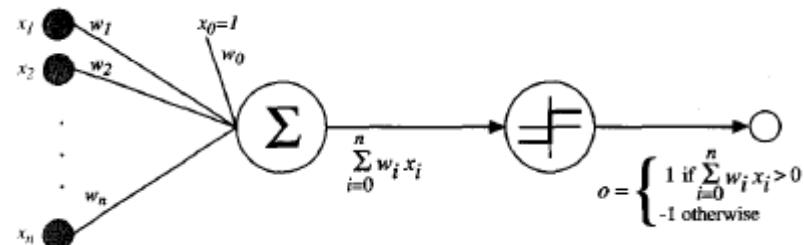
Sisteme inteligente – SIS – RNA

□ Proiectare

- Inițializarea parametrilor RNA
 - Inițializarea ponderile între oricare 2 noduri de pe straturi diferite
 - Stabilirea funcției de activare corespunzătoare fiecărui neuron (de pe straturile ascunse)
- Antrenarea (învățarea) RNA
 - Scop:
 - stabilirea valorii optime a ponderilor dintre 2 noduri
 - Algoritm:
 - Se caută valorile optime ale ponderilor între oricare 2 noduri ale rețelei prin minimizarea erorii (diferența între rezultatul real y și cel calculat de către rețea)
 - Cum învață rețeaua?
 - Rețeaua = mulțime de unități primitive de calcul interconectate între ele →
 - Învățarea rețelei = \cup Învățarea unităților primitive
 - Unități primitive de calcul
 - Perceptron
 - Unitate liniară
 - Unitate sigmoidală

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață rețeaua?
 - Neuronul ca element simplu de calcul
 - Structura neuronului
 - Fiecare nod are intrări și ieșiri
 - Fiecare nod efectuează un calcul simplu
 - Procesarea neuronului
 - Se transmite informația neuronului
 - Neuronul procesează informația
 - Se citește răspunsul neuronului
 - Învățarea neuronului – algoritmul de învățare a ponderilor care procesează corect informațiile
 - Se pornește cu un set inițial de ponderi oarecare
 - Cât timp nu este îndeplinită o condiție de oprire
 - Se procesează informația și se stabilește calitatea ponderilor curente
 - Se modifică ponderile astfel încât să se obțină rezultate mai bune



Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață rețeaua?
 - Neuronul ca element simplu de calcul
 - Structura neuronului
 - Fiecare nod are intrări și ieșiri
 - Fiecare nod efectuează un calcul simplu prin intermediul unei funcții asociate
 - Procesarea neuronului
 - Se transmite informația neuronului → se calculează suma ponderată a intrărilor
 - Neuronul procesează informația → se folosește o funcție de activare:
 - Funcția constantă
 - Funcția prag
 - Funcția rampă
 - Funcția liniară
 - Funcția sigmoidală
 - Funcția Gaussiană
 - Funcția Relu

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

■ Funcția de activare a unui neuron

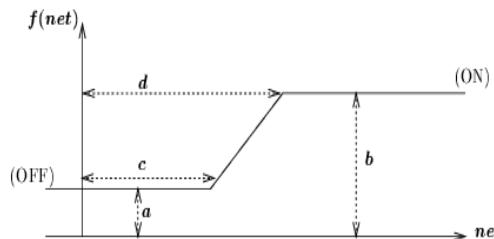
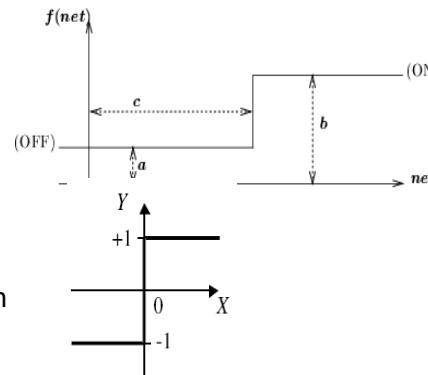
- Funcția constantă $f(\text{net}) = \text{const}$
- Funcția prag (c - pragul)

$$f(\text{net}) = \begin{cases} a, & \text{dacă } \text{net} < c \\ b, & \text{dacă } \text{net} > c \end{cases}$$

- Pentru $a=+1$, $b=-1$ și $c=0 \rightarrow$ funcția semn
- Funcție discontinuă

□ Funcția rampă

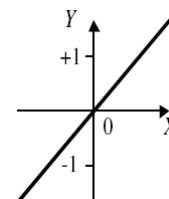
$$f(\text{net}) = \begin{cases} a, & \text{dacă } \text{net} \leq c \\ b, & \text{dacă } \text{net} \geq d \\ a + \frac{(\text{net} - c)(b - a)}{d - c}, & \text{altfel} \end{cases}$$



□ Funcția liniară

$$f(\text{net}) = a * \text{net} + b$$

- Pentru $a=1$ și $b=0 \rightarrow$ funcția identitate $f(\text{net})=\text{net}$
- Funcție continuă



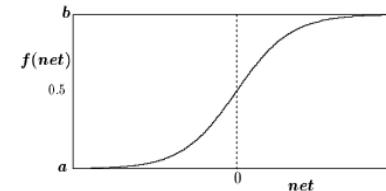
Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

■ Funcția de activare a unui neuron

□ Funcția sigmoidală

- În formă de S
- Continuă și diferențiabilă în orice punct
- Simetrică rotațional față de un anumit punct ($net = c$)
- Atinge asymptotic puncte de saturare



$$\lim_{\text{net} \rightarrow -\infty} f(\text{net}) = a \quad \lim_{\text{net} \rightarrow \infty} f(\text{net}) = b$$

- Exemple de funcții sigmoidale:

$$f(\text{net}) = z + \frac{1}{1 + \exp(-x \cdot \text{net} + y)}$$

$$f(\text{net}) = \tanh(x \cdot \text{net} - y) + z$$

$$\text{unde } \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

- Pentru $y=0$ și $z = 0 \Rightarrow a=0, b = 1, c=0$
- Pentru $y=0$ și $z = -0.5 \Rightarrow a=-0.5, b = 0.5, c=0$
- Cu cât x este mai mare, cu atât curba este mai abruptă

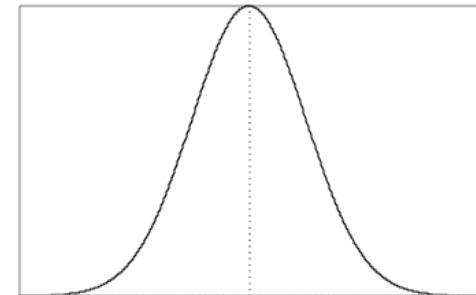
Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

■ Funcția de activare a unui neuron

- Funcția Gaussiană
 - În formă de clopot
 - Continuă
 - Atinge asimptotic un punct de saturatie

$$\lim_{net \rightarrow \infty} f(net) = a$$



- Are un singur punct de optim (maxim) – atins când net = μ
- Exemplu

$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{net - \mu}{\sigma}\right)^2\right]$$

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

■ Funcția de activare a unui neuron

■ Funcția ReLU

- În formă de rampă
- Continuă, monotonă
- Derivata ei este monotonă
- Codomeniu pozitiv $[0, \infty)$

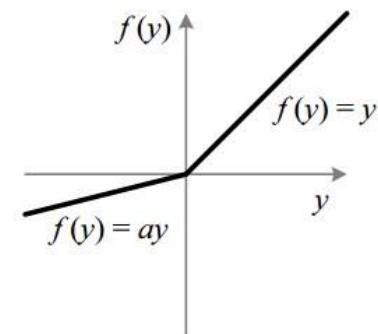
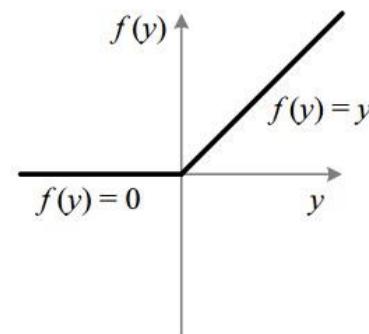
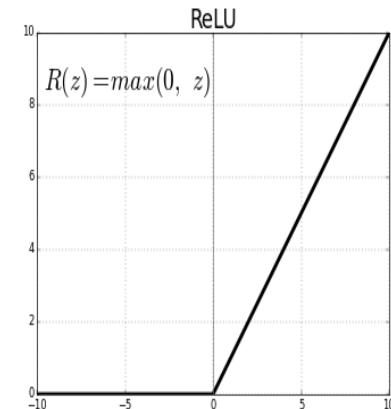
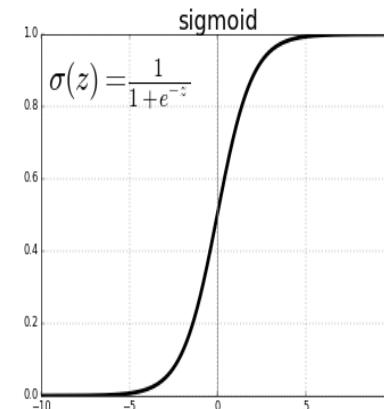
$$f(\text{net}) = \max(0, \text{net})$$

$$f(\text{net}) = \begin{cases} 0, & \text{dacă } \text{net} < 0 \\ \text{net}, & \text{dacă } \text{net} \geq 0 \end{cases}$$

■ Variantă: Leaky ReLU

- Compensează problemele cu argumentele negative din ReLU

$$f(\text{net}) = \begin{cases} a \cdot \text{net}, & \text{dacă } \text{net} < 0 \\ \text{net}, & \text{dacă } \text{net} \geq 0 \end{cases}$$



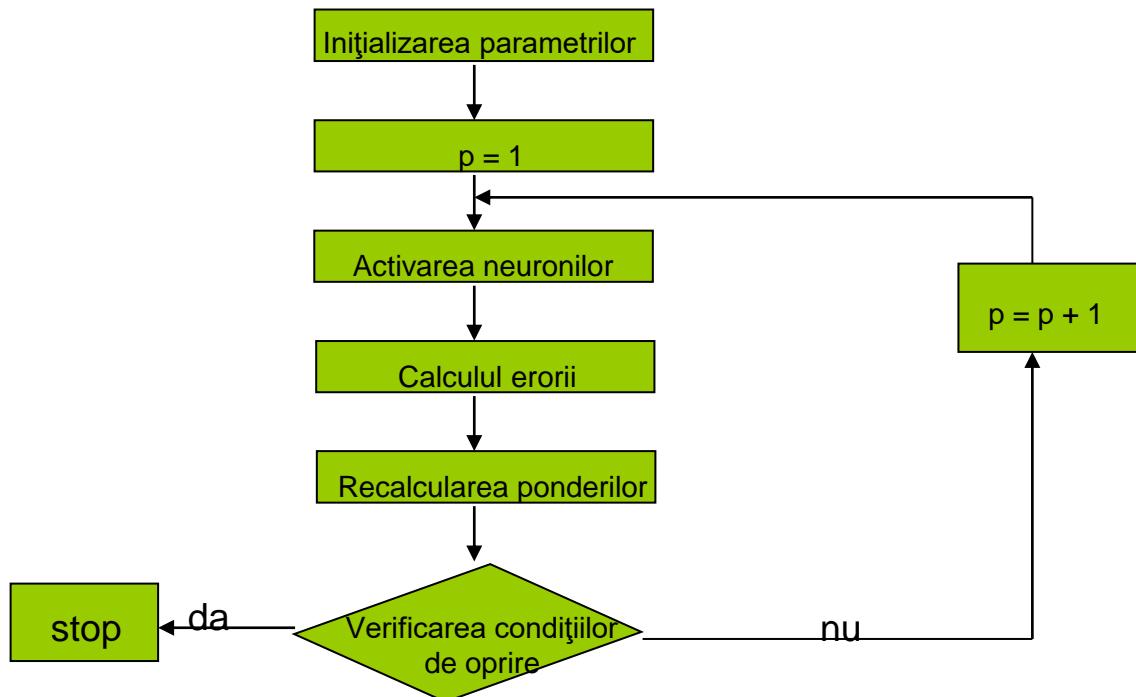
Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață rețeaua?
 - Neuronul ca element simplu de calcul
 - Structura neuronului
 - Procesarea neuronului
 - Se transmite informația neuronului → se calculează suma ponderată a intrărilor
 - Neuronul procesează informația → se folosește o funcție de activare:
 - Funcția constantă
 - Funcția prag
 - Funcția rampă
 - Funcția liniară
 - Funcția sigmoidală
 - Funcția Gaussiană
 - Se citește răspunsul neuronului → se stabilește dacă rezultatul furnizat de neuron coincide sau nu cu cel dorit (real)

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

- Neuronul ca element simplu de calcul
 - Structura neuronului
 - Procesarea neuronului
 - Învățarea neuronului
 - Algoritm



Sisteme inteligente – SIS – RNA

❑ Proiectare → Antrenarea RNA → Cum învață RNA?

■ Învățarea neuronului

❑ 2 reguli de bază

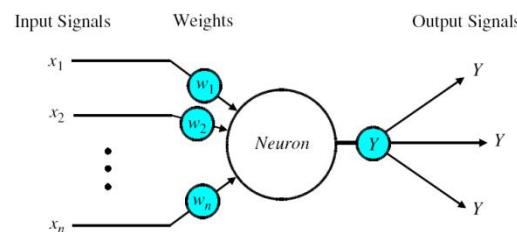
- Regula perceptronului → algoritmul perceptronului
 1. Se porneste cu un set de ponderi oarecare
 2. Se stabilește calitatea modelului creat pe baza acestor ponderi pentru **UNA** dintre datele de intrare
 3. Se ajustează ponderile în funcție de calitatea modelului
 4. Se reia algoritmul de la pasul 2 până când se ajunge la calitate maximă
- Regula Delta → algoritmul scăderii după gradient
 1. Se porneste cu un set de ponderi oarecare
 2. Se stabilește calitatea modelului creat pe baza acestor ponderi pentru **TOATE** dintre datele de intrare
 3. Se ajustează ponderile în funcție de calitatea modelului
 4. Se reia algoritmul de la pasul 2 până când se ajunge la calitate maximă
- Similar regulii perceptronului, dar calitatea unui model se stabilește în funcție de toate datele de intrare (tot setul de antrenament)

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață RNA?

■ Învățarea neuronului

- Pp că avem un set de date de antrenament de forma:
 - (x^d, t^d) , cu:
 - $x^d \in \mathbb{R}^m \rightarrow x^d = (x_1^d, x_2^d, \dots, x_m^d)$
 - $t^d \in \mathbb{R}^R \rightarrow t^d = (t_1^d, t_2^d, \dots, t_R^d)$, și $R = 1$ (adică $t^d = (t_1^d)$)
 - cu $d = 1, 2, \dots, n$
 - RNA = unitate primitivă de calcul (un neuron) \rightarrow o rețea cu:
 - m noduri de intrare
 - legate de neuronul de calcul prin ponderile w_i , $i = 1, 2, \dots, m$ și
 - cu un nod de ieșire



Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață RNA?

■ Învățarea neuronului

□ Algoritmul perceptronului

- Se bazează pe minimizarea erorii asociată unei instanțe din setul de date de antrenament
- Modificarea ponderilor pe baza erorii asociate unei instanțe din setul de antrenament

Inițializare ponderi din rețea

$$w_i = \text{random}(a,b), \text{ unde } i=1,2,\dots,m$$

$$d = 1$$

Cât timp mai există exemple de antrenament clasificate incorrect

Se activează neuronul și se calculează ieșirea

Perceptron → funcția de activare este funcția semn (funcție prag de tip discret, nediferențiabil)

$$o^d = \text{sign}(\mathbf{wx}) = \text{sign}\left(\sum_{i=1}^m w_i x_i\right)$$

$$\text{Se stabilește ajustarea ponderilor } \Delta w_i = \eta(t^d - o^d)x_i^d, \text{ unde } i=1,2,\dots,m$$

unde η - rată de învățare

$$\text{Se ajustează ponderile } w'_i = w_i + \Delta w_i$$

Dacă $d < n$ atunci $d++$

Altfel $d = 1$

SfCâtTimp

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață RNA?
 - Învățarea neuronului
 - Algoritmul scădere după gradient
 - Se bazează pe eroarea asociată întregului set de date de antrenament
 - Modificarea ponderilor în direcția dată de cea mai abruptă pantă a reducerii erorii $E(\mathbf{w})$ pentru tot setul de antrenament
 - Cum se determină cea mai abruptă pantă? ➔ se derivează E în funcție de w (se stabilește gradientul erorii E)
$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_m} \right)$$
 - Gradientul erorii E se calculează în funcție de funcția de activare a neuronului (care trebuie să fie diferențiabilă, deci continuă)
 - Funcția liniară $f(net) = \sum_{i=1}^m w_i x_i^d$
 - Funcția sigmoidală $f(net) = \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}}} = \frac{1}{1 + e^{-\sum_{i=1}^m w_i x_i^d}}$
 - Cum se ajustează ponderile?

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \text{ unde } i = 1, 2, \dots, m$$

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA → Cum învață RNA? ■ Învățarea neuronului

- Algoritmul scădere după gradient → calcularea gradientului erorii
 - Funcția liniară

$$\begin{aligned}f(\text{net}) &= \sum_{i=1}^m w_i x_i^d \\ \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d=1}^n (t^d - o^d)^2 = \frac{1}{2} \sum_{d=1}^n \frac{\partial (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n 2(t^d - o^d) \frac{\partial (t^d - o^d)}{\partial w_i} \\ \frac{\partial E}{\partial w_i} &= \sum_{d=1}^n (t^d - o^d) \frac{\partial (t^d - w_1 x_1^d - w_2 x_2^d - \dots - w_m x_m^d)}{\partial w_i} = \sum_{d=1}^n (t^d - o^d)(-x_i^d)\end{aligned}$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d=1}^n (t^d - o^d)x_i^d$$

- Funcție sigmoidală

$$f(\text{net}) = \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}}} = \frac{1}{1 + e^{-\sum_{i=1}^m w_i x_i^d}} \quad y = s(z) = \frac{1}{1 + e^{-z}} \Rightarrow \frac{\partial s(z)}{\partial z} = s(z)(1 - s(z))$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d=1}^n (t^d - o^d)^2 = \frac{1}{2} \sum_{d=1}^n \frac{\partial (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n 2(t^d - o^d) \frac{\partial (t^d - \text{sig}(\mathbf{w}\mathbf{x}^d))}{\partial w_i} = \sum_{d=1}^n (t^d - o^d)(1 - o^d)o^d(-x_i^d)$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d=1}^n (t^d - o^d)(1 - o^d)o^d x_i^d$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață RNA?
 - Învățarea neuronului
 - Algoritmul scădere după gradient (ASG)

ASG simplu	ASG stocastic
<p>Inițializare ponderi din rețea $w_i = \text{random}(a,b)$, unde $i=1,2,\dots,m$</p> <p>$d = 1$</p> <p>Cât timp nu este îndeplinită condiția de oprire $\Delta w_i = 0$, unde $i=1,2,\dots,m$</p> <p>Pentru fiecare exemplu de antrenament (x^d, t^d), unde $d=1,2,\dots,n$</p> <p>Se activează neuronul și se calculează ieșirea o^d</p> <p>funcția de activare = funcția liniară $\rightarrow o^d = \mathbf{wx}^d$</p> <p>funcția de activare = funcția sigmoid $\rightarrow o^d = \text{sig}(\mathbf{wx}^d)$</p> <p>Pentru fiecare pondere w_i, unde $i=1,2,\dots,m$</p> <p>Se stabilește ajustarea ponderii $\Delta w_i = \Delta w_i - \eta \frac{\partial E}{\partial w_i}$</p> <p>Pentru fiecare pondere w_i, unde $i=1,2,\dots,m$</p> <p>Se ajustează fiecare pondere w_i $w_i = w_i + \Delta w_i$</p>	<p>Inițializare ponderi din rețea $w_i = \text{random}(a,b)$, unde $i=1,2,\dots,m$</p> <p>$d = 1$</p> <p>Cât timp nu este îndeplinită condiția de oprire $\Delta w_i = 0$, unde $i=1,2,\dots,m$</p> <p>Pentru fiecare exemplu de antrenament (x^d, t^d), unde $d=1,2,\dots,n$</p> <p>Se activează neuronul și se calculează ieșirea o^d</p> <p>funcția de activare = funcția liniară $\rightarrow o^d = \mathbf{wx}^d$</p> <p>funcția de activare = funcția sigmoid $\rightarrow o^d = \text{sig}(\mathbf{wx}^d)$</p> <p>Pentru fiecare pondere w_i, unde $i=1,2,\dots,m$</p> <p>Se stabilește ajustarea ponderilor $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$</p> <p>Se ajustează ponderea w_i $w_i = w_i + \Delta w_i$</p>

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață RNA?
 - Învățarea neuronului

Diferențe	Algoritmul perceptronului	Algoritmul scădere după gradient (regula Delta)
Ce reprezintă o^d	$o^d = \text{sign}(\mathbf{w}\mathbf{x}^d)$	$o^d = \mathbf{w}\mathbf{x}^d$ sau $o^d = \text{sig}(\mathbf{w}\mathbf{x}^d)$
Cum converge	Într-un nr finit de pași (până la separarea perfectă)	Asimtotic (spre eroarea minimă)
Ce fel de probleme se pot rezolva	Cu date liniar separabile	Cu orice fel de date (separabile liniar sau neliiniar)
Ce tip de ieșire are neuronul	Discretă și cu prag	Continuă și fără prag

Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA

■ Cum învață rețeaua?

- Rețeaua = mulțime de unități primitive de calcul interconectate între ele →
 - Învățarea rețelei = \cup învățarea unităților primitive
- Rețea cu mai mulți neuroni așezăți pe unul sau mai multe straturi → RNA este capabilă să învețe un model mai complicat (nu doar liniar) de separare a datelor
- Algoritmul de învățare a ponderilor → backpropagation
 - Bazat pe algoritm scădere după gradient
 - Îmbogățit cu:
 - Informația se propagă în RNA înainte (dinspre stratul de intrare spre cel de ieșire)
 - Eroarea se propagă în RNA înapoi (dinspre stratul de ieșire spre cel de intrare)

Se inițializează ponderile

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se propagă informația înainte și se calculează ieșirea corespunzătoare fiecărui neuron al rețelei

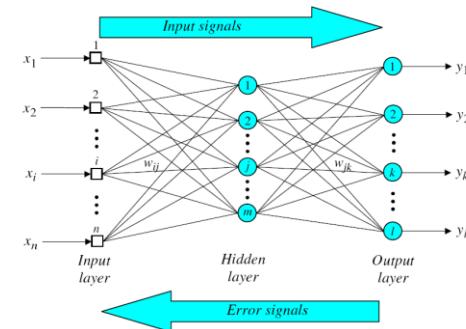
Se ajustează ponderile

Se stabilește și se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieșire

Se propagă aceste erori înapoi în toată rețeaua → se distribuie erorile pe toate conexiunile existente în rețea proporțional cu valorile ponderilor asociate acestor conexiuni

Se modifică ponderile



Sisteme inteligente – SIS – RNA

□ Proiectare → Antrenarea RNA

■ Cum învață o întreagă RNA?

- Pp că avem un set de date de antrenament de forma:

- (x^d, t^d) , cu:

- $x^d \in \mathbb{R}^m \rightarrow x^d = (x_{d_1}^d, x_{d_2}^d, \dots, x_{d_m}^d)$
 - $t^d \in \mathbb{R}^R \rightarrow t^d = (t_{d_1}^d, t_{d_2}^d, \dots, t_{d_R}^d)$
 - cu $d = 1, 2, \dots, n$

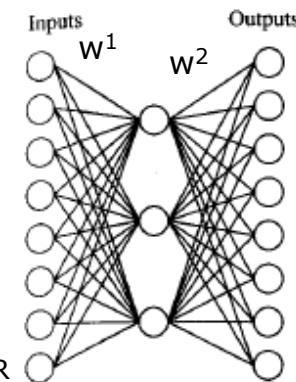
- Presupunem 2 cazuri de RNA

- O RNA cu un singur strat ascuns cu H neuroni \rightarrow RNA_1

- m neuroni pe stratul de intrare,
 - R neuroni pe stratul de ieșire,
 - H neuroni pe stratul ascuns
 - Ponderile între stratul de intrare și cel ascuns w_{ih}^1 cu $i=1, 2, \dots, m$, $h = 1, 2, \dots, H$
 - Ponderile între stratul ascuns și cel de ieșire w_{hr}^2 cu $h = 1, 2, \dots, H$ și $r = 1, 2, \dots, R$

- O RNA cu p straturi ascunse, fiecare strat cu H_i ($i = 1, 2, \dots, p$) neuroni $\rightarrow RNA_p$

- m neuroni pe stratul de intrare,
 - R neuroni pe stratul de ieșire,
 - P straturi ascunse
 - H_p neuroni pe stratul ascuns p , $p = 1, 2, \dots, P$
 - Ponderile între stratul de intrare și primul strat ascuns w_{ih}^1 cu $i=1, 2, \dots, m$, $h_1 = 1, 2, \dots, H_1$
 - Ponderile între primul strat ascuns și cel de-al doilea strat ascuns $w_{h_1 h_2}^2$ cu $h_1 = 1, 2, \dots, H_1$ și $h_2 = 1, 2, \dots, H_2$
 - Ponderile între cel de-al doilea strat ascuns și cel de-al treilea strat ascuns $w_{h_2 h_3}^3$ cu $h_2 = 1, 2, \dots, H_2$ și $h_3 = 1, 2, \dots, H_3$
 - ...
 - Ponderile între cel de-al $p-1$ strat ascuns și ultimul strat ascuns $w_{h_{p-1} h_p}^p$ cu $h_{p-1} = 1, 2, \dots, H_{p-1}$ și $h_p = 1, 2, \dots, H_p$
 - Ponderile între ultimul strat ascuns și cel de ieșire $w_{h_p r}^{p+1}$ cu $h_p = 1, 2, \dots, H_p$ și $r = 1, 2, \dots, R$



Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață o întreagă RNA?
 - Algoritmul backpropagation pentru RNA₁

Se inițializează ponderile w_{ih}^1 și w_{hr}^2 cu $i=1,2,\dots,m$, $h=1,2,\dots,H$ și $r=1,2,\dots,R$

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se propagă informația înainte și se calculează ieșirea corespunzătoare fiecărui neuron al rețelei

$$o_h^d = \sum_{i=1}^m w_{ih}^1 x_i^d \text{ sau } o_h^d = \text{sig}\left(\sum_{i=1}^m w_{ih}^1 x_i^d\right), \text{ cu } h=1,2,\dots,H$$

$$o_r^d = \sum_{h=1}^H w_{hr}^2 o_h^d \text{ sau } o_r^d = \text{sig}\left(\sum_{h=1}^H w_{hr}^2 o_h^d\right), \text{ cu } r=1,2,\dots,R$$

Se ajustează ponderile

Se stabilește și se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieșire

$$\delta_r^d = t_r^d - o_r^d \text{ sau } \delta_r^d = o_r^d(1-o_r^d)(t_r^d - o_r^d), \text{ cu } r=1,2,\dots,R$$

Se modifică ponderile între nodurile de pe stratul ascuns și stratul de ieșire

$$w_{hr}^2 = w_{hr}^2 + \eta \delta_r^d o_h^d, \text{ unde } h=1,2,\dots,H \text{ și } r=1,2,\dots,R$$

Se propagă erorile nodurilor de pe stratul de ieșire înapoi în toată rețeaua → se distribuie erorile pe toate conexiunile existente în rețea proporțional cu valorile ponderilor asociate acestor conexiuni

$$\delta_h^d = \sum_{r=1}^R w_{hr}^2 \delta_r^d \text{ sau } \delta_h^d = o_h^d(1-o_h^d) \sum_{r=1}^R w_{hr}^2 \delta_r^d$$

Se modifică ponderile între nodurile de pe stratul de intrare și stratul ascuns

$$w_{ih}^1 = w_{ih}^1 + \eta \delta_h^d x_i^d, \text{ unde } i=1,2,\dots,m \text{ și } h=1,2,\dots,H$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață o întreagă RNA?
 - Algoritmul backpropagation pentru RNA_p

Se inițializează ponderile $w_{ih_1}^1, w_{h_1 h_2}^2, \dots, w_{h_{p-1} h_p}^p, w_{h_p r}^{p+1}$

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se propagă informația înainte și se calculează ieșirea corespunzătoare fiecărui neuron al rețelei

$$o_{h_1}^d = \sum_{i=1}^m w_{ih_1}^1 x_i^d \text{ sau } o_{h_1}^d = \text{sig}\left(\sum_{i=1}^m w_{ih_1}^1 x_i^d\right), \text{ cu } h_1 = 1, 2, \dots, H_1$$

$$o_{h_2}^d = \sum_{h_1=1}^{H_1} w_{h_1 h_2}^2 o_{h_1}^d \text{ sau } o_{h_2}^d = \text{sig}\left(\sum_{h_1=1}^{H_1} w_{h_1 h_2}^2 o_{h_1}^d\right), \text{ cu } h_2 = 1, 2, \dots, H_2$$

...

$$o_{h_p}^d = \sum_{h_{p-1}=1}^{H_{p-1}} w_{h_{p-1} h_p}^p o_{h_{p-1}}^d \text{ sau } o_{h_p}^d = \text{sig}\left(\sum_{h_{p-1}=1}^{H_{p-1}} w_{h_{p-1} h_p}^p o_{h_{p-1}}^d\right), \text{ cu } h_p = 1, 2, \dots, H_p$$

$$o_r^d = \sum_{h_p=1}^{H_p} w_{h_p r}^{p+1} o_{h_p}^d \text{ sau } o_r^d = \text{sig}\left(\sum_{h_p=1}^{H_p} w_{h_p r}^{p+1} o_{h_p}^d\right), \text{ cu } r = 1, 2, \dots, R$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață o întreagă RNA?
 - Algoritmul backpropagation pentru RNA_p

Se inițializează ponderile $w_{ih_1}^1, w_{ih_2}^2, \dots, w_{h_{p-1}h_p}^p, w_{h_p r}^{p+1}$

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se ajustează ponderile

Se stabilește și se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieșire

$\delta_r^d = t_r^d - o_r^d$ sau $\delta_r^d = o_r^d(1 - o_r^d)(t_r^d - o_r^d)$, cu $r = 1, 2, \dots, R$
Se modifică ponderile între nodurile de pe ultimul strat ascuns și stratul de ieșire

$$w_{h_p r}^{p+1} = w_{h_p r}^{p+1} + \eta \delta_r^d o_{h_p}^d, \text{ unde } h_p = 1, 2, \dots, H_p \text{ și } r = 1, 2, \dots, R$$

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață o întreagă RNA?
 - Algoritmul backpropagation pentru RNA_p

Se inițializează ponderile $w_{ih_1}^1, w_{h_1 h_2}^2, \dots, w_{h_{p-1} h_p}^p, w_{h_p r}^{p+1}$

Cât timp nu este îndeplinită condiția de oprire

Pentru fiecare exemplu (x^d, t^d)

Se activează fiecare neuron al rețelei

Se ajustează ponderile

Se stabilește și se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieșire

Se modifică ponderile între nodurile de pe ultimul strat ascuns și stratul de ieșire

Se propagă (pe starturi) aceste erori înapoi în toată rețeaua → se distribuie erorile pe toate conexiunile existente în rețea proporțional cu valorile ponderilor asociate acestor conexiuni și se modifică ponderile corespunzătoare

$$\delta_{h_p}^d = \sum_{r=1}^R w_{h_p r}^{p+1} \delta_r^d \text{ sau } \delta_{h_p}^d = o_{h_p}^d (1 - o_{h_p}^d) \sum_{r=1}^R w_{h_p r}^{p+1} \delta_r^d$$

$$w_{h_p r}^{p+1} = w_{h_p r}^{p+1} + \eta \delta_r^d o_{h_p}^d, \text{ unde } h_p = 1, 2, \dots, H_p \text{ și } r = 1, 2, \dots, R$$

$$\delta_{h_{p-1}}^d = \sum_{h_p=1}^{H_p} w_{h_{p-1} h_p}^p \delta_{h_p}^d \text{ sau } \delta_{h_{p-1}}^d = o_{h_{p-1}}^d (1 - o_{h_{p-1}}^d) \sum_{h_p=1}^{H_p} w_{h_{p-1} h_p}^p \delta_{h_p}^d$$

$$w_{h_{p-1} h_p}^p = w_{h_{p-1} h_p}^p + \eta \delta_{h_p}^d o_{h_{p-1}}^d, \text{ unde } h_{p-1} = 1, 2, \dots, H_{p-1} \text{ și } h_p = 1, 2, \dots, H_p$$

...

$$\delta_{h_1}^d = \sum_{h_2=1}^{H_2} w_{h_1 h_2}^2 \delta_{h_2}^d \text{ sau } \delta_{h_1}^d = o_{h_1}^d (1 - o_{h_1}^d) \sum_{h_2=1}^{H_2} w_{h_1 h_2}^2 \delta_{h_2}^d$$

$$w_{ih_1}^1 = w_{ih_1}^1 + \eta \delta_{h_1}^d x_i^d, \text{ unde } i = 1, 2, \dots, m \text{ și } h_1 = 1, 2, \dots, H_1$$

Inteligentă artificială - sisteme inteligente (RNA)

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învață o întreagă RNA?
 - Algoritmul backpropagation
 - Condiții de oprire
 - S-a ajuns la eroare 0
 - S-au efectuat un anumit număr de iterații
 - La o iterație se procesează un singur exemplu
 - n iterații = o epocă

Sisteme inteligente – SIS – RNA

- Proiectare
 - Construirea RNA pentru rezolvarea problemei P
 - Inițializarea parametrilor RNA
 - Antrenarea RNA
 - **Testarea RNA**

Sisteme inteligente – SIS – RNA

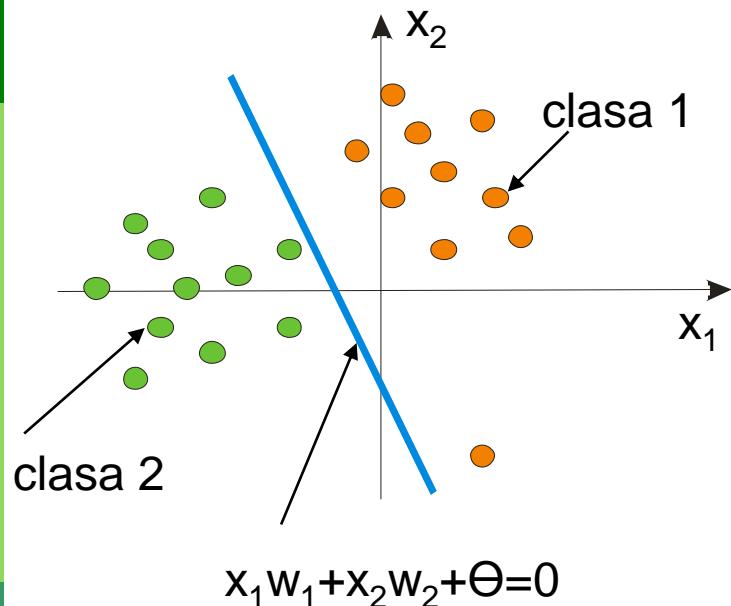
□ Proiectare

■ Testarea RNA

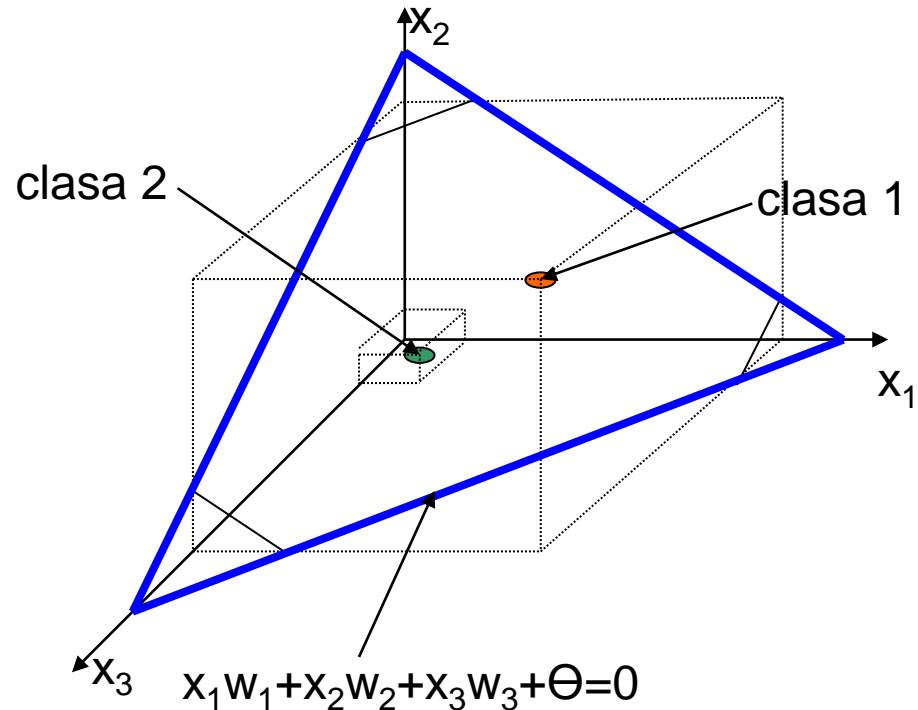
- Se decodifică modelul învățat de RNA
 - prin combinarea ponderilor cu intrările
 - ținând cont de funcțiile de activare a neuronilor și de structura rețelei

Sisteme inteligente – SIS – RNA

□ Exemplu



Clasificare binară cu $m=2$ intrări



Clasificare binară cu $m=3$ intrări

Sisteme inteligente – SIS – RNA

□ Exemplu

- Perceptron pentru rezolvarea problemei *SI logic*

Epoch	Inputs		Desired output Y_d	Initial weights		Actual output Y	Error e	Final weights	
	x_1	x_2		w_1	w_2			w_1	w_2
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	1	1	1	0.2	-0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	-1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	-1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	-1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

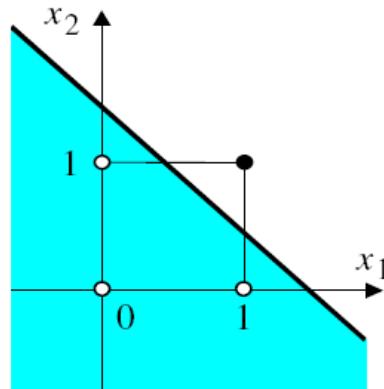
Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Sisteme inteligente – SIS – RNA

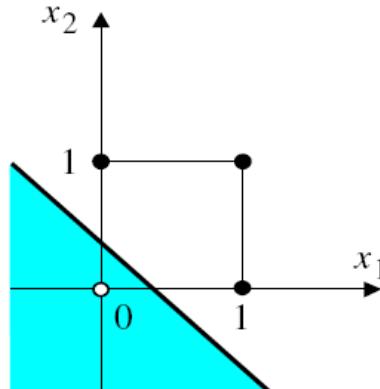
□ Exemplu

■ Perceptron - limitări

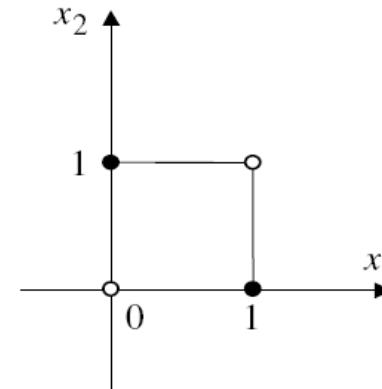
- Un perceptron poate învăța operațiile AND și OR, dar nu poate învăța operația XOR (nu este liniar separabilă)



(a) $AND (x_1 \cap x_2)$



(b) $OR (x_1 \cup x_2)$



(c) $Exclusive-OR$
 $(x_1 \oplus x_2)$

- Nu poate clasifica date non-liniar separabile

- soluții

- Neuron cu un prag continu
 - Mai mulți neuroni

Sisteme inteligente – SIS – RNA

□ Tipologie

■ RNA feed-forward

- Informația se procesează și circulă de pe un strat pe altul
- Conexiunile între noduri nu formează cicluri
- Se folosesc în special pentru învățarea supervizată
- Funcțiile de activare a nodurilor → liniare, sigmoidale, gaussiene

■ RNA recurente (cu feedback)

- Pot conține conexiuni între noduri de pe același strat
 - Conexiunile între noduri pot forma cicluri
 - RNA de tip Jordan
 - RNA de tip Elman
 - RNA de tip Hopfield
 - RNA auto-organizate → pentru învățarea nesupervizată
 - De tip Hebbian
 - De tip Kohonen (*Self organised maps*)
- 
- pentru învățarea supervizată

Sisteme inteligente – SIS – RNA

□ Avantaje

- Pot rezolva atât probleme de învățare super-vizată, cât și nesupervizată
- Pot identifica relații dinamice și neliniare între date
- Pot rezolva probleme de clasificare cu oricâte clase (multi-clasă)
- Se pot efectua calcule foarte rapid (în paralel și distribuit)

□ Dificultăți și limite

- RNA se confruntă cu problema overfitting-ului chiar și când modelul se învăță prin validare încrucișată
- RNA pot găsi (uneori) doar optimele locale (fără să identifice optimul global)

Deep learning

□ Deep learning

- methodology in which we can train machine complex representations
- addresses the problem of learning hierarchical representations with a single (a few) algorithm(s)
- models with a feature hierarchy (lower-level features are learned at one layer of a model, and then those features are combined at the next level).
- it's deep if it has more than one stage of non-linear feature transformation
- Hierarchy of representations with increasing level of abstraction
 - ▣ Image recognition
 - Pixel → edge → texton → motif → part → object
 - ▣ Text
 - Character → word → word group → clause → sentence → story
 - ▣ Speech
 - Sample → spectral band → sound → ... → phone → phoneme → word

□ Deep networks/architectures

- Convolutional NNs
- Auto-encoders
- Deep Belief Nets (Restricted Boltzmann machines)
- Recurrent Neural Networks

Deep learning

- Collection of methods to improve the optimisation and generalisation of learning methods, especially NNs:
 - Rectified linear units
 - Dropout
 - Batch normalisation
 - Weight decay regularisation
 - Momentum learning
- Stacking layers of transformations to create successively more abstract levels of representation
 - Depth over breadth
 - Deep MLPs
- Shared parameters
 - Convolutional NNs
 - Recurrent NNs
- Technological improvements
 - Massively parallel processing: GPUs, CUDA
 - Fast libraries: Torch, cuDNN, CUDA-convNet, Theano

ML & optimisation

- An ML algorithm as an optimisation approach
 - An optimization problem
 - minimize the loss function
 - with respect to the parameters of the score function.
 - **score function**
 - maps the raw data to class scores/labels
 - **loss function**
 - quantifies the agreement between the predicted scores and the ground truth scores/labels
 - ANN: quantifies the quality of any particular set of weights **W**
 - two components
 - The data loss computes the compatibility between the computed scores and the true labels.
 - The regularization loss is only a function of the weights

Classification

□ Suppose a supervised classification problem

- Some input data (examples, instances, cases)
 - Training data – as pairs (attribute_data_i , label_i), where
 - $i = 1, N$ ($N = \#$ of training data)
 - $\text{attribute_data}_i = (\text{attr}_{i1}, \text{attr}_{i2}, \dots, \text{attr}_{im})$, $m = \#$ attributes (characteristics, features) for an input data
 - $\text{label}_i \in \{\text{Label}_1, \text{Label}_2, \dots, \text{Label}_{\#classes}\}$
 - Test data – as (attribute_data_i), $i = 1, n$ ($n = \#$ of testing data).
- Determine
 - An unknown function that maps inputs (features) into outputs (labels)
 - Output (label/class/value/score) associated to a new data by using the learnt function

□ Quality of learning

- Accuracy/Precision/Recall/etc
 - does not reflect the learnt decision model
- A loss function
 - Expresses (encodes) the learnt model
 - Difference between desired (D) and computed (C) output
 - L_2 norm - Quadratic cost (*mean squared error*) $\sum \| D - C \|^2$
 - L_1 norm $\sum | D - C |$
 - SVM loss (hinge loss, max-margin loss) $\sum_i \sum_{j, j \neq y_i} \max(C_j - D_{y_i} + \Delta, 0)$
 - Softmax loss $\sum_i [-\ln(\exp(D_{y_i}) / \sum_{j, j \neq y_i} \exp(C_j))]$
 - Cross-entropy - $\sum [D \ln C + (1 - D) \ln(1 - C)] / n$

Classifiers

□ Several important mappings

- Constant $f(x) = c$
- Step $f(x) = a$, if $x < \theta$
 b , otherwise
- Linear $f(x) = a x + b$
- Sigmoid $\sigma(x) = 1/(1+e^{-x})$ (avoid it in a Conv NN)
- Hyperbolic tangent function $\tanh(x) = 2\sigma(2x) - 1$
- Rectified linear neuron/unit (ReLU) $f(x) = \max(0, x)$
- Leak ReLU (Parametric rectifier) $f(x) = \max(\alpha x, x)$
- Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$
- Exponential linear units (ELU) $f(x) = x$, if $x > 0$
 $\alpha (\exp(x) - 1)$, if $x \leq 0$

A linear classifier

$$f(x, w) = w \cdot x + b,$$
$$w \in \mathbb{R}^{\text{#classes} \times \text{#features}}$$
$$x \in \mathbb{R}^{\text{#features} \times 1}$$
$$b \in \mathbb{R}^{\text{#classes}}$$

A non linear classifier

$$f(x, w) = w_2 \max(0, w_1 \cdot x + b_1) + b_2,$$
$$w_1 \in \mathbb{R}^{\text{PARAM} \times \text{#features}}$$
$$x \in \mathbb{R}^{\text{#features} \times 1}$$
$$b_1 \in \mathbb{R}^{\text{PARAM}}$$
$$w_2 \in \mathbb{R}^{\text{#classes} \times \text{PARAM}}$$
$$b_2 \in \mathbb{R}^{\text{#classes}}$$

Classical ANN

- Architectures – special graphs with nodes placed on layers
 - Layers
 - Input layer – size = input's size (#features)
 - Hidden layers – various sizes (#layers, # neurons/layer)
 - Output layers – size = output size (e.g. # classes)
 - Topology
 - Full connected layers (one-way connections, recurrent connections)
- Mechanism
 - Neuron activation
 - Constant, step, linear, sigmoid
 - Cost & Loss function → smooth cost function (depends on w&b)
 - Difference between desired (D) and computed © output
 - Quadratic cost (*mean squared error*)
 - $\sum \| D - C \|^2 / 2n$
 - Cross-entropy
 - $-\sum [D \ln C + (1 - D) \ln(1 - C)] / n$
 - Learning algorithm
 - Perceptron rule
 - Delta rule (Simple/Stochastic Gradient Descent)

Convolutional Neural Networks

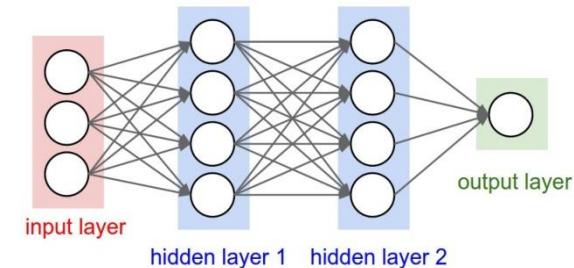
- ❑ More layers (< 10)
 - Wide NNs

- ❑ More nodes/layer

- ❑ Topology of connections

- Regular NNs → fully connected
- Conv NNs → partially connected

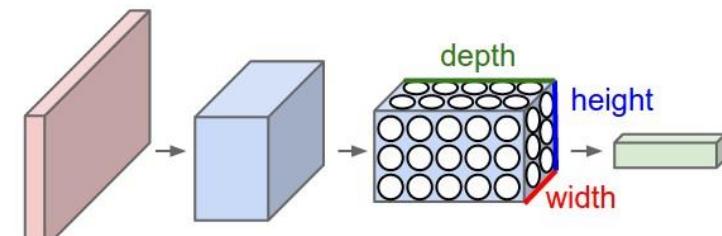
- ❑ connect each neuron to only a local region of the input volume



- ❑ Topology of layers

- Regular NNs → linear layers

- Conv NNs → 2D/3D layers (width, height, depth)



Conv NNs

❑ Layers of a Conv NN

- Convolutional Layer → feature map
 - ❑ Convolution
 - ❑ Activation (thresholding)
- Pooling/Aggregation Layer → size reduction
- Fully-Connected Layer → answer

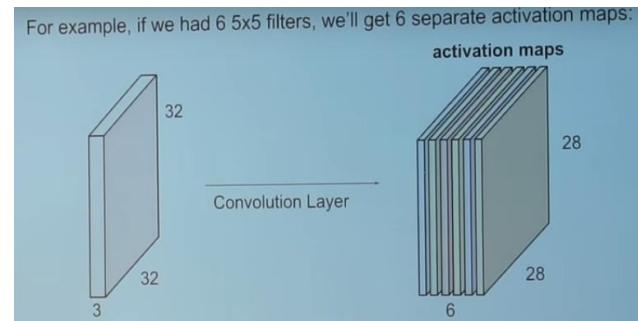
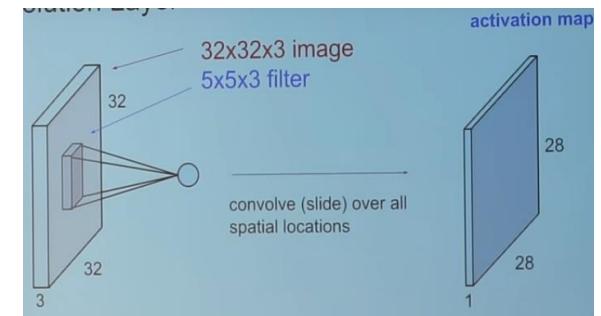
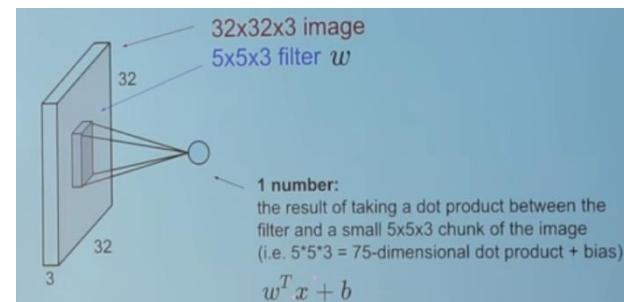
Conv NNs

□ Convolutional layer

- Aim
 - *learn data-specific kernels*
- Filters or Local receptive fields or Kernels
 - content
 - a little (square) window on the input pixels
 - How it works?
 - slide the local receptive field across the entire input image
 - Size
 - Size of field/filter (F)
 - Stride (S)
 - Learning process
 - each hidden neuron has
 - FxF shared weights connected to its local receptive field
 - a shared bias
 - an activation function
 - each connection learns a weight
 - the hidden neuron learns an overall bias as well
 - all the neurons in the first hidden layer detect exactly the same feature (just at different locations in the input image) → map from input to the first hidden layer = feature map / activation map

Conv NNs

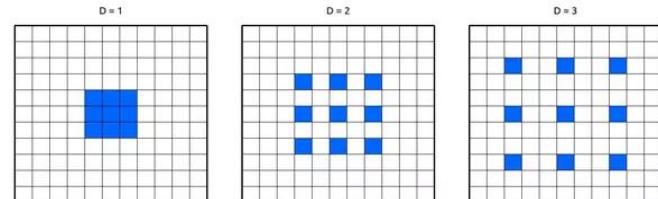
- Convolutional Layer – How does it work?
 - Take an input I (example, instance, data) of various dimensions
 - A signal \rightarrow 1D input (I_{length})
 - a grayscale image \rightarrow 2D input ($I_{\text{Width}} \& I_{\text{Height}}$)
 - an RGB image \rightarrow 3D input ($I_{\text{Width}}, I_{\text{Height}} \& I_{\text{Depth}} = 3$)
 - Consider a set of filters (kernels) $F_1, F_2, \dots, F_{\# \text{filters}}$
 - A filter must have the same # dimensions as the input
 - A signal \rightarrow 1D filter
 - $F_{\text{length}} << I_{\text{length}}$
 - a grayscale image \rightarrow 2D filter
 - $F_{\text{width}} << I_{\text{width}} \& F_{\text{height}} << I_{\text{height}}$
 - an RGB image \rightarrow 3D filter
 - $F_{\text{width}} << I_{\text{width}} \& F_{\text{height}} << I_{\text{height}} \&$
 - $F_{\text{depth}} = I_{\text{Depth}} = 3$
 - Apply each filter over the input
 - Overlap filter over a window of the input
 - Stride
 - Padding
 - Multiply the filter and the window
 - Store the results in an activation map
 - # activation maps = # filters
 - Activate all the elements of each activation map
 - ReLU or other activation function



Conv NNs

□ Convolutional layer

- CONV layer's parameters consist of a set of learnable filters
 - Contiguous filters (without spaces between each cell)
 - Dilated filters (with spaces between each cell)
 - apply the same filter at different ranges using different dilation factors

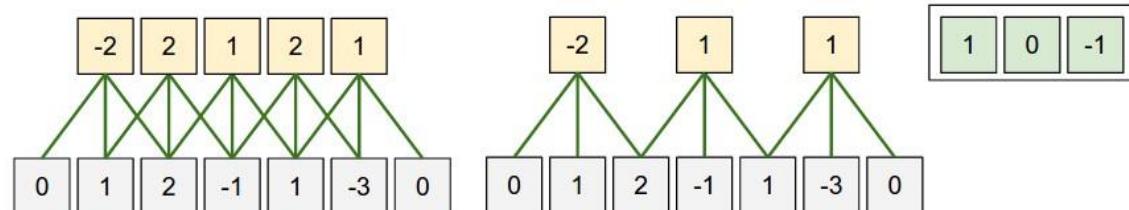


- By sliding a filter → a 2-dimensional activation map
 - All filters → depth 2D activation maps
- Partial connected neurons
 - The spatial extent of this connectivity is a hyperparameter called the **receptive field** of the neuron (filter size)
 - The connections are local in space (along width and height), but always full along the entire depth of the input volume

Conv NNs

□ Convolutional layer - Hyperparameters

- input volume size N (L or $W \& H$ or $W \& H \& D$)
- size of zero-padding of input volume P (P_L or $P_W \& P_H$ or $P_W \& P_H$)
- the receptive field size (filter size) F ($F_L, F_W \& F_H, F_W \& F_H \& F_D$)
- stride of the convolutional layer S ($S_L, S_W \& S_H, S_W \& S_H$)
- # of filters (K)
 - depth of the output volume
- # neurons of an activation map = $(N + 2P - F)/S + 1$
- Output size
 - $K * [(N + 2P - F)/S + 1]$
- $N = L = 5, P = 1,$
 - $F = 3, S = 1$
 - $F = 3, S = 2$



Conv NNs

- Convolutional layer - ImageNet challenge in 2012 (Alex Krizhevsky)
<http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
 - Input images of size [227x227x3]
 - $F=11, S=4, P=0, K = 96 \rightarrow$ Conv layer output volume of size [55x55x96]
 - $55*55*96 = 290,400$ neurons in the first Conv Layer
 - each has $11*11*3 = 363$ weights and 1 bias.
 - $290400 * 364 = 105,705,600$ parameters on the first layer

Conv NNs

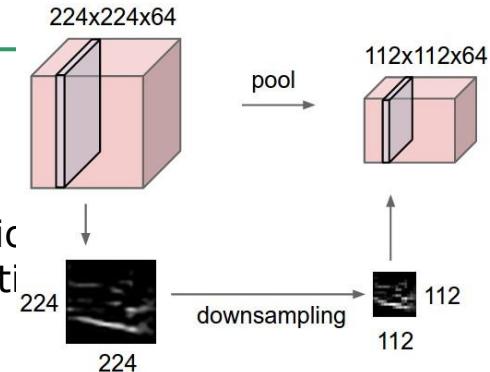
- Convolutional layer - parameter sharing scheme
 - constrain the neurons in each depth slice to use the same weights and bias
 - detect exactly the same feature, just at different locations in the input image
 - convolutional networks are well adapted to the translation invariance of images
 - Example
 - 96 unique set of weights (one for each depth slice),
 - for a total of $96 \times 11 \times 11 \times 3 = 34,848$ unique weights,
 - 34,944 parameters (+96 biases).
 - if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can in each depth slice be computed as a **convolution** of the neuron's weights with the input volume → the name: Convolutional Layer
 - Set of weights = filter (kernel)
 - Can be applied, but are image-dependent
 - faces that have been centered in the image
 - A Convolutional Layer without parameter sharing → **Locally-Connected Layer**

Conv NNs

□ Pooling layer

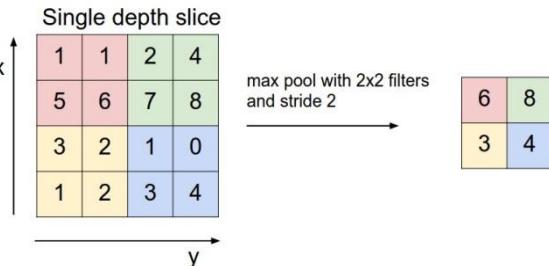
■ Aim

- progressively reduce the spatial size of the representation
 - to reduce the amount of parameters and computation
 - to also control overfitting
- downsample the spatial dimensions of the input.
- simplify the information in the output from the convolutional layer



■ How it works

- takes each feature map output from the convolutional layer and creates a condensed feature map
- each unit in the pooling layer may summarize a region of the input
- apply pooling filters to each feature map separately
 - Pooling filter size (spatial extent of pooling) P
 - Pooling filter stride PS
 - No padding
- resizes it spatially, using
 - the MAX operation
 - the average operation
 - L2-norm operation (square root of the sum of the squares of the activations in the 2×2 region)
 - L_p norm L_p : $\text{sqrt}(\text{ord}_p)(X^p)$
 - Log prob PROB: $1/b \log (\sum(\exp(bX)))$



Conv NNs

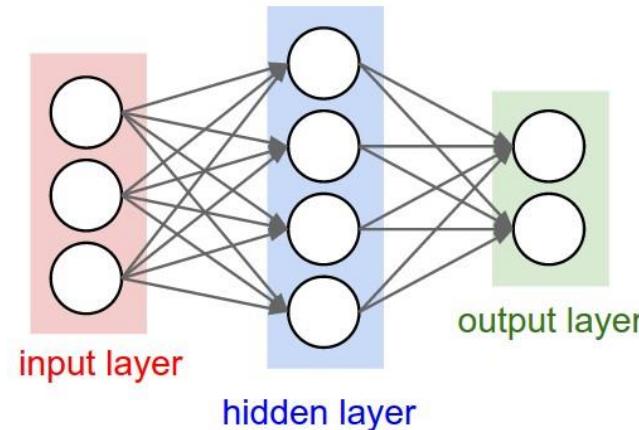
- Pooling layer
 - Size conversion
 - Input:
 - $K \times N$
 - Output:
 - $K \times [(N - PF)/PS + 1]$
 - Remark
 - introduces zero parameters since it computes a fixed function of the input
 - note that it is not common to use zero-padding for Pooling layers
 - pooling layer with $PF=3, PS=2$ (also called overlapping pooling), and more commonly $PF=2, PS=2$
 - pooling sizes with larger filters are too destructive
 - keep track of the index of the max activation (sometimes also called *the switches*) so that gradient routing is efficient during backpropagation

Conv NNs

❑ Fully-connected layer

- Neurons have full connections to all inputs from the previous layer

- Various activations
 - ❑ ReLU (often)



Conv NNs

□ CNN architectures

- INPUT -> [[CONV -> RELU]*N -> POOL?] *M -> [FC -> RELU]*K -> FC
- Most common:
 - INPUT -> FC \longleftrightarrow a linear classifier
 - INPUT -> CONV -> RELU -> FC
 - INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC.
 - INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC
 - a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation

Conv NNs

□ Remarks

- Prefer a stack of small filter CONV to one large receptive field CONV layer
 - Pro:
 - Non-linear functions
 - Few parameters
 - Cons:
 - more memory to hold all the intermediate CONV layer results
- Input layer size → divisible by 2 many times
- Conv layers → small filters
 - $S \geq 1$
 - $P = (F - 1) / 2$
- Pool layers
 - $F \leq 3, S = 2$

Conv NNs

□ Output layer

- Multiclass SVM
 - Largest score indicates the correct answer
- Softmax (normalized exponential function)
 - Largest probability indicates the correct answer
 - converts raw scores to probabilities
 - "squashes" a #classes-dimensional vector **z** of arbitrary real values to a #classes-dimensional vector $\sigma(z)$ of real values in the range (0, 1) that add up to 1
 - $\sigma(z)_j = \exp(z_j) / \sum_{k=1.. \# \text{classes}} \exp(z_k)$

Conv NNs

□ Common architectures

- LeNet (Yann LeCun, 1998) - <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>
 - A conv layer + a pool layer
- AlexNet (Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, 2012)
<http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
 - More conv layers + more pool layers
- ZF Net (Matthew Zeiler and Rob Fergus, 2013)
<https://arxiv.org/pdf/1311.2901.pdf>
 - AlexNet + optimisation of hyper-parameters
- GoogleLeNet (Christian Szegedy et al., 2014)
<https://arxiv.org/pdf/1409.4842.pdf>
 - *Inception Module* that dramatically reduced the number of parameters in the network (AlexNet 60M, GoogleLeNet 4M) <https://arxiv.org/pdf/1602.07261.pdf>
 - uses Average Pooling instead of Fully Connected layers at the top of the ConvNet → eliminating parameters
- VGGNet (Karen Simonyan and Andrew Zisserman, 2014)
<https://arxiv.org/pdf/1409.1556.pdf>
 - 16 Conv/FC layers (FC → a lot more memory; they can be eliminated)
 - pretrained model is available for plug and play use in Caffe
- ResNet (Kaiming He et al., 2015) <https://arxiv.org/pdf/1512.03385.pdf>
(Torch)
 - *skip connections*
 - batch normalization

Conv NNs

- Reducing overfitting
 - increasing the amount of training data
 - Artificially expanding the training data
 - Rotations, adding noise,
 - reduce the size of the network
 - Not recommended
 - regularization techniques
 - Effect:
 - the network prefers to learn small weights, all other things being equal. Large weights will only be allowed if they considerably improve the first part of the cost function
 - a way of compromising between finding small weights and minimizing the original cost function (when λ is small we prefer to minimize the original cost function, but when λ is large we prefer small weights)
 - Give importance to all features
 - $X = [1, 1, 1, 1]$
 - $W_1 = [1, 0, 0, 0]$
 - $W_2 = [0.25, 0.25, 0.25, 0.25]$
 - $W_1^T X = W_2^T X = 1$
 - $L1(W_1) = 0.25 + 0.25 + 0.25 + 0.25 = 1$
 - $L1(W_2) = 1 + 0 + 0 + 0 = 1$

Conv NNs

□ Reducing overfitting - regularization techniques

■ Methods

- L1 regularisation – add the sum of the absolute values of the weights $C = C_0 + \lambda/n \sum |w|$
 - the weights shrink by a constant amount toward 0
 - Sparsity (feature selection – more weights are 0)
- *weight decay (L2 regularization)* - add an extra term to the cost function (the *L2 regularization term* = the sum of the squares of all the weights in the network = $\lambda/2n \sum w^2$): $C = C_0 + \lambda/2n \sum w^2$
 - the weights shrink by an amount which is proportional to w
- Elastic net regularisation
 - $\lambda_1|w| + \lambda_2 w^2$
- Max norm constraints (clapping)
- Dropout - modify the network itself (<http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>)
 - Some neurons are temporarily deleted
 - propagate the input and backpropagate the result through the modified network
 - update the appropriate weights and biases.
 - repeat the process, first restoring the dropout neurons, then choosing a new random subset of hidden neurons to delete

Improve NN's performance

- Cost functions → loss functions
- Regularisation
- Initialisation of weights
- NN's hyper-parameters

Improve NN's performance

- Cost functions → loss functions
 - Possible cost functions
 - Quadratic cost
 - $\frac{1}{2n} \sum_x \|D - C\|^2$
 - Cross-entropy loss (negative log likelihood)
 - $-\frac{1}{n} \sum_x [D \ln C + (1 - D) \ln(1 - C)]$
 - Optimizing the cost function
 - Stochastic gradient descent by backpropagation
 - Hessian technique
 - Pro: it incorporates not just information about the gradient, but also information about how the gradient is changing
 - Cons: the sheer size of the Hessian matrix
 - Momentum-based gradient descent
 - Velocity & friction

Improve NN's performance

□ Initialisation of weights

- Pitfall
 - all zero initialization
- Small random numbers
 - $W = 0.01 * \text{random}(D, H)$
- Calibrating the variances with $1/\sqrt{\# \text{Inputs}}$
 - $w = \text{random}(\# \text{Inputs}) / \sqrt{\# \text{Inputs}}$
- Sparse initialization
- Initializing the biases
- In practice
 - $w = \text{random}(\# \text{Inputs}) * \sqrt{2.0 / \# \text{Inputs}}$

Improve NN's performance

□ NN's hyper-parameters*

- Learning rate η
 - Constant rate
 - Not-constant rate
- Regularisation parameter λ
- Mini-batch size

*see Bengio's papers: <https://arxiv.org/pdf/1206.5533v2.pdf> and <http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf> or
Snock's paper <http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf>

Improve NN's performance

- NN's hyper-parameters
 - Learning rate η
 - Constant rate
 - Not-constant rate
 - Annealing the learning rate
 - Second order methods
 - Per-parameter adaptive learning rate methods

Improve NN's performance

- NN's hyper-parameters - Learning rate η
 - Not-constant rate
 - Annealing the learning rate
 - **Step decay**
 - Reduce the learning rate by some factor every few epochs
 - $\eta = \eta * \text{factor}$
 - Eg. $\eta = \eta * 0.5$ every 5 epochs
 - Eg. $\eta = \eta * 0.1$ every 20 epochs
 - Exponential decay
 - $a=a_0\exp(-kt)$,
where a_0 , k are hyperparameters and t is the iteration number (but you can also use units of epochs).
 - $1/t$ decay
 - $a=a_0/(1+kt)$
where a_0 , k are hyperparameters and t is the iteration number.

Improve NN's performance

- NN's hyper-parameters - Learning rate η
 - Not-constant rate
 - Second order methods
 - Newton's method (Hessian)
 - *quasi-Newton* methods
 - L- BGFS (Limited memory Broyden–Fletcher–Goldfarb–Shanno)
 - https://static.googleusercontent.com/media/research.google.com/ro//archive/large_deep_networks_nips2012.pdf
 - <https://arxiv.org/pdf/1311.2115.pdf>

Improve NN's performance

- NN's hyper-parameters - Learning rate η
 - Not-constant rate
 - Per-parameter adaptive learning rate methods
 - Adagrad
 - <http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>
 - RMSprop
 - http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
 - Adam
 - <https://arxiv.org/pdf/1412.6980.pdf>

Tools

- Keras
 - NN API
 - <https://keras.io/>
 - + Theano (machine learning library; multi-dim arrays)
<http://www.deeplearning.net/software/theano/>
http://www.iro.umontreal.ca/~lisa/poiteurs/theano_scipy2010.pdf
 - + TensorFlow (numerical computation) <https://www.tensorflow.org/>
- Pylearn2 <http://deeplearning.net/software/pylearn2/>
 - ML library
 - + Theano
- Torch <http://torch.ch/>
 - scientific computing framework
 - Multi-dim array
 - NN
 - GPU
- Caffe
 - deep learning framework
 - Berkley



Recapitulare

■ Sisteme care învață singure (SIS)

■ Rețele neuronale artificiale

- Modele computaționale inspirate de rețelele neuronale artificiale
- Grafe speciale cu noduri așezate pe straturi
 - Strat de intrare → citește datele de intrare ale problemei de rezolvat
 - Strat de ieșire → furnizează rezultate problemei date
 - Strat(uri) ascunse → efectuează calcule
- Nodurile (neuronii)
 - Au intrări ponderate
 - Au funcții de activare (liniare, sigmoidale, etc)
 - necesită antrenare → prin algoritmi precum:
 - Perceptron
 - Scădere după gradient
- Algoritm de antrenare a întregii RNA → Backpropagation
 - Informația utilă se propagă înainte
 - Eroarea se propagă înapoi

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

C. Sisteme inteligente

- Sisteme care învață singure
 - Arbori de decizie
 - Rețele neuronale artificiale
 - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit și legături utile

- capitolul 15 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*
- Documentele din directorul *12_svm* și *13_GP*

❑ Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:

■ Conf. Dr. Mihai Oltean –
www.cs.ubbcluj.ro/~moltean

■ Lect. Dr. Crina Groșan -
www.cs.ubbcluj.ro/~cgrosan

■ Prof. Dr. Horia F. Pop -
www.cs.ubbcluj.ro/~hfpop

Inteligentă Artificială Generativă - LLMs

AI Curs 7 – 10.04.2024

Capitole

- Întroducere în Generative AI – Alexandru Manole
- Embeddings – Răzvan Petec
- Modele Markov – Adrian Iurian

Despre mine



- Student la doctorat în anul I la UBB FMI
- Domenii de interes: Computer Vision, Multitask Models, Image Generation
- alexandru.manole@ubbcluj.ro

Despre voi



- <https://www.menti.com/>
- cod: **8415 5783**

Introducere în Generative AI

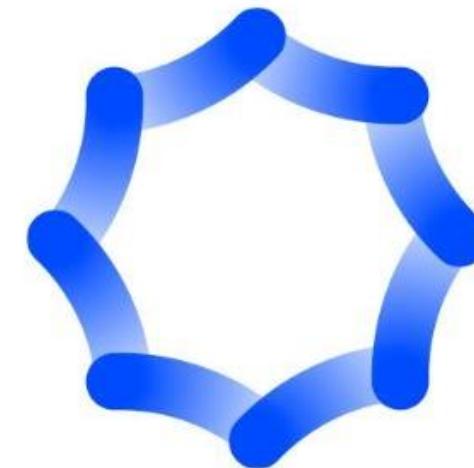
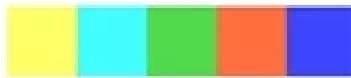


ChatGPT

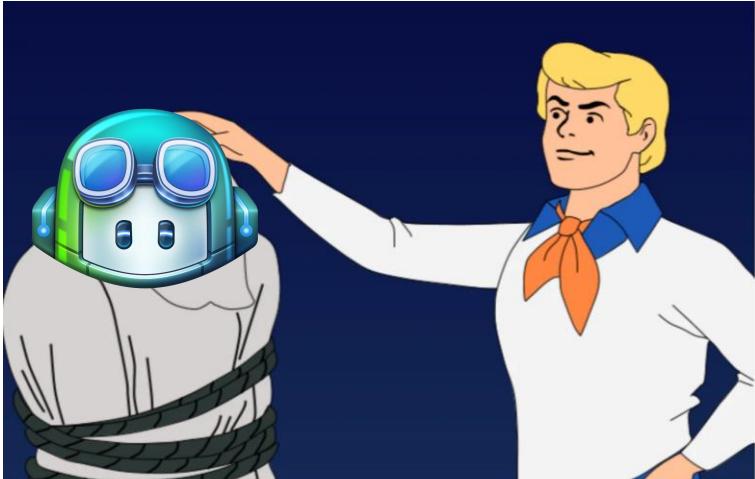


Bard

DALL·E

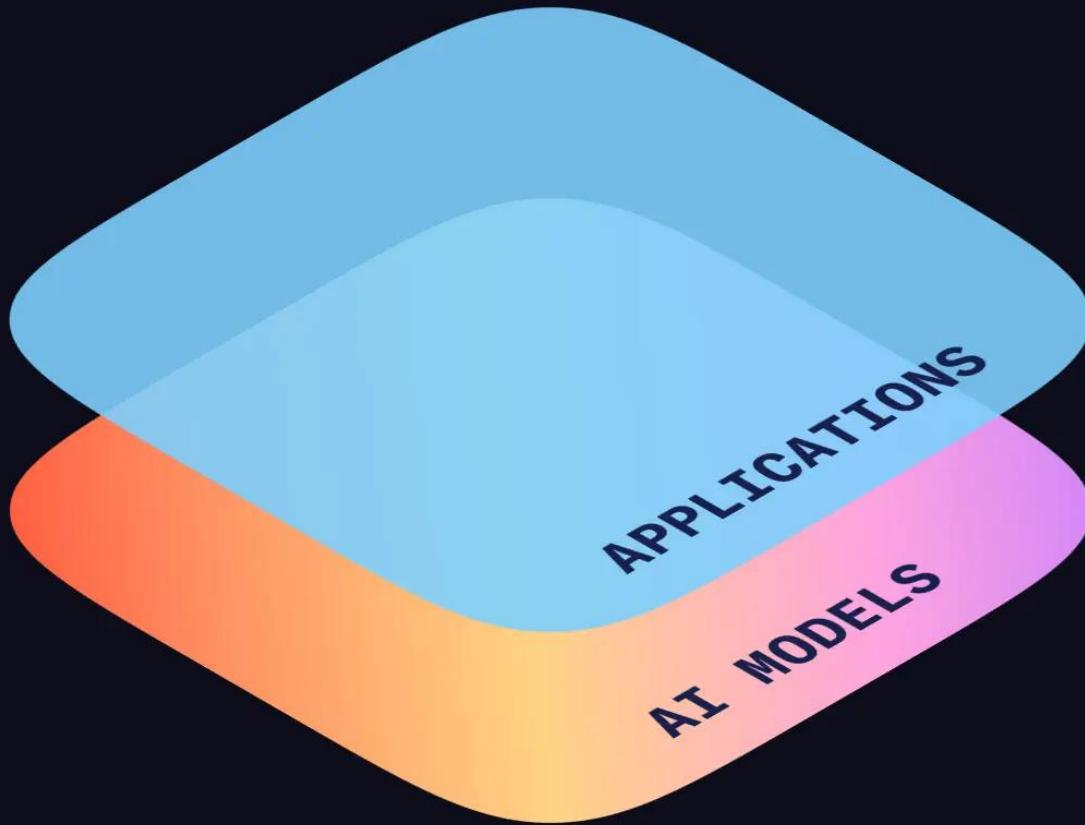


Introducere în Generative AI



- ChatGPT, Bard, Copilot sunt sisteme software / tool-uri complexe
- Aceste au în spate modele inteligente generative

The Generative Tech Stack



- TOOLS THAT COLLABORATE WITH AI MODELS.

WORKFLOWS, SECURITY,
NETWORK EFFECTS, PAYMENTS, ETC.
(10,000'S OF THESE)

- MODELS GENERATE UNIQUE AND NOVEL OUTPUT.

GPT-3, STABLE DIFFUSION,
CUSTOM DATA SETS, ETC.
(1,000'S OF THESE)

LANGUAGE MODEL SIZES TO MAR/2023

- BERT 340M
- GPT-1 117M
- GPT-2 1.5B

T5

11B Plato-XL

11B Macaw

Cohere

52.4B Megatron-11B

ruGPT-3

GPT-3 175B

Jurassic-1 178B

GPT-NeoX-20B 20B

Luminous 200B

CM3 13B

VLM-4 10B

mGPT 13B

BLOOM 176B

Atlas 11B

Flan-T5 11B

Kosmos-1 1.6B*

NLLB 54.5B

GLM-130B ChatGLM-6B

OPT-175B BB3 175B

OPT-IML 175B

MOSS 20B*

GPT-4 Undisclosed *

LLaMA 65B*

Alpaca 7B

Toolformer 6.7B*

YaLM 100B

UL2 20B

NOOR 10B

SeeKeR 2.7B

Z-Code++ 710M*

Gato 1.2B

FIM 20B *

PaLI 17B

Galactica 120B

AlexaTM 10B *

WeLM 200M

VIMA 10B *

- Parameters
- AI lab/group
- Available
- Closed
- * Chinchilla scale

Beeswarm/bubble plot, sizes linear to scale. Selected highlights only. *Chinchilla scale means T:P ratio >15:1. <https://lifearchitect.ai/chinchilla/> Alan D. Thompson. March 2023. <https://lifearchitect.ai/>



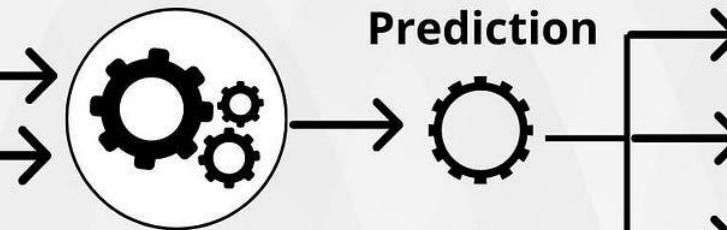
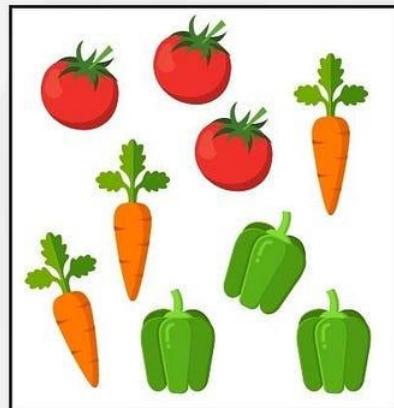
LifeArchitect.ai/models

Învățare supervizată

SUPERVISED LEARNING

Supervised machine learning is a branch of artificial intelligence that focuses on training models to make predictions or decisions based on labeled training data.

Labeled Data

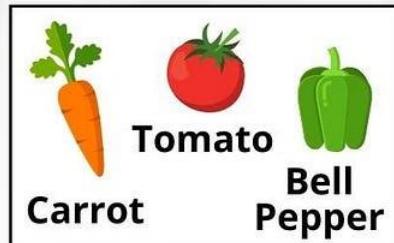


Carrot

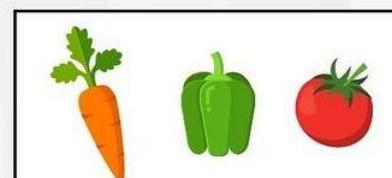
Bell Pepper

Tomato

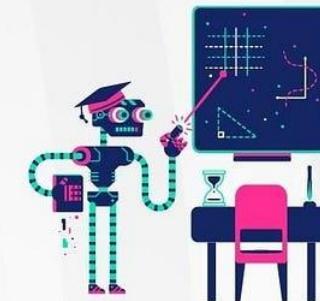
Labels



DatabaseTown

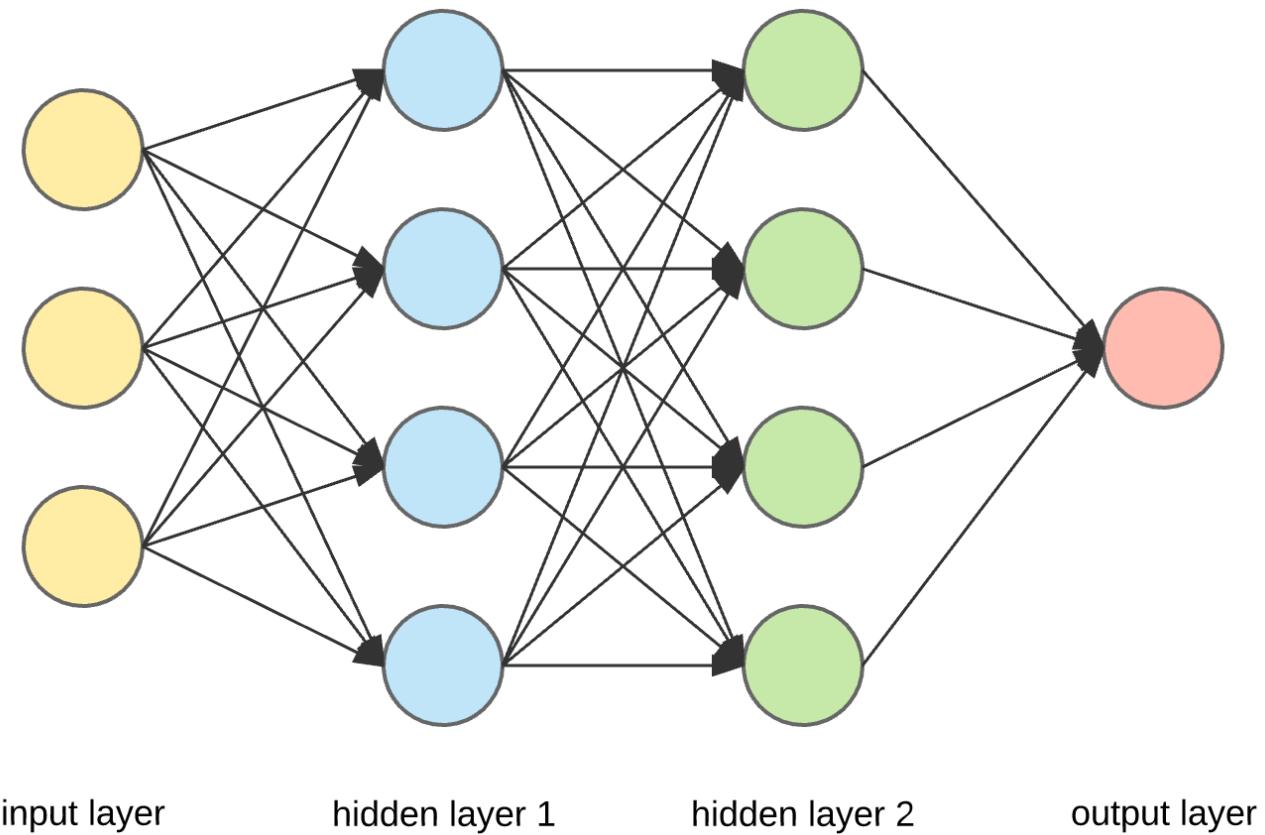


Test Data

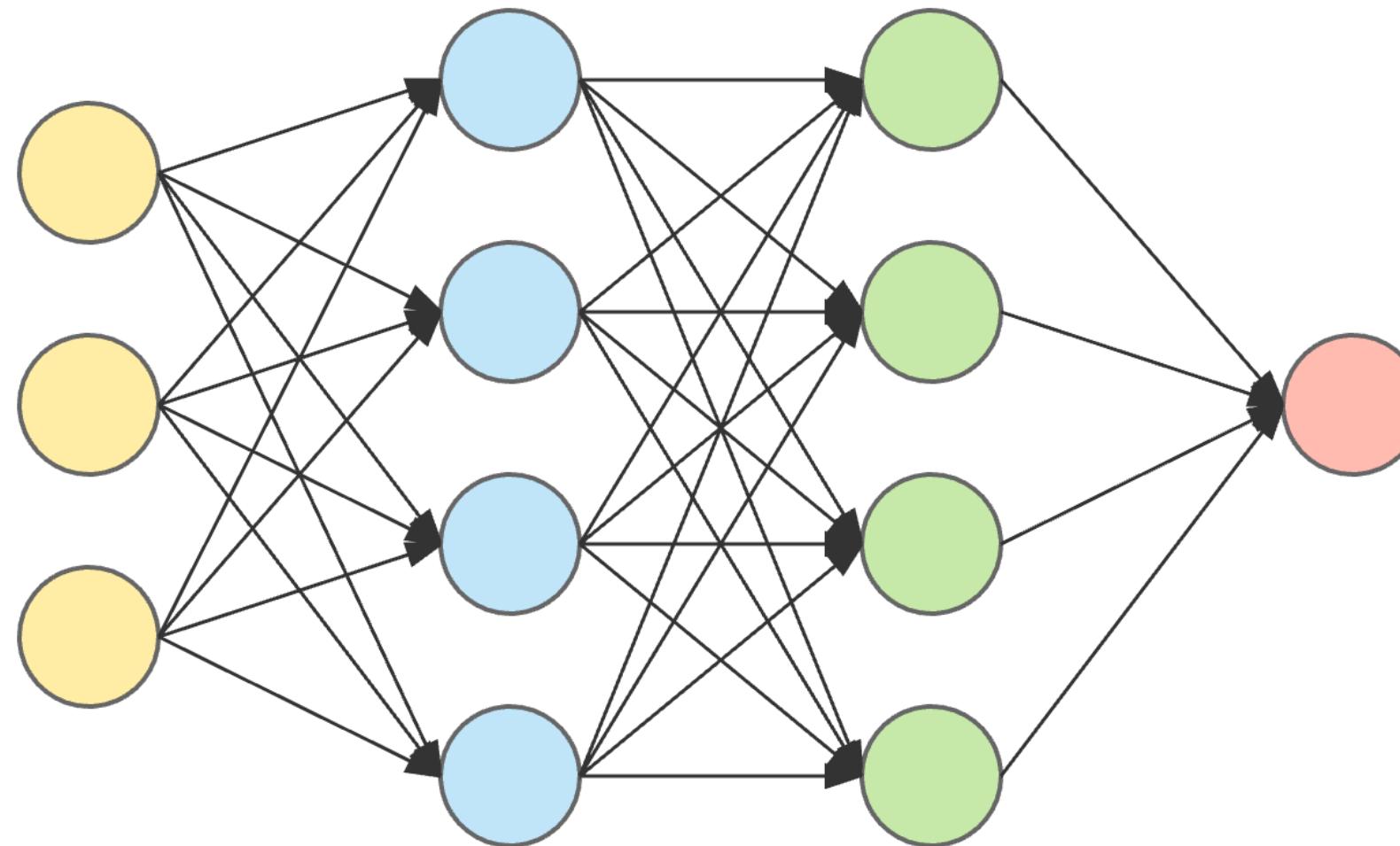


Elemente AI / Machine Learning:

1. Algoritm / Model intelligent
2. Date
 - Date de intrare
 - Date de ieșire



Cum se numește acest algoritm inteligent:
menti.com cod: **8415 5783**

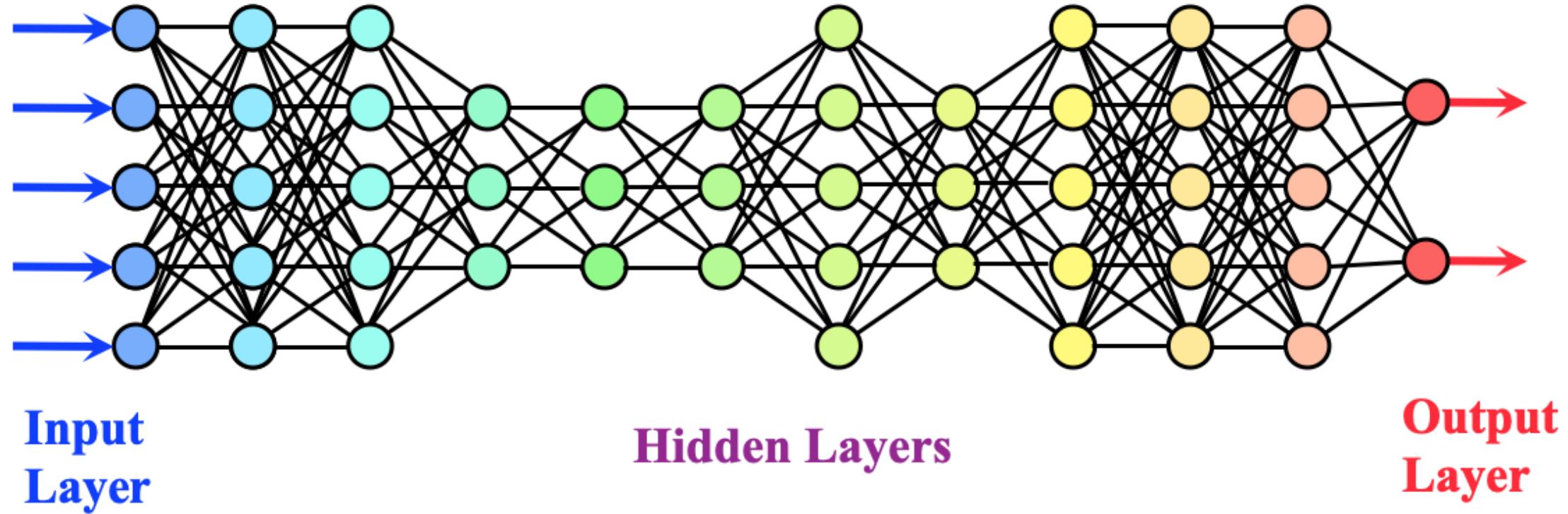


input layer

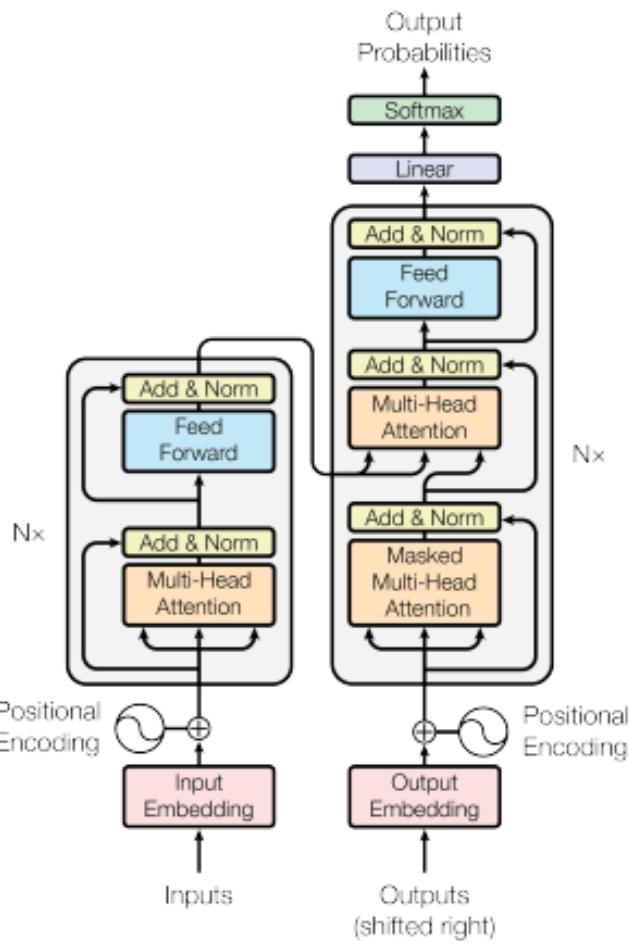
hidden layer 1

hidden layer 2

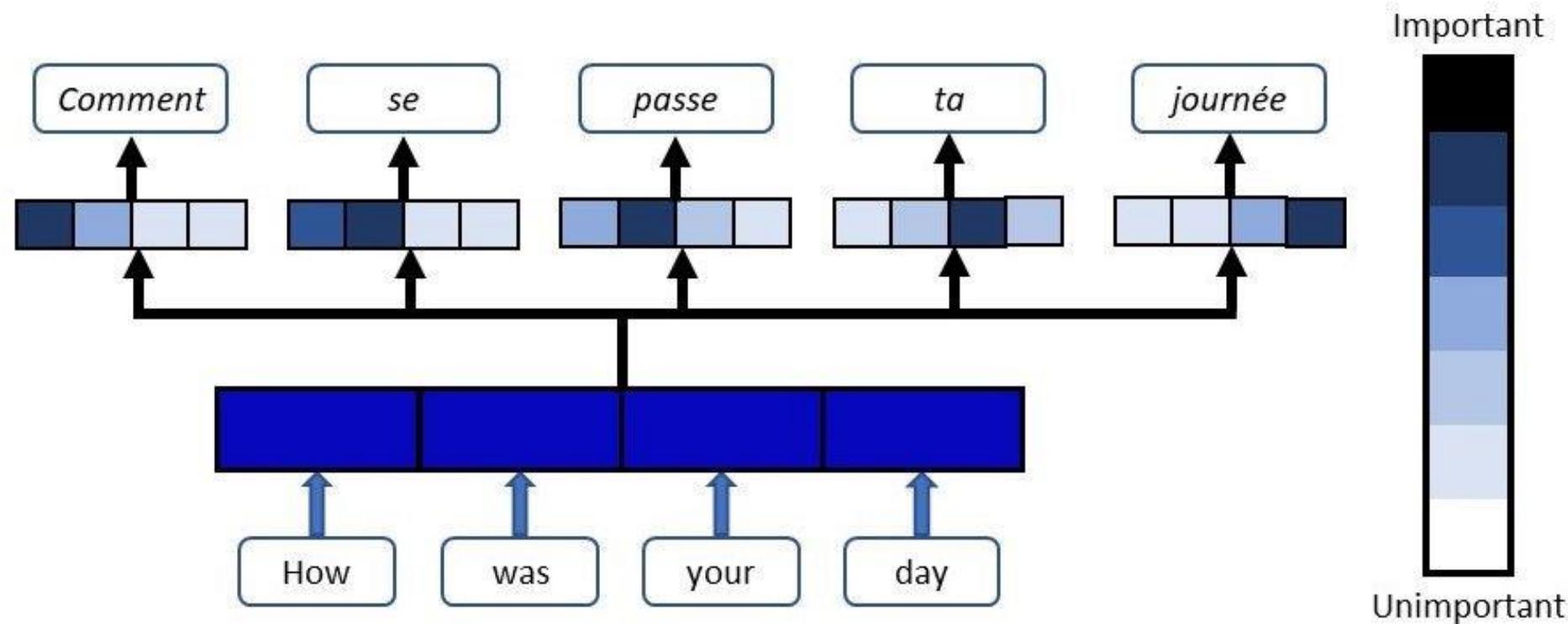
output layer



Transformers



Attention



LANGUAGE MODEL SIZES TO MAR/2023

- BERT 340M
- GPT-1 117M
- GPT-2 1.5B

T5

Megatron-11B

ruGPT-3

GPT-3
175B

Jurassic-1
178B

LaMDA
LaMDA 2
Bard
137B

Anthropic-LM

GPT-J

BlenderBot2.0

Plato-XL

Macaw

Cohere

52.4B

GPT-NeoX-20B

20B

XGLM

Luminous
200B

MT-NLG

530B

Cedille

Fairseq

Anthropic-LM

GPT-J

BlenderBot2.0

52B
RL-CAI
Claude

Gopher
280B

Chinchilla

70B*

CM3
VLM-4
mGPT

BLOOM
BLOOMZ
176B

13B

10B

13B

Atlas
Flan-T5

11B

11B

Kosmos-1 1.6B*

ChatGLM-6B

11B

GLM-130B

ChatGLM-6B

54.5B

OPT-175B

BB3

OPT-IML
175B

MOSS
20B*

GPT-4
Undisclosed

*

LLaMA
65B*

Alpaca
Toolformer

7B

YaLM
100B

NOOR

UL2
20B

PaLI
17B

Z-Code++
710M*

SeeKeR
2.7B

Gato
1.2B

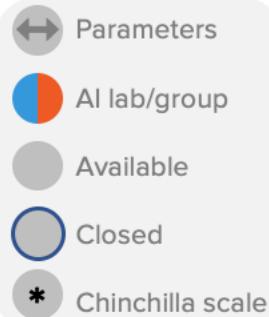
FIM
20B*

AlexaTM
WeLM

10B*

VIMA
200M

10B*



Beeswarm/bubble plot, sizes linear to scale. Selected highlights only. *Chinchilla scale means T:P ratio >15:1. <https://lifearchitect.ai/chinchilla/> Alan D. Thompson. March 2023. <https://lifearchitect.ai/>



LifeArchitect.ai/models

Exemplu algoritm intelligent

- Pathways Language Model (**PaLM**)
- Dezvoltat de Google folosit în  Bard
- **530 MILIARDE** de parametri
- menti.com cod: **8415 5783**

PaLM 2

- Dezvoltat de Google folosit în 
- **250 MILIARDE** de parametri
- Antrenat pe peste 1.3 triliarde de cuvinte

Set de date



Proces de antrenare

Modelele generative sunt

cuv 1 cuv 2 cuv3 cuv4

- Pornind de la un sir de cuvinte modelul prezice ce cuvant ar putea urma

Ce mai poate fi generat aşa?

menti.com cod: 8415 5783

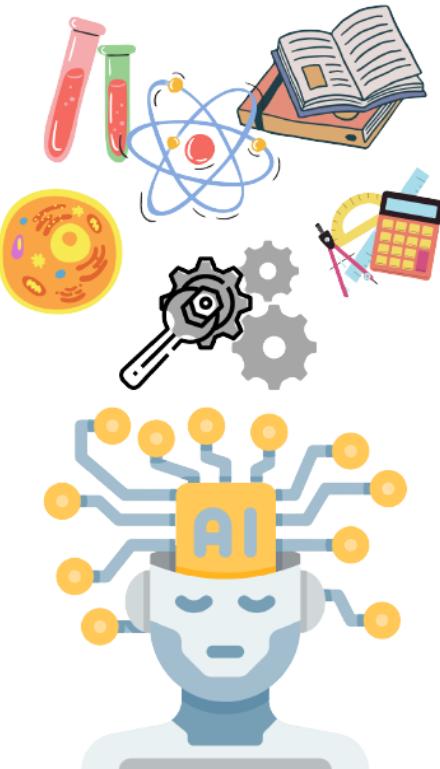


Muzică

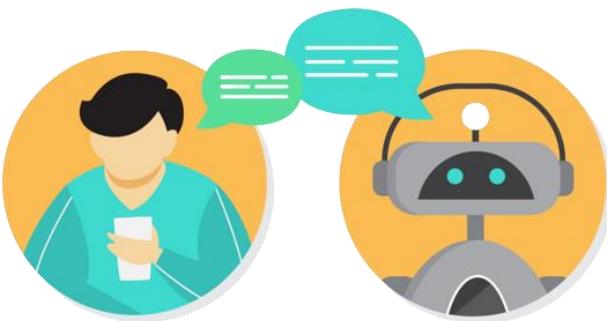
```
    ws.on("message", m => {
      let a = m.split(" ")
      switch(a[0]){
        case "connect":
          if(a[1]){
            if(clients.has(a[1])){
              ws.send("connected");
              ws.id = a[1];
            }else{
              ws.id = a[1];
              clients.set(a[1], {client: {position: {x: 0, y: 0, z: 0}}, id: a[1]});
              ws.send("connected")
            }
          }else{
            let id = Math.random().toString().slice(3, 10);
            ws.id = id;
            clients.set(id, {client: {position: {x: 0, y: 0, z: 0}}, id: id});
          }
        }
      })
    })
```

Cod

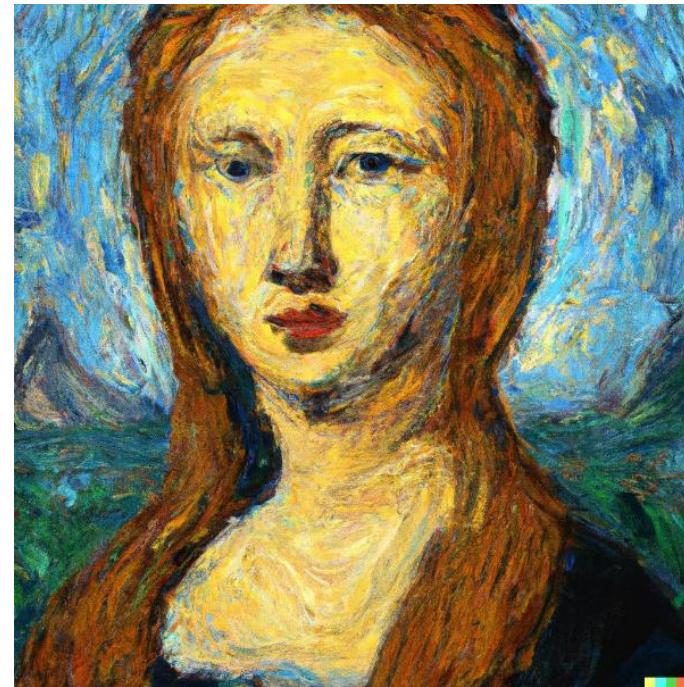
Potențial Generative AI



**Pseudo-expert
în mai multe
domenii**



**Mod interactiv de a
afla informații**



**Inspirație
creativă**

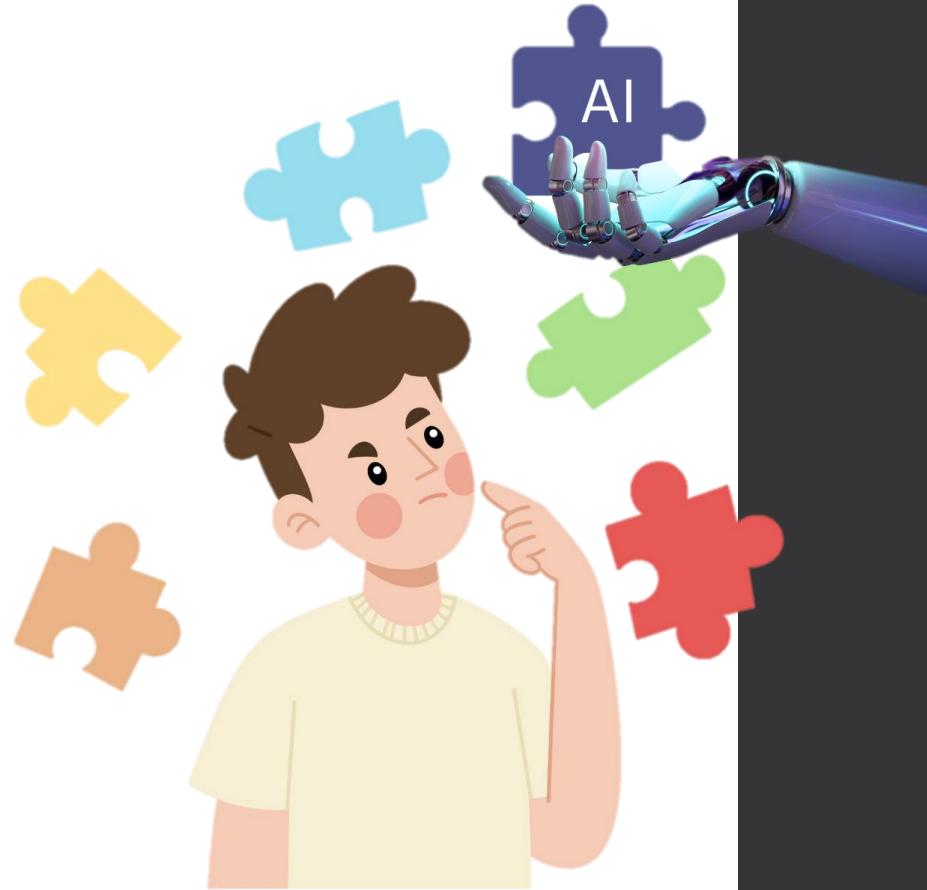
Pericole Generative AI



Greșeli, bias,
halucinare



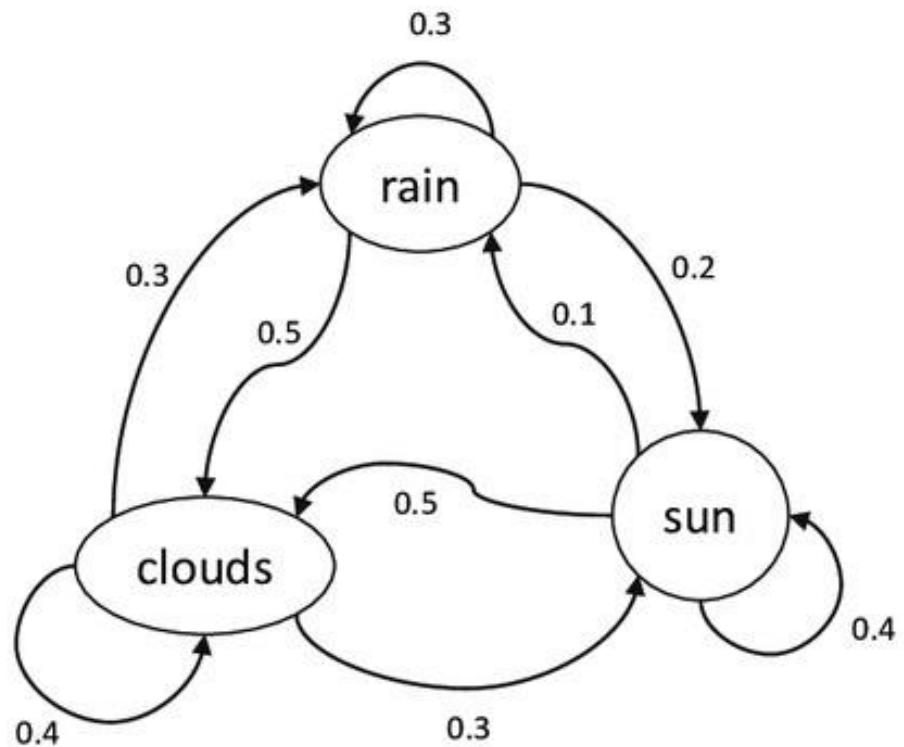
Gândire de
grup



Are potențialul de a
distrugă gândirea
critică

Sfărsit partea I

Ce valoare lipsește în tabel?



	clouds	rain	sun
clouds	?	0.3	0.3
rain	0.5	0.3	0.2
sun	0.5	0.1	0.4

Cum construim?

Altă matematică - Nichita Stănescu

Noi ştim că unu ori unu fac unu,
1 dar un inorog ori o pară
nu ştim cât face.
Ştim că cinci fără patru fac unu,
2 dar un nor fără o corabie
nu ştim cât face.
Ştim, noi ştim că opt

împărțit la opt fac unu,
3 dar un munte împărțit la o capră
nu ştim cât face.
Ştim că unu plus unu fac doi,
dar eu şi cu tine,
nu ştim, vai, nu ştim cât facem.

Markov - o stare

input: unu

output:

ori: 14.28%

fac: 28.57%

plus: 14.28%

dar: 42.85%

Altă matematică - Nichita Stănescu

Noi ştim că unu ori unu fac unu,
dar un inorog ori o pară
nu ştim cât face.
Ştim că cinci fără patru fac unu,
dar un nor fără o corabie
nu ştim cât face.
Ştim, noi ştim că opt

împărțit la opt fac unu,
dar un munte împărțit la o capră
nu ştim cât face.
Ştim că unu plus unu fac doi,
dar eu şi cu tine,
nu ştim, vai, nu ştim cât facem.

Markov - două stări

input: dar un

output:

inorog: 33.33%

nor: 33.33%

munte: 33.33%

Care poezie este scrisă de un om?

3

*Spălări împrăştiate! Înnoiţi
Arginturile mari botezătoare
Şi inima călărilor - spuziţi
De dreaptă ziua-aceasta suitoare.*

7

*Greu îmi e şi viaţa mea e iubire.
Umiliţi de nevoi şi cu plus de moarte,
Sfios vin la tine, am învăţat
Pentru că n-am nici o realitate.*

Mulțumesc!

Reprezentarea numerică a datelor în Inteligență Artificială

Funcții matematice

Cum arată o funcție matematică?

a) Cu un singur parametru și un singur rezultat ($f: \mathbb{R} \rightarrow \mathbb{R}$)

$$f(x) = x^2 + 2x + 1 \quad f(3) = 16$$

b) Cu mai mulți parametri și un singur rezultat ($f: \mathbb{R}^2 \rightarrow \mathbb{R}$)

$$f(x, y) = x^2 + y^2 + xy \quad f(2, 3) = 19$$

c) Cu un singur parametru și mai multe rezultate ($f: \mathbb{R} \rightarrow \mathbb{R}^2$)

$$f(x) = (x^3 + 3, x - 4) \quad f(2) = (11, -2)$$

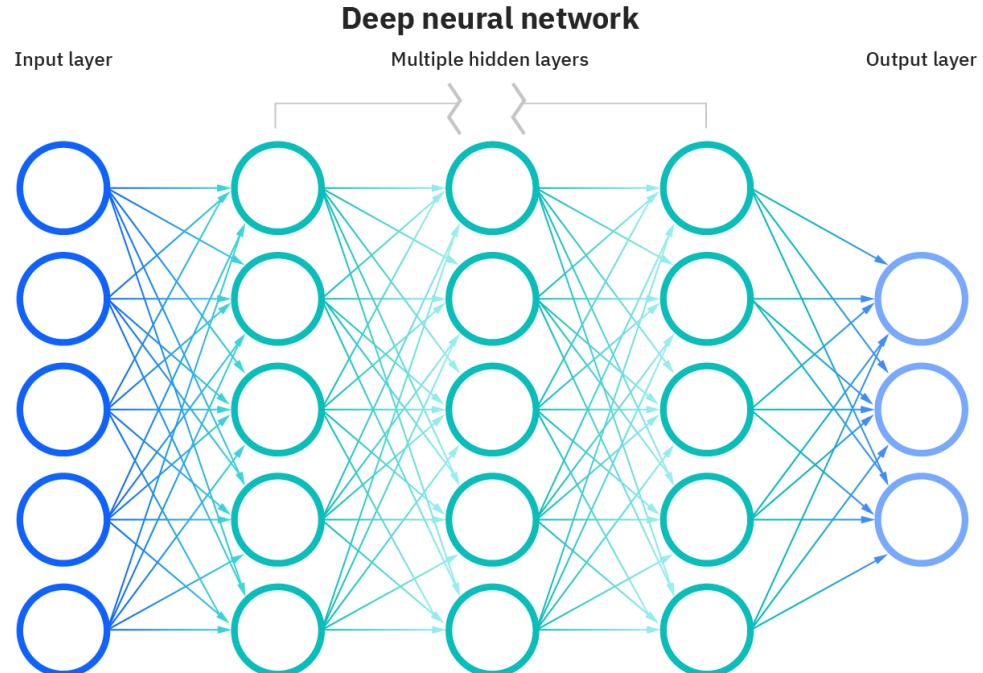
d) Cu mai mulți parametri și mai multe rezultate ($f: \mathbb{R}^n \rightarrow \mathbb{R}^k$)

$$f(x_1, x_2, \dots, x_n) = (f_1(x_1, x_2, \dots, x_n), \dots, f_k(x_1, x_2, \dots, x_n))$$

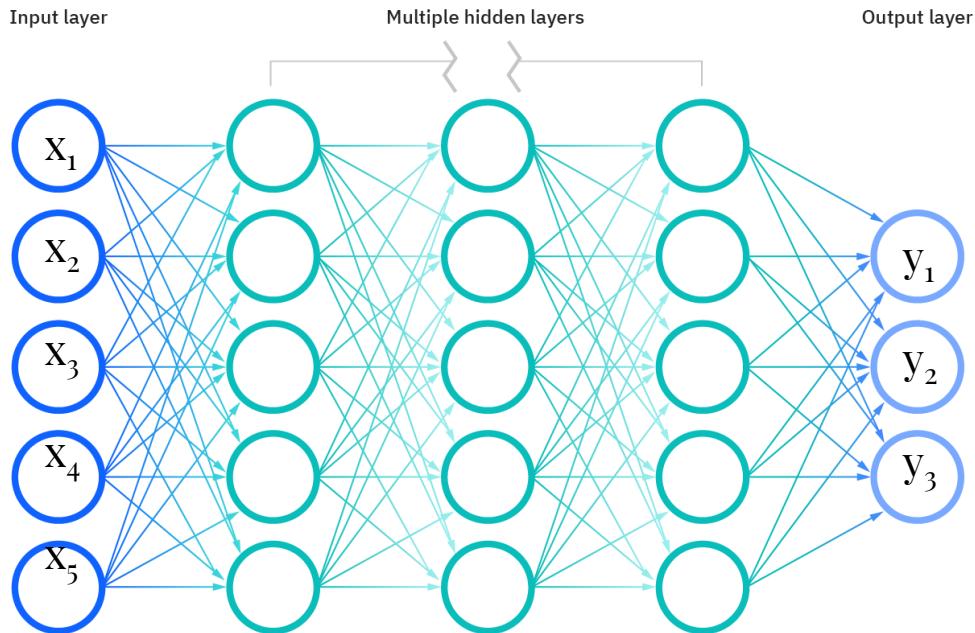
Legătura cu Modelele de Machine Learning

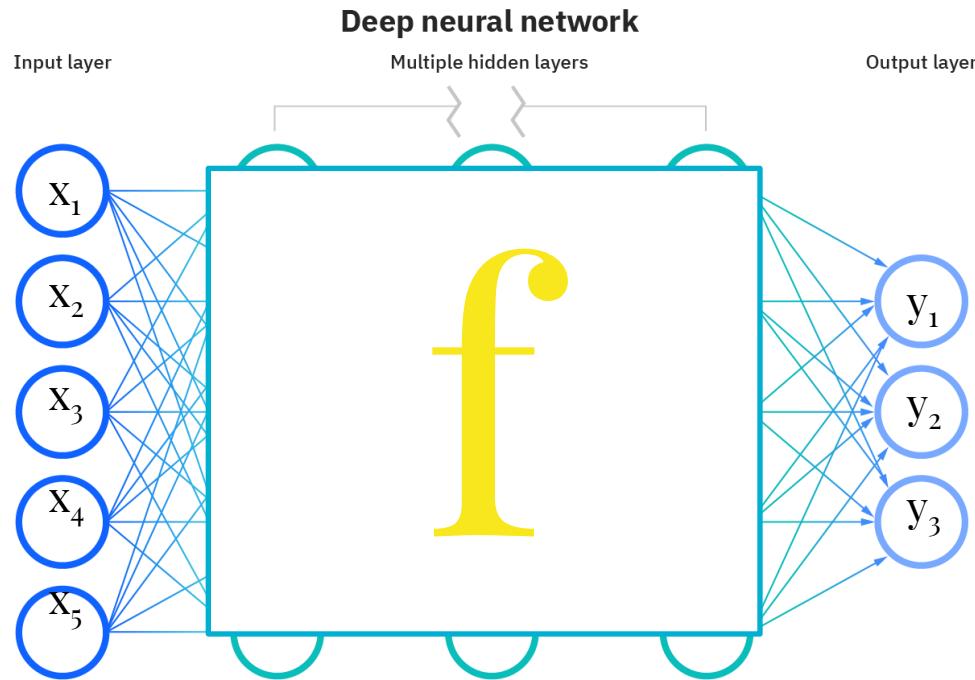
Modelele de ML știu să lucreze doar cu numere!

Majoritatea modelelor de Machine Learning primesc datele într-o formă numerică, efectuează câteva operații matematice din care rezultă alte numere, care în cele din urmă sunt interpretate de programatori.



Deep neural network





$$f(x_1, x_2, x_3, x_4, x_5) = (y_1, y_2, y_3)$$

Cum putem reprezenta o rețea neuronală?

```
# takes as input  $x \in R^n$  and gives an output  $\in R^m$ 
class MyNeuralNetwork {
    float* w: [w1, w2, ..., wp]; # weights
    float* b: [b1, b2, ..., bq]; # biases
    function fArc: Rn+p+q  $\rightarrow$  Rm # the architecture
    function loss: Rm  $\times$  Rm  $\rightarrow$  R # loss function

    float* predict(xs [x1, x2, ..., xn]) {
        return fArc(
            x1, x2, ..., xn,
            w1, w2, ..., wp,
            b1, b2, ..., bq
        );
    }
}
```

```
void train(x: [x1, x2, ..., xn], y: [y1, y2, ..., ym])
{
    float* pred = fArc(x, w, b);
    float L = loss(pred, y);
    # compute gradients,  $\delta w_i / \delta L$  and  $\delta b_j / \delta L$ 
    # apply gradient descent on w and b
}
```

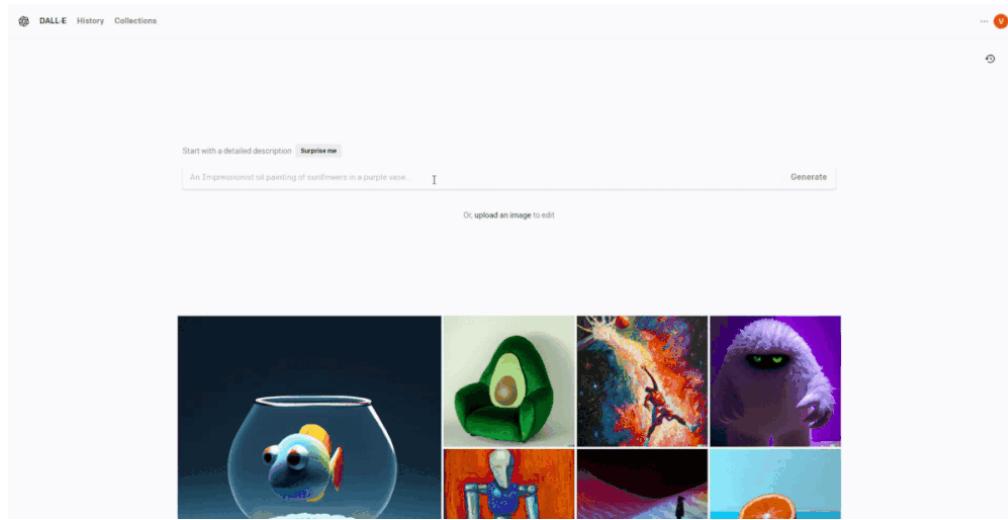
↑ <https://youtu.be/VMj-3S1tkuo>
(Andrej Karpathy – building micrograd from scratch)

**Ce facem când datele cu care
lucrăm nu mai sunt numerice?**

DALL-E

DALL-E este un model creat de cei de la OpenAI care traduce text în imagini corespunzătoare (există mai multe astfel de modele, acesta este unul dintre ele).

<https://labs.openai.com/>



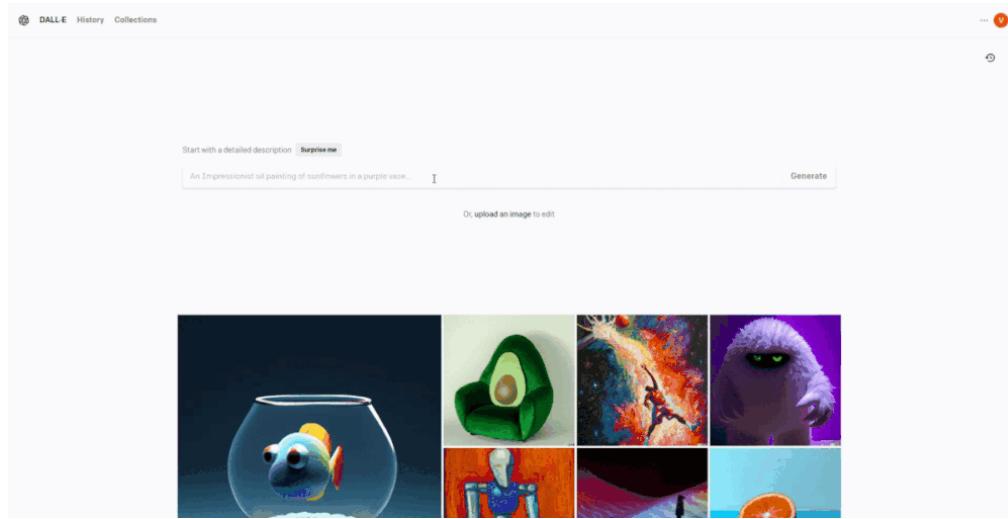
DALL-E

DALL-E este un model creat de cei de la OpenAI care traduce text în imagini corespunzătoare (există mai multe astfel de modele, acesta este unul dintre ele).

<https://labs.openai.com/>

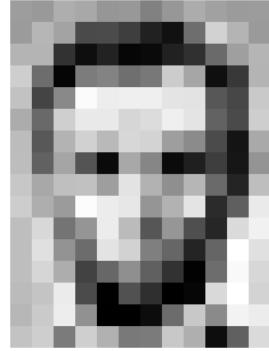
Nu există funcție matematică care să primească text, nici care să scoată imagini, ce putem face?

f("a corgi playing a flame throwing trumpet") ??



Reprezentarea numerică a imaginilor

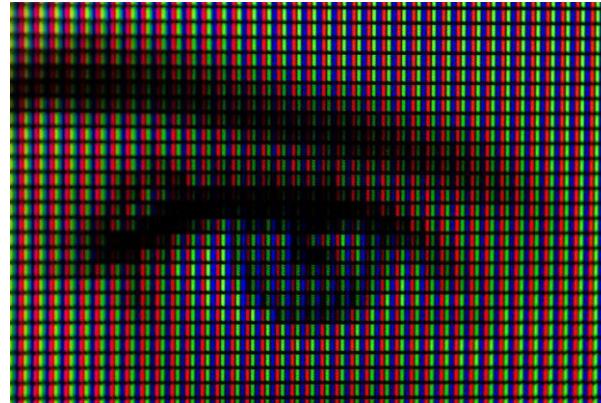
O imagine este doar o matrice formată din pixeli, deci, pentru imaginile alb-negru, putem vedea intensitatea fiecarui pixel ca output pentru funcția noastră. Imaginele colorate au pentru fiecare pixel 3 valori (RGB – Red, Green, Blue, câte o intensitate pentru fiecare), deci vom avea de 3 ori mai multe valori. De obicei aceste valori sunt numere naturale cuprinse între 0 și 255.



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	106	5	124	131	111	120	204	166	15	56	180
194	68	157	251	237	239	239	228	227	87	71	201
172	106	207	234	233	214	220	239	228	98	74	206
188	63	179	209	185	215	211	158	139	75	29	169
189	97	155	84	10	168	134	11	31	62	22	148
199	169	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	211	241
190	224	147	108	237	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	285	211
183	202	237	143	0	0	12	108	200	138	243	236
195	206	123	207	177	121	128	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	106	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201

172	105	207	233	233	213	220	220	239	239	228	98	74	206										
188	69	179	209	185	215	211	158	139	75	29	169	75	20	169									
189	97	155	84	10	168	134	11	31	62	22	148	11	31	62	22	148							
199	168	191	193	158	227	178	143	182	106	36	190	143	182	106	36	190							
205	174	155	252	236	231	149	178	228	43	95	234	178	228	43	95	234							
190	216	116	149	236	187	86	150	79	38	211	241	190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	237	210	127	102	36	101	255	224	190	224	147	108	237	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215	190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	285	211	187	196	235	75	1	81	47	0	6	217	285	211
183	202	237	143	0	0	12	108	200	138	243	236	183	202	237	143	0	0	12	108	200	138	243	236
195	206	123	207	177	121	128	200	175	13	96	218	195	206	123	207	177	121	128	200	175	13	96	218

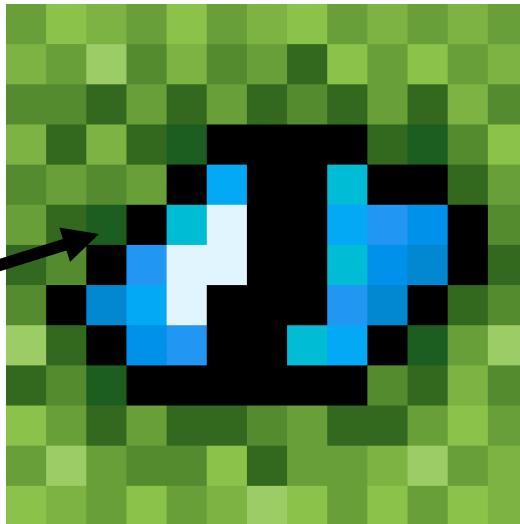


Reprezentarea numerică a imaginilor

Cod HEX: #1B5E20

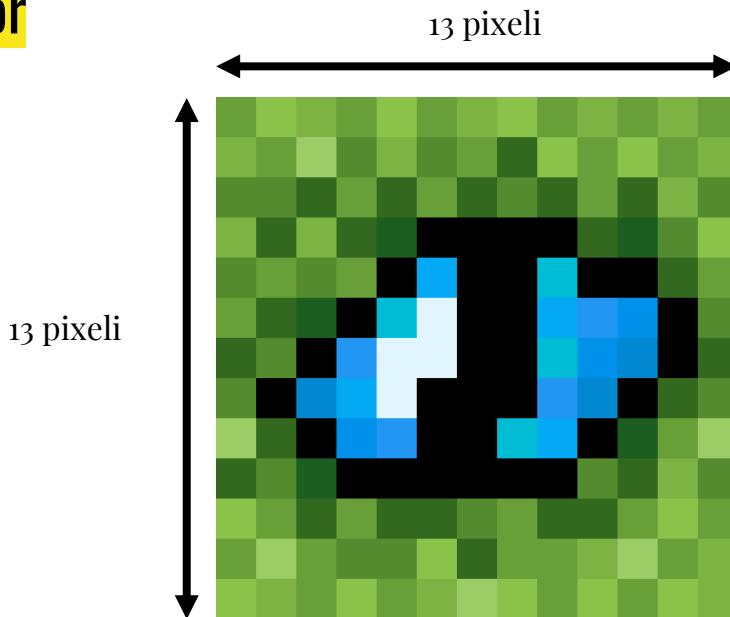
Intensitați:

- Roșu: 27
- Verde: 94
- Albastru: 32



Reprezentarea numerică a imaginilor

În total vom avea $13 \times 13 = 169$ de pixeli, iar cum fiecare pixel conține 3 valori (intensitatea pentru roșu, cea pentru verde și cea pentru albastru), vom avea în total $3 * 169 = 507$ valori.



Reprezentarea numerică a imaginilor

DALL-E generează imagini cu dimensiunea de 1024 x 1024 pixeli, deci va fi nevoie să genereze în final 2^{20} pixeli, adică $3 * 2^{20} = 3,145,728$ valori.



“Teddy bears working on new AI research underwater with 1990s technology”



“An armchair in the shape of an avocado”

Reprezentarea numerică a imaginilor

DALL-E generează imagini cu dimensiunea de 1024 x 1024 pixeli, deci va fi nevoie să genereze în final 2^{20} pixeli, adică $3 * 2^{20} = 3,145,728$ valori.

Deci, până acum, funcția matematică pentru modelul nostru de generat imagini din text ar arăta în stilul acesta:

$$f(?) = (y_1, y_2, \dots, y_{3,145,728})$$

Mai este nevoie să găsim o reprezentare a textului în numere.



“Teddy bears working on new AI research underwater with 1990s technology”

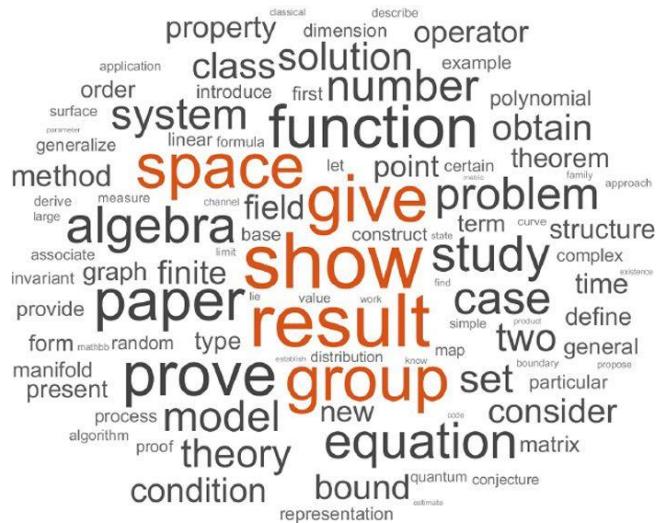


“An armchair in the shape of an avocado”

Reprezentarea numerică a textului

Acest task este puțin mai dificil, fiind nevoie să păstrăm cumva o legătură între cuvinte. De exemplu, nouă ne este ușor să găsim o asociere între cuvintele “părinte” și “tată”, însă pentru calculator este un task mai dificil.

O variantă ar fi să ținem toate cuvintele într-o listă, astfel vom avea un index pentru fiecare și aceea ar fi reprezentarea numerică, însă aşa nu ar fi nicio legătură între cuvinte, și modelul de Machine Learning ar trebui să învețe el acest lucru → va trebui antrenat mai mult, deci nu este cea mai fiabilă variantă.



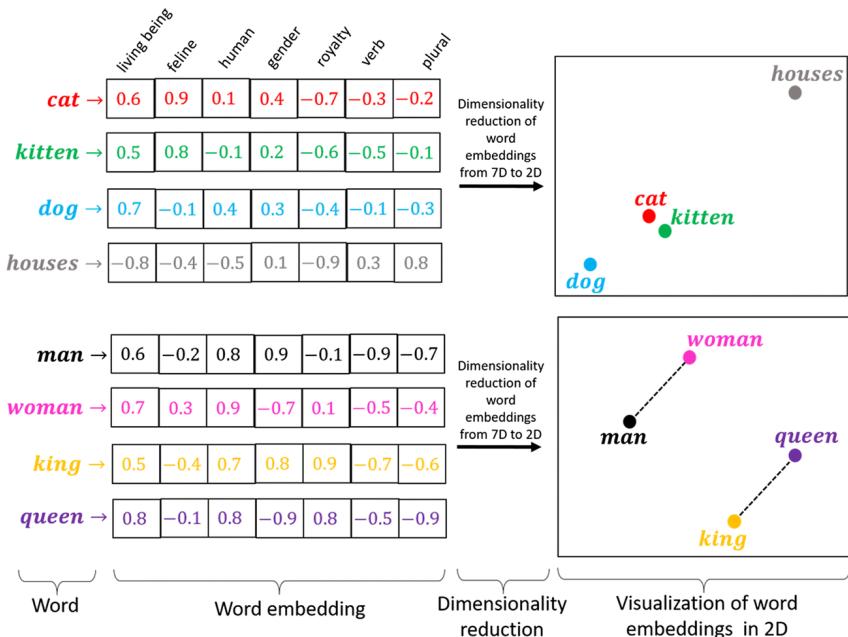
Reprezentarea numerică a textului

De aceea, au apărut câteva modele de Machine Learning care se ocupă cu partea de creat embedding-uri (perechi de numere reprezentative) pentru cuvinte.

Modelele respective au învățat să găsească relații între cuvinte, acestea putând fi interpretate și de noi.

Putem considera *Emb* un astfel de model, rezultatul aplicării acestui algoritm asupra unui text o să fie o pereche de numere:

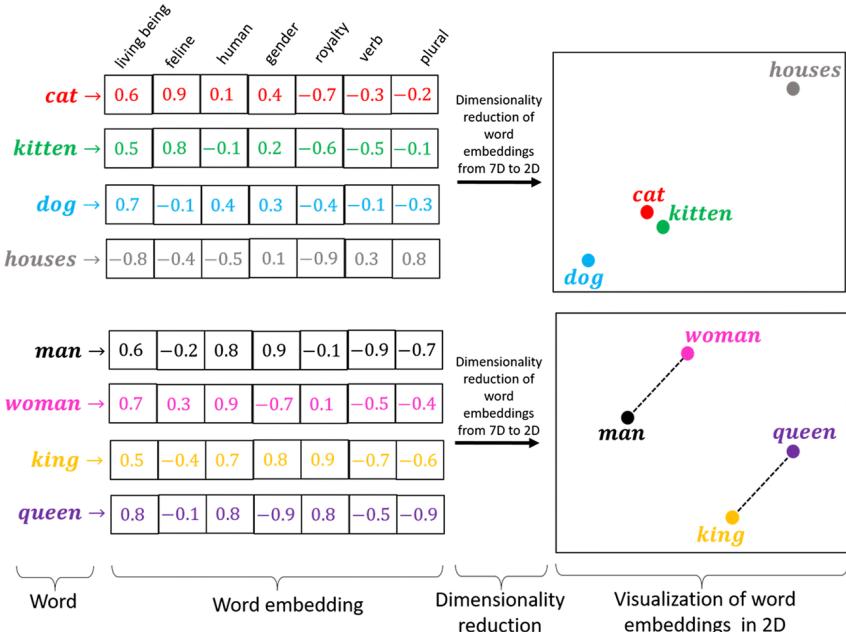
$$Emb(\text{"text"}) = (x_1, x_2, \dots, x_k)$$



Reprezentarea numerică a textului

Din figura din dreapta, partea de sus, putem observa că perechile de numere rezultatele pentru cuvintele “cat” și “kitten” sunt mai apropiate între ele decât de alte cuvinte, precum “dog” și “houses”, relație pe care și noi, ca oameni o putem observa.

=> cuvintele asemănătoare au reprezentări asemănătoare



Reprezentarea numerică a textului

În partea de jos a figurii, putem observa următoarele lucruri:

$$Emb(\text{"man"}) = (0.6, -0.2, 0.8, 0.9, -0.1, -0.9, -0.7)$$

$$Emb(\text{"woman"}) = (0.7, 0.3, 0.9, -0.7, 0.1, -0.5, -0.4)$$

$$Emb(\text{"man"}) - Emb(\text{"woman"})$$

$$= (-0.1, -0.5, -0.1, 1.6, -0.2, -0.4, -0.3)$$

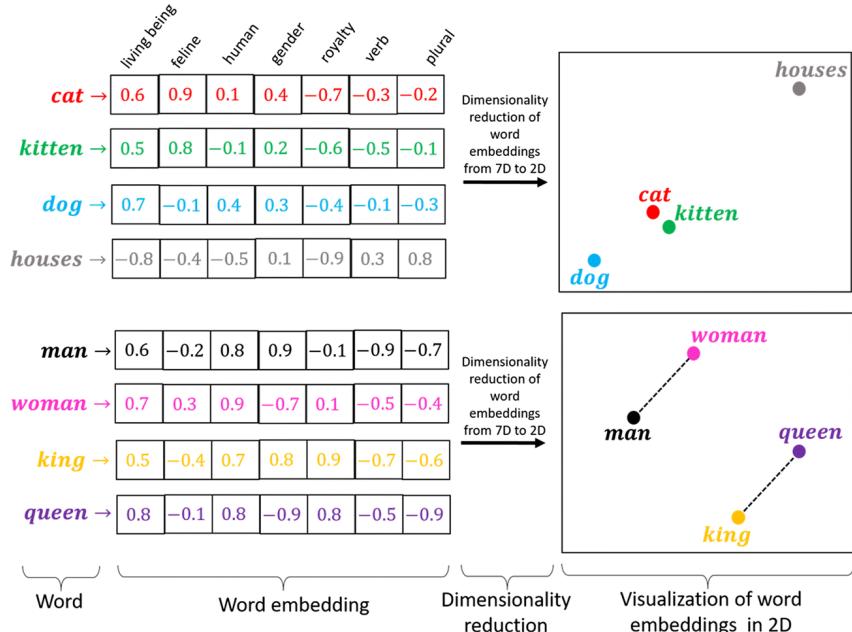
$$Emb(\text{"king"}) = (0.5, -0.4, 0.7, 0.8, 0.9, -0.7, -0.6)$$

$$Emb(\text{"queen"}) = (0.8, -0.1, 0.8, -0.9, 0.8, -0.5, -0.9)$$

$$Emb(\text{"king"}) - Emb(\text{"queen"})$$

$$= (-0.3, -0.3, -0.1, 1.7, 0.1, -0.2, 0.3)$$

Comparând cele 2 diferențe, putem observa că sunt destul de apropriate, însemnând că algoritmul a învățat că diferența dintre "man" și "woman" este asemănătoare cu cea dintre "king" și "queen".



Menti: **8415 5783**
<https://menti.com>

word2vec

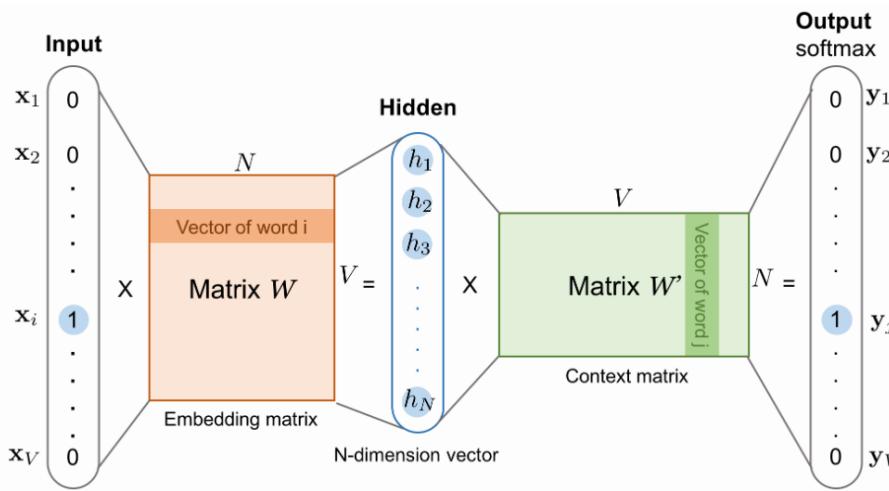


Table 1: Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Table 8: Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Este suficient?

În momentul de față, funcția noastră matematică pentru modelul de generat imagini din text arată astfel:

input: text

$$(x_1, x_2, \dots, x_k) = Emb(\text{text})$$

$$(y_1, y_2, \dots, y_n) = f(x_1, x_2, \dots, x_k)$$

image = *imageFromVector*(y₁, y₂, ..., y_n)

output: image

Să nu uităm că *Emb* și *f* sunt funcții matematice!



“flying flamingo in an aquarium”



“flying flamingo in an aquarium”

Funcțiile matematice au mereu același rezultat!

$$f(x) = x^2 + 2x + 1$$

$f(3) = 16$ în orice moment, nu se va schimba niciodată, dar totuși DALL-E produce output-uri diferite pentru același text. De ce?

“Insanity is doing the same thing over and over and expecting different results.” – Albert Einstein



“flying flamingo in an aquarium”



“flying flamingo in an aquarium”

Spațiul latent

Pe lângă text-ul dat ca input, de multe ori, algoritmii generativi primesc și un set de numere aproximativ aleator (z_1, z_2, \dots, z_p), astfel că, la fiecare generare, fiecare imagine va arăta diferit.



“flying flamingo in an aquarium”



“flying flamingo in an aquarium”

Spațiul latent

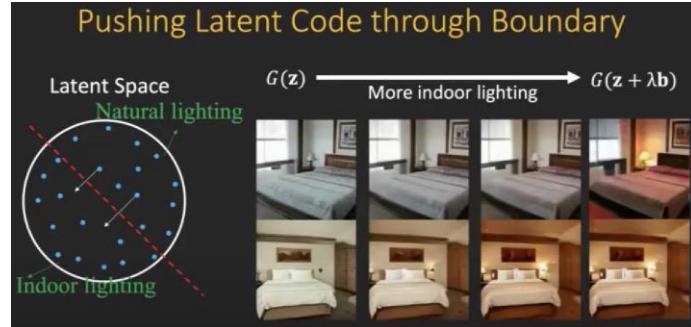
Pe lângă text-ul dat ca input, de multe ori, algoritmii generativi primesc și un set de numere aproximativ aleator (z_1, z_2, \dots, z_p), astfel că, la fiecare generare, fiecare imagine va arăta diferit.

Un lucru interesant observat, este că aceste valori au anumite caracteristici asupra imaginii finale, deci, modificând câte puțin unul dintre numerele date aleator pentru o imagine (ex. z_i), o caracteristică din imaginea rezultat se va modifica tot câte puțin (asemănător cu rețelele de embedding de la traducerea textului).

$$\text{Intuiție: } f(x) = x^2 + 2x + 1$$

$$f(3) = 16; f(3.01) = 16.08; f(3.02) = 16.16$$

➔ modificări mici input conduc la modificări mici ale rezultatului pentru funcțiile continue



Sursa: <https://youtu.be/8Hm4ad5QIUE>

Reprezentarea matematică finală

Deci în final, modelul la care am reușit să ajungem este o funcție matematică:

input: text

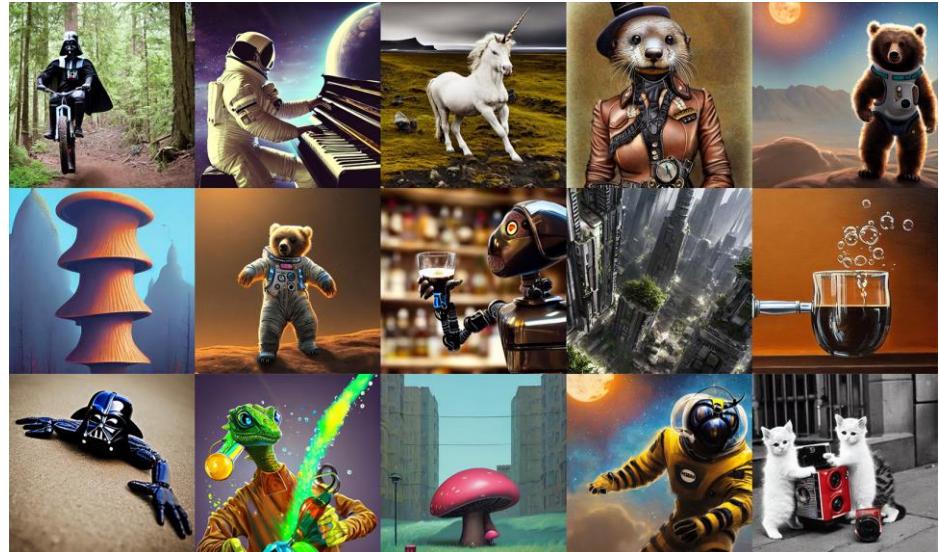
$$(z_1, z_2, \dots, z_p) = \text{generateRandomNumbers}()$$

$$(x_1, x_2, \dots, x_n) = \text{Emb}(\text{text})$$

$$(y_1, y_2, \dots, y_k) = f(x_1, x_2, \dots, x_n, z_1, z_2, \dots, z_p)$$

$$\text{image} = \text{imageFromVector}(y_1, y_2, \dots, y_k)$$

output: image



Reprezentarea matematică finală

Deci în final, modelul la care am reușit să ajungem este o funcție matematică:

input: text

$(z_1, z_2, \dots, z_p) = generateRandomNumbers()$

$(x_1, x_2, \dots, x_n) = Emb(\text{text})$

$(y_1, y_2, \dots, y_k) = f(x_1, x_2, \dots, x_n, z_1, z_2, \dots, z_p)$

$\text{image} = imageFromVector(y_1, y_2, \dots, y_k)$

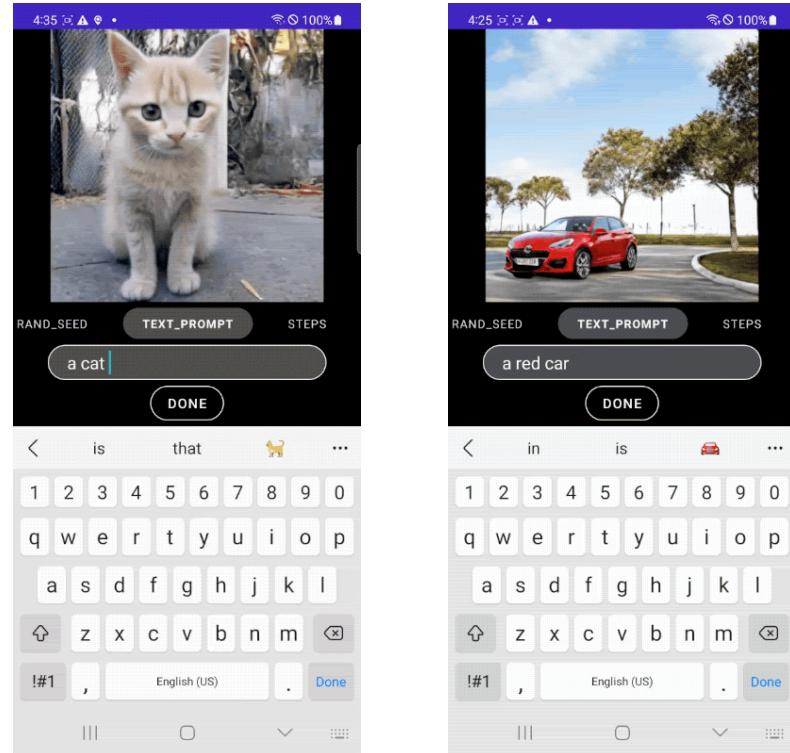
output: image

Partea ușoară: $generateRandomNumbers()$,

$Emb()$ (există deja multe soluții),

$imageFromVector()$

Partea dificilă: găsirea funcției f



<https://research.google/blog/mobilediffusion-rapid-text-to-image-generation-on-device/>

That's it.
Mersi de participare!

Concepțe cheie:

- Algoritmii de ML lucrează cu numere → Trebuie să transformăm datele de prelucrat în numere.
- Majoritatea algoritmilor de ML sunt funcții matematice, învățarea constă în găsirea acestora.
- Cele mai semnificative reprezentări numerice ale unor date păstrează relații între date pe care putem într-o oarecare măsură să le interpretăm și noi.



Lanțuri **MARKOV**

Noțiuni teoretice

Un lanț Markov este un model matematic care ne ajută să înțelegem și să prezicem comportamentul unui sistem care evoluează în timp.



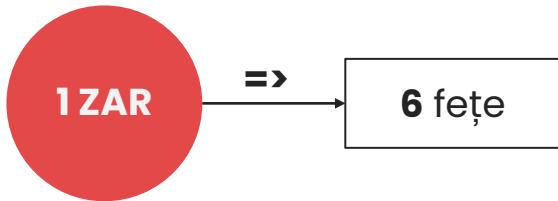
Sistemul este descris printr-o serie de stări posibile și de tranziții între aceste stări.

Fiecare stare este o situație sau o condiție a sistemului într-un anumit moment în timp.



Proprietatea lui Markov afirmă că starea viitoare a unui sistem stocastic este determinată exclusiv de starea sa actuală și nu depinde de întreaga istorie a sistemului.

Exemplu: Aruncarea cu zarul

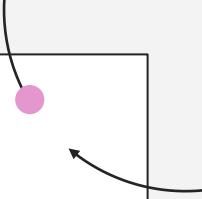


1: 1/6 (16,6 %) **4: 1/6 (16,6 %)**
2: 1/6 (16,6 %) **5: 1/6 (16,6 %)**
3: 1/6 (16,6 %) **6: 1/6 (16,6 %)**



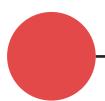


Moduri de reprezentare



Ciclurile Markov pot fi reprezentate în diverse moduri în funcție de tipul de proces Markov și de nivelul de detaliu necesar în modelare.

Cele mai comune moduri de a reprezenta ciclurile Markov:



Graf orientat/
diagramă de stări

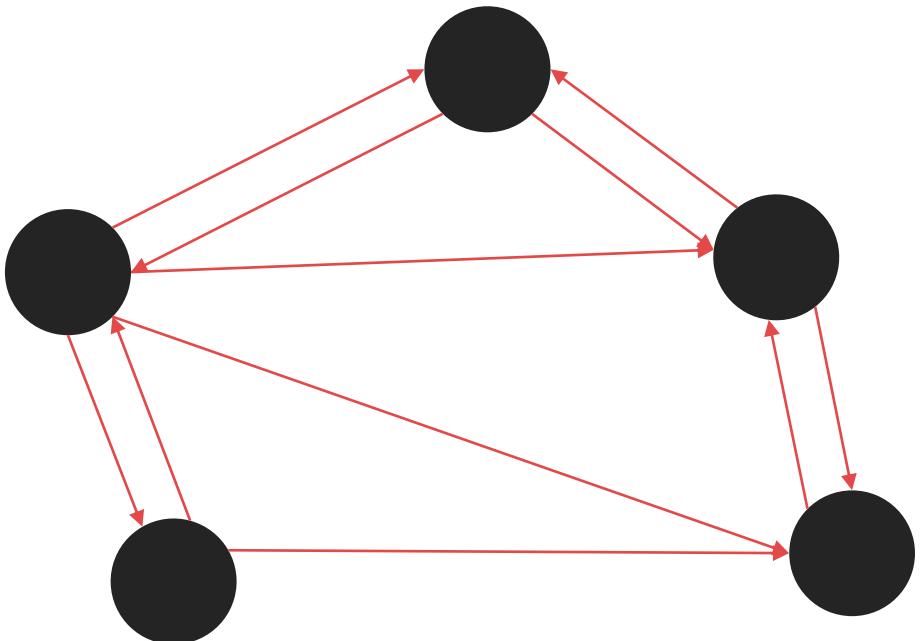


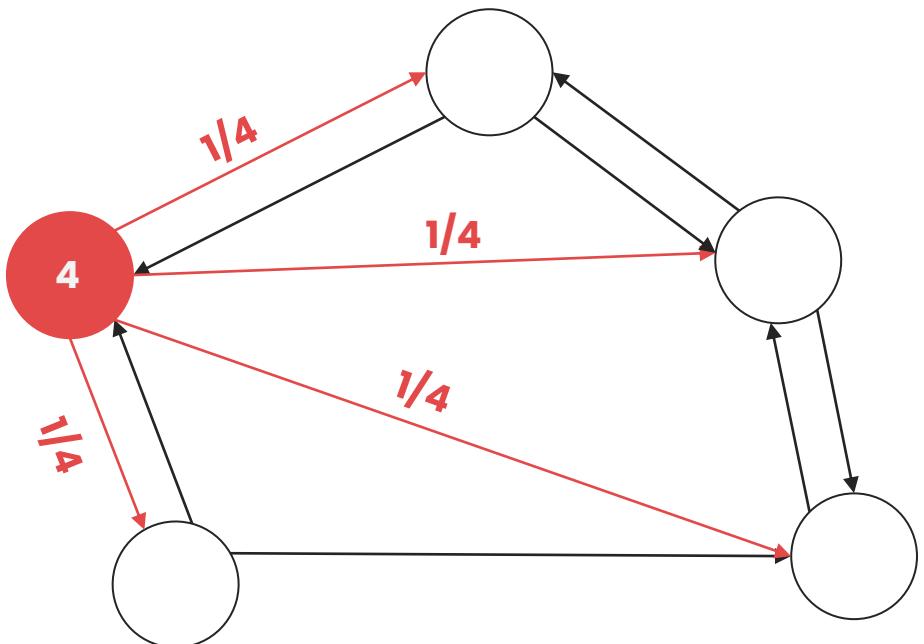
Matricea de tranziție

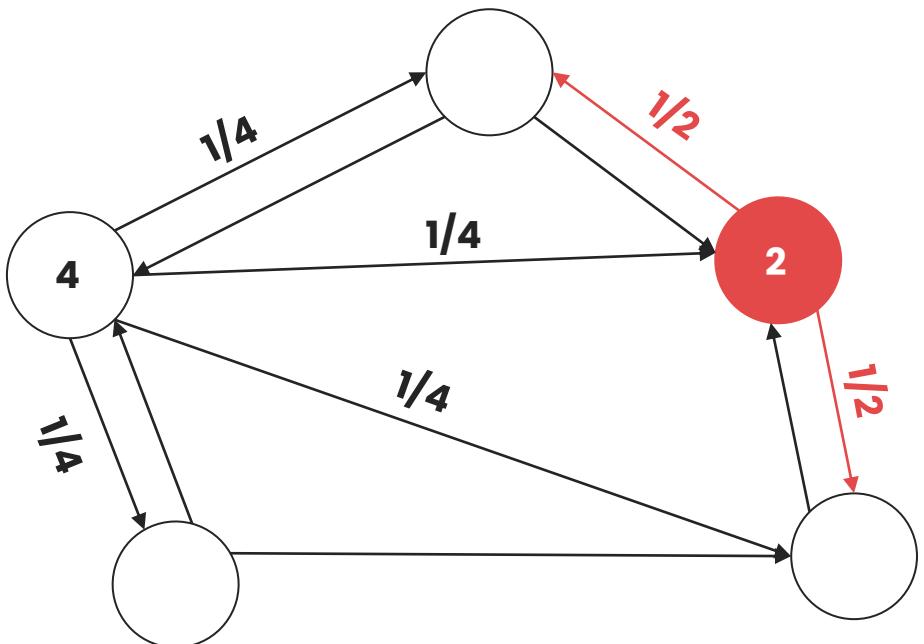


Arbore*

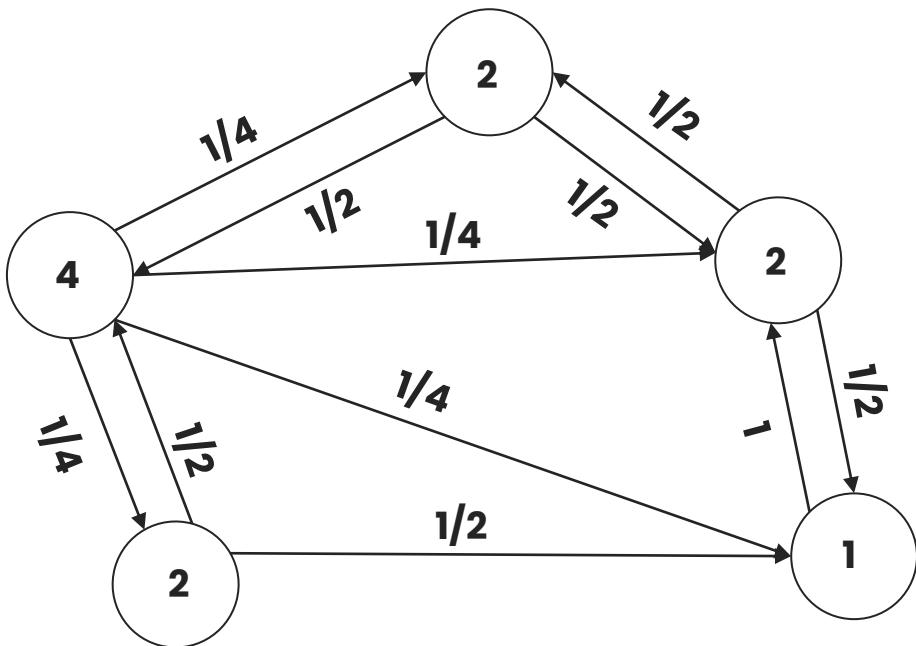






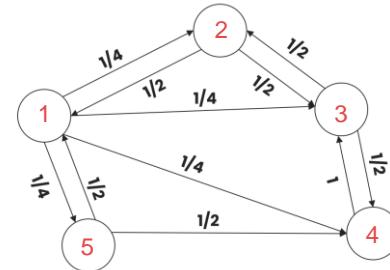


Graf orientat

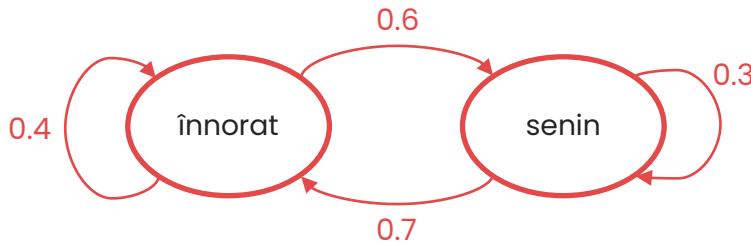
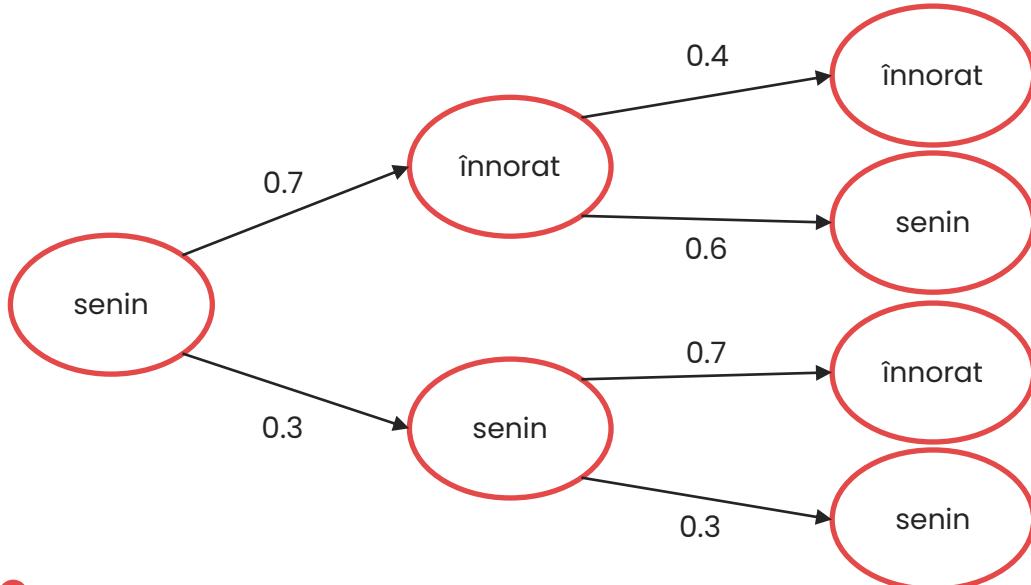


Matricea de tranziție

	Nod 1	Nod 2	Nod 3	Nod 4	Nod 5
Nod 1	0.0	0.25	0.25	0.25	0.25
Nod 2	0.5	0.0	0.5	0.0	0.0
Nod 3	0.0	0.5	0.0	0.5	0.0
Nod 4	0.0	0.0	1.0	0.0	0.0
Nod 5	0.5	0.0	0.0	0.5	0.0



Arbore*



Şansele ca după 2 zile să fie:

- senin = $0.3 \cdot 0.3 + 0.7 \cdot 0.6 = 0,51$
- înnorat = $0.3 \cdot 0.7 + 0.7 \cdot 0.4 = 0,49$

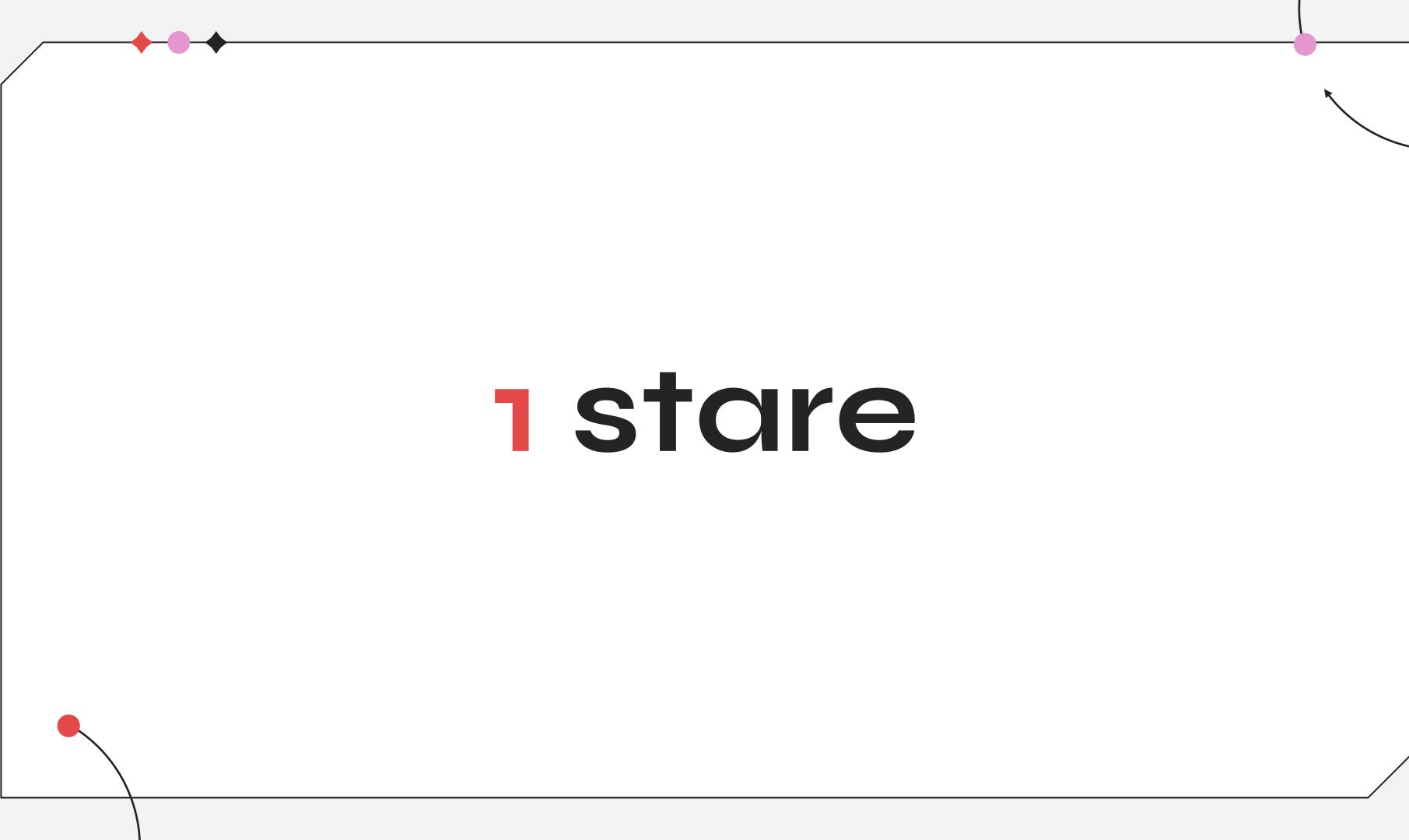


Exercițiu: Prezicerea vremii

Având la dispoziție matricea de tranziție între stările vremii, scrieți un program ce prezice cu ajutorul proprietății lui Markov vremea din ziua următoare.

	Sunny	Partly Cloudy	Cloudy	Rainy	Storm	Snowy	Foggy
Sunny	0,65	0,1	0,05	0,05	0,03	0,02	0,1
Partly Cloudy	0,2	0,55	0,1	0,05	0,02	0,03	0,05
Cloudy	0,1	0,15	0,45	0,1	0,05	0,05	0,1
Rainy	0,05	0,05	0,1	0,6	0,1	0,05	0,05
Storm	0,03	0,02	0,05	0,1	0,6	0,1	0,1
Snowy	0,02	0,03	0,05	0,05	0,15	0,55	0,15
Foggy	0,1	0,05	0,1	0,05	0,1	0,15	0,55





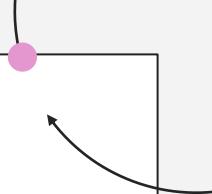
1 stare

1 stare -> k stari



1 stare

k stări



*Noi ştim că unu ori unu fac unu,
dar un inorog ori o pară
nu ştim cât face.*

*Ştim că cinci fără patru fac unu,
dar un nor fără o corabie
nu ştim cât face.*

*Ştim, noi ştim că opt
împărţit la opt fac unu,
dar un munte împărţit la o capră
nu ştim cât face.*

*Ştim că unu plus unu fac doi,
dar eu şi cu tine,
nu ştim, vai, nu ştim cât facem.*

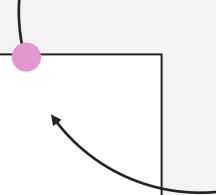


1 stare

Noi ştim că unu ori unu fac unu,
dar un inorog ori o pară
nu ştim cât face.
Ştim că cinci fără patru fac unu,
dar un nor fără o corabie
nu ştim cât face.
Ştim, noi ştim că opt
împărţit la opt fac unu,
dar un munte împărţit la o capră
nu ştim cât face.
Ştim că unu plus unu fac doi,
dar eu şi cu tine,
nu ştim, vai, nu ştim cât facem.

Input: **unu**

k stări



Noi ştim că unu ori unu fac unu,
dar un inorog ori o pară
nu ştim cât face.
Ştim că cinci fără patru fac unu,
dar un nor fără o corabie
nu ştim cât face.
Ştim, noi ştim că opt
împărţit la opt fac unu,
dar un munte împărţit la o capră
nu ştim cât face.
Ştim că unu plus unu fac doi,
dar eu şi cu tine,
nu ştim, vai, nu ştim cât facem.

Input: **dar un**

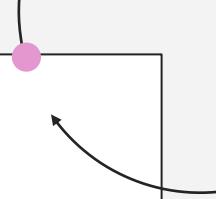


1 stare

Noi ştim că **unu** ori **unu** fac **unu**,
dar un inorog ori o pară
nu ştim cât face.
Ştim că cinci fără patru fac **unu**,
dar un nor fără o corabie
nu ştim cât face.
Ştim, noi ştim că opt
împărţit la opt fac **unu**,
dar un munte împărţit la o capră
nu ştim cât face.
Ştim că **unu** plus **unu** fac doi,
dar eu şi cu tine,
nu ştim, vai, nu ştim cât facem.

Input: **unu**

k stări



Noi ştim că **unu** ori **unu** fac **unu**,
dar un inorog ori o pară
nu ştim cât face.
Ştim că cinci fără patru fac **unu**,
dar un nor fără o corabie
nu ştim cât face.
Ştim, noi ştim că opt
împărţit la opt fac **unu**,
dar un munte împărţit la o capră
nu ştim cât face.
Ştim că **unu** plus **unu** fac doi,
dar eu şi cu tine,
nu ştim, vai, nu ştim cât facem.

Input: **dar un**

1 stare

Noi ştim că **unu** **ori** **unu** **fac** **unu**,
dar un **inorog** **ori** o pară
nu ştim cât face.

Ştim că cinci fără patru fac **unu**,
dar un **nor** fără o corabie
nu ştim cât face.

Ştim, noi ştim că opt
împărţit la opt fac **unu**,
dar un **munte** împărţit la o capră
nu ştim cât face.

Ştim că **unu** **plus** **unu** **fac** doi,
dar eu şi cu tine,
nu ştim, vai, nu ştim cât facem.

Input: **unu**

Output:

- **ori:** **14.28%**
- **fac:** **28.57%**
- **dar:** **42.85%**
- **plus:** **14.28%**

k stări

Noi ştim că **unu** **ori** **unu** **fac** **unu**,
dar un **inorog** **ori** o pară
nu ştim cât face.

Ştim că cinci fără patru fac **unu**,
dar un **nor** fără o corabie
nu ştim cât face.

Ştim, noi ştim că opt
împărţit la opt fac **unu**,
dar un **munte** împărţit la o capră
nu ştim cât face.

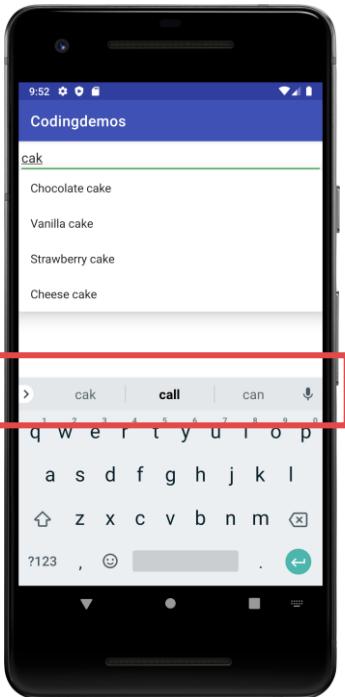
Ştim că **unu** **plus** **unu** **fac** doi,
dar eu şi cu tine,
nu ştim, vai, nu ştim cât facem.

Input: **dar** **un**

Output:

- **inorog:** **33.3%**
- **nor:** **33.3%**
- **munte:** **33.3%**

EXEMPLU: auto-complete





Întrebări?



INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure

– generative AI –

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

c. Sisteme inteligente

■ Sisteme care învață singure

- Arbori de decizie
- **Rețele neuronale artificiale**
- kNN
- Algoritmi evolutivi
- Mașini cu suport vectorial

■ Sisteme bazate pe reguli

■ Sisteme hibride

B. Rezolvarea problemelor prin căutare

■ Definirea problemelor de căutare

■ Strategii de căutare

- Strategii de căutare neinformate
- Strategii de căutare informate
- Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
- Strategii de căutare adversială

Materiale de citit și legături utile

- <https://www.tensorflow.org/text/tutorials/word2vec>
- <https://radimrehurek.com/gensim/models/word2vec.html>
- <https://www.ruder.io/word-embeddings-1/>
- <https://aylien.com/blog/a-review-of-the-recent-history-of-natural-language-processing?ref=ruder.io>
- <https://nlp.stanford.edu/~manning/> - a lot of lectures about NLP

De ce reprezentari vectoriale ale intelelesului unui cuvant/text?

- Permit determinarea similaritatii intre cuvinte/texte
 - **fast** is similar to **rapid**
 - **tall** is similar to **height**
- Question answering:
- Q: "How **tall** is Mt. Everest?"
Candidate A: "The official **height** of Mount Everest is 29029 feet"

Intuitia din spatele similaritatii

□ Exemplu: Ce este o Sirrus X 3.0?

*The Sirrus X 3.0 invites you to explore beyond boundaries. With confidence-inspiring tires, an upright riding position, and intuitive components, **Sirrus X 3.0** is your ticket to adventure.*

- Din context, o persoana poate ghici ca Sirrus X 3.0 este un model de bicicleta

- Pentru un algoritm, intuitia e ca **doua cuvinte sunt similar daca ele sunt folosite in context similar**

Diferite reprezentari vectoriale pentru text

□ Reprezentari rare (*sparse*):

1. Mutual-information weighted word co-occurrence matrices

□ Reprezentari dense (compacte) :

2. Singular Value Decomposition (si Latent Semantic Analysis)
3. Neural-network-inspired models (skip-grams, CBOW)
4. Altele (e.g. brown clusters)

Vectori rari vs. densi

- Vectorii → matricea de co-ocurenta a termenilor
 - **lungi** (length $|V| = 20,000 \rightarrow 50,000$)
 - **rari** (f multe elemente sunt 0)
- Alternativa: vectori invatati (prin AI/ML)
 - scurti (length 200-1000)
 - **densi** (multe elemente nu sunt 0)
- De ce vectori densi?
 - Vectorii scurti -> folositi mai usor ca si *features* in algoritmii de invatare (mai putini coeficienti de invatat)
 - Vectorii densi pot generaliza mai bine, captand sinonimia termenilor
 - Bike – scooter
 - Car – automobile
 - House – apartment

Modele de predictie (invatare) a reprezentarilor

❑ Evolutie

■ Word-level

- 2003 - N-gram Neural language model (Montreal - Bengio)
 - <https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf>
 - Probability of the target word based on previous k words
 - Estimation of the probability by an ANN -> prediction of the next word in a sequence
- 2008 - multi-task model (Princeton – Collobert)
 - https://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf
 - Probability of MORE target words (a sequence of words)
 - Estimation of the probability by an ANN –similar to Bengio's model,
- 2013 – word2vec (Google – Mikolov)
 - <https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf>
 - Continous BOW model -> try to predict a word based on its context (neighbours); input = neighbour words, output = target word
 - Skip-gram model -> try to predict context (neighbours) of a word; input = target word; output = neighbour words;
 - Visualisation of embeddings <https://ronxin.github.io/wevi/>
 - Trained vectors <https://radimrehurek.com/gensim/models/word2vec.html> or <https://github.com/3Top/word2vec-api#where-to-get-a-pretrained-models>

■ Sentence (document) level (2014 - ...)

- 2014 - Paragraph embedding <https://arxiv.org/abs/1405.4053>

■ Contextual word-vectors (Word vectors compress all contexts into a *single vector*) (2016 - ...)

- 2016 context2vec <https://www.aclweb.org/anthology/K16-1006.pdf>
- 2017 tagLM <https://arxiv.org/abs/1705.00108>
- 2017 CoVe <https://papers.nips.cc/paper/2017/hash/20c86a628232a67e7bd46f76fba7ce12-Abstract.html>
- 2018 ELMo <https://www.aclweb.org/anthology/N18-1202/>
- 2018 ULMFiT <https://arxiv.org/abs/1801.06146>
- 2019 BERT <https://arxiv.org/abs/1810.04805>

Modele de predictive (invatare) a reprezentarilor

□ Modelul Word2vec

- **Skip-gram** (Mikolov et al. 2013a), **CBOW** (Mikolov et al. 2013b)
- Se invata reprezentari, numite embeddings, ca parte din procesul de predictive/generare a textului
- Se antreneaza o retea neuronală pentru prezicerea urmatorului cuvant
- Avantaje
 - Rapid, simplu de antrenat
 - Modele gata antrente disponibile online

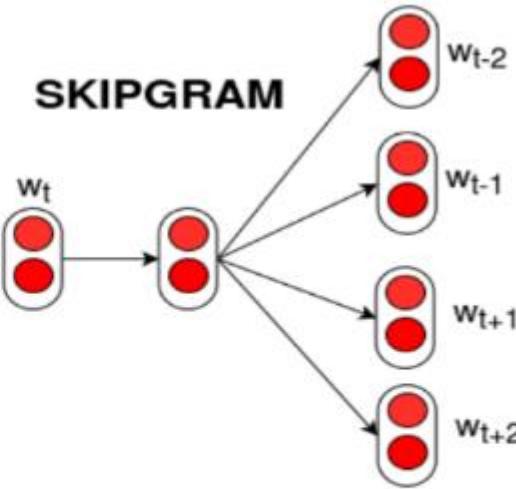
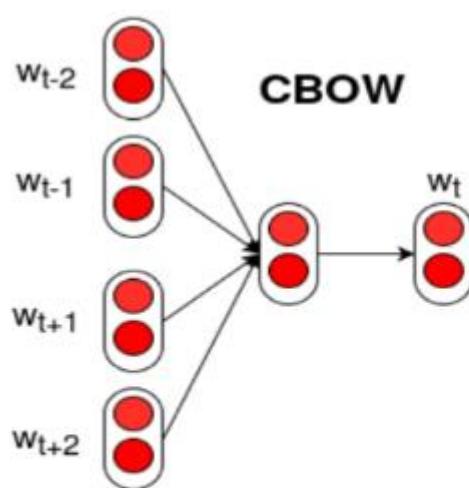
Invatare supervizata fara etichetare manuala de date

- Text: "*She rides a bike. He rides a scooter in park. My father drives a motorcycle. ...*"
- Vocabular $V = \{she, ride, bike, he, scooter, park, my, father, drive, motorcycle\}$
- Perechi (context, cuvant)

Negative sampling

[she ride] bike	1		[she ride] he	0
[ride bike] he	1		[she ride] scooter	0
[he ride] scooter	1		[she ride] park	0
[ride scooter] park	1		[she ride] my	0
[my father] drive	1		[she ride] father	0
[father drive] motorcycle	1		[she ride] drive	0
			[ride bike] scooter	0
			[ride bike] park	0
			...	
			[father drive] she	0

Arhitecturi

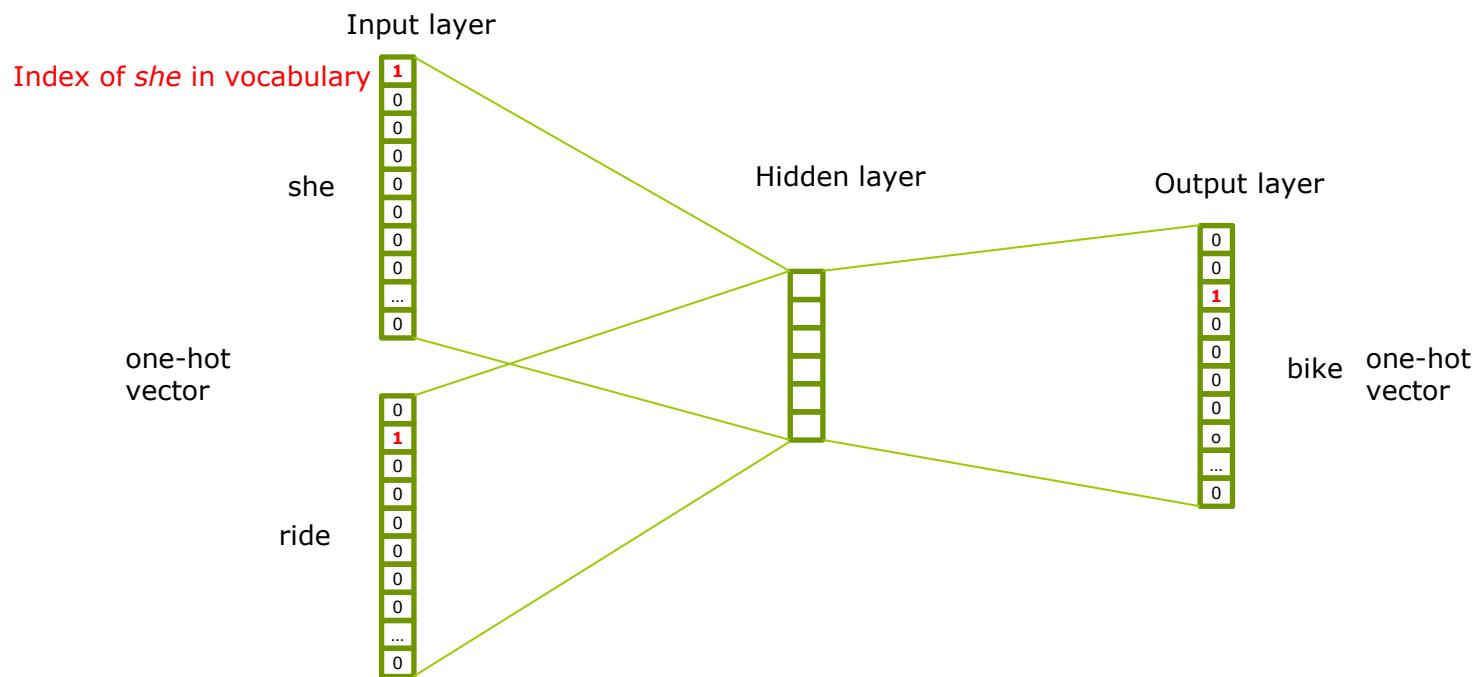


$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_t | w_{t+j})$$

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

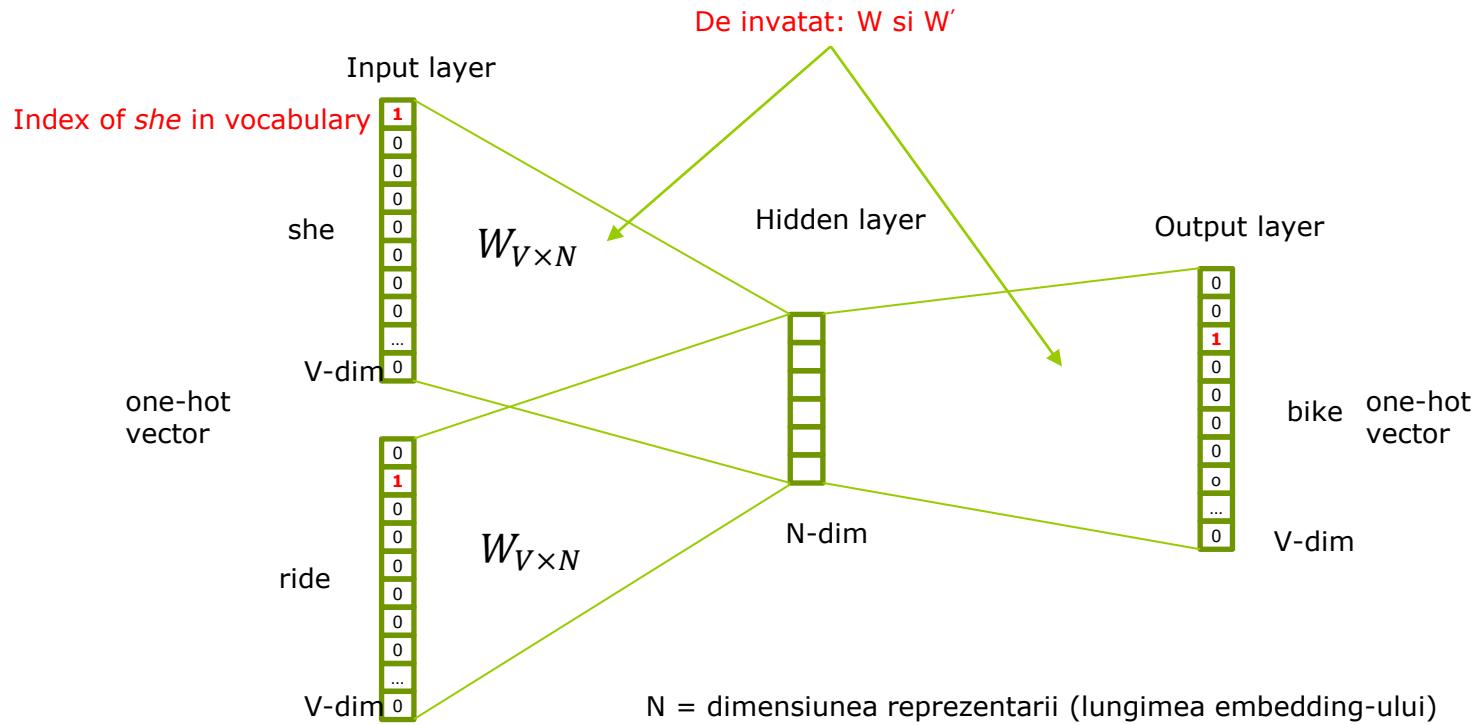
CBOW (Continuous Bag of Words)

	she	ride	bike	he	scooter	park	my	father	drive	motorcycle
She	1	0	0	0	0	0	0	0	0	0
Ride	0	1	0	0	0	0	0	0	0	0
Bike	0	0	1	0	0	0	0	0	0	0

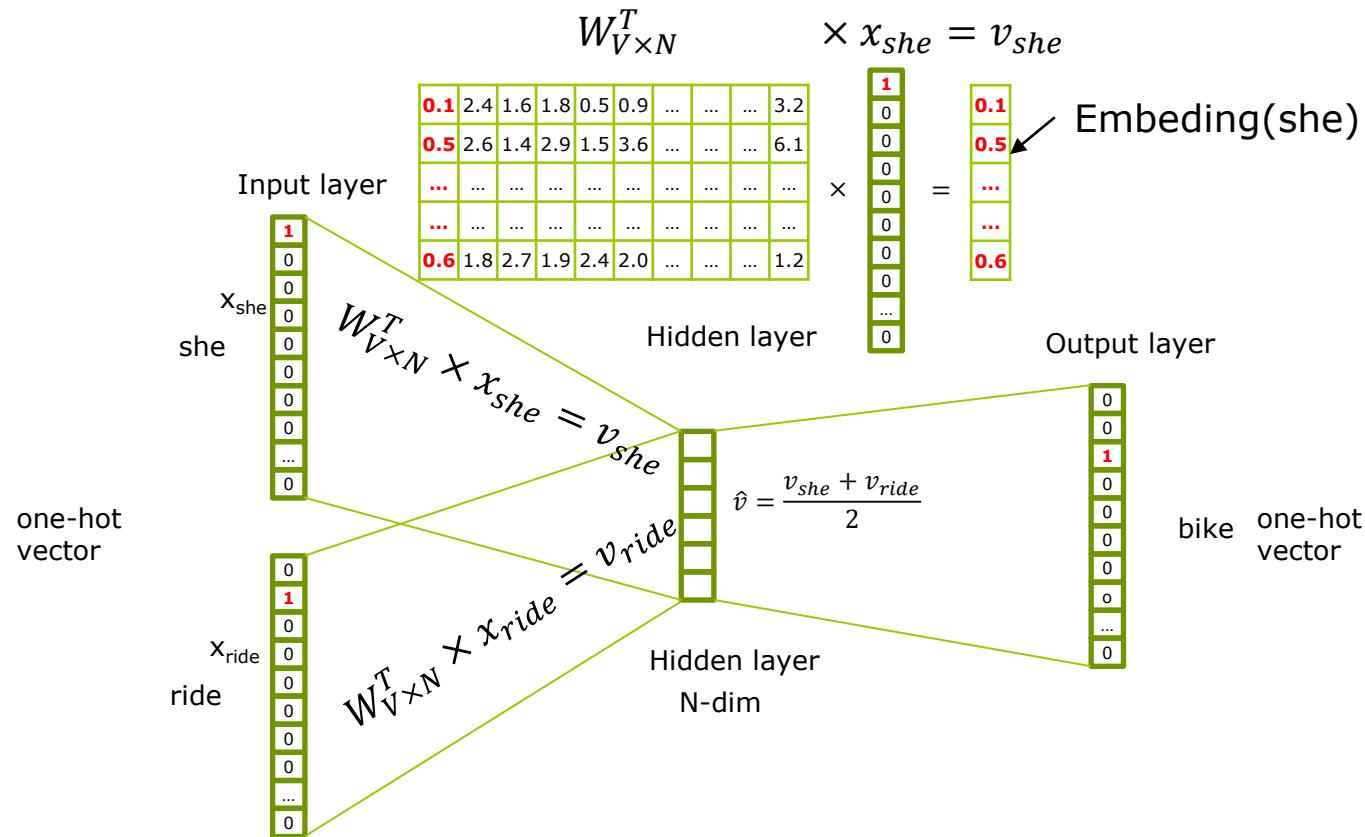


CBOW (Continuous Bag of Words)

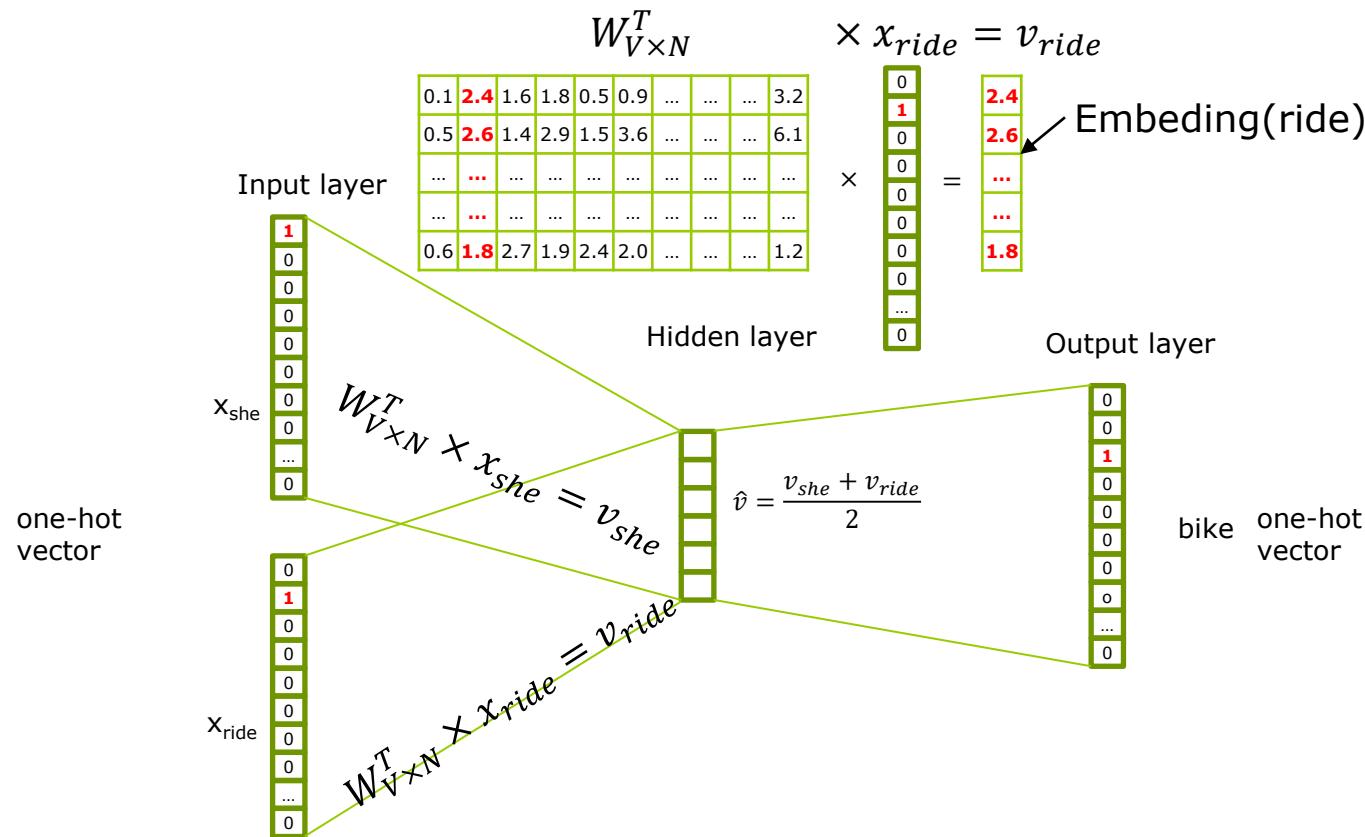
	she	ride	bike	he	scooter	park	my	father	drive	motorcycle
She	1	0	0	0	0	0	0	0	0	0
Ride	0	1	0	0	0	0	0	0	0	0
Bike	0	0	1	0	0	0	0	0	0	0



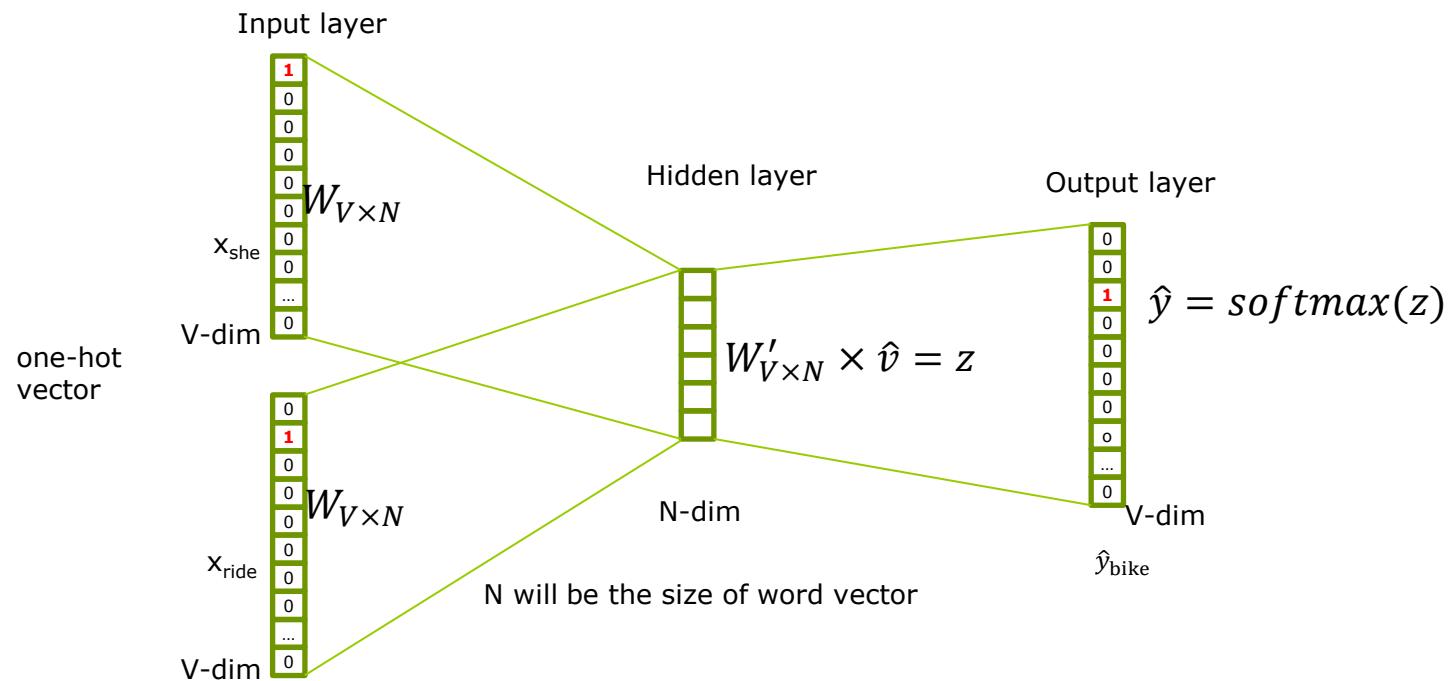
CBOW (Continuous Bag of Words)



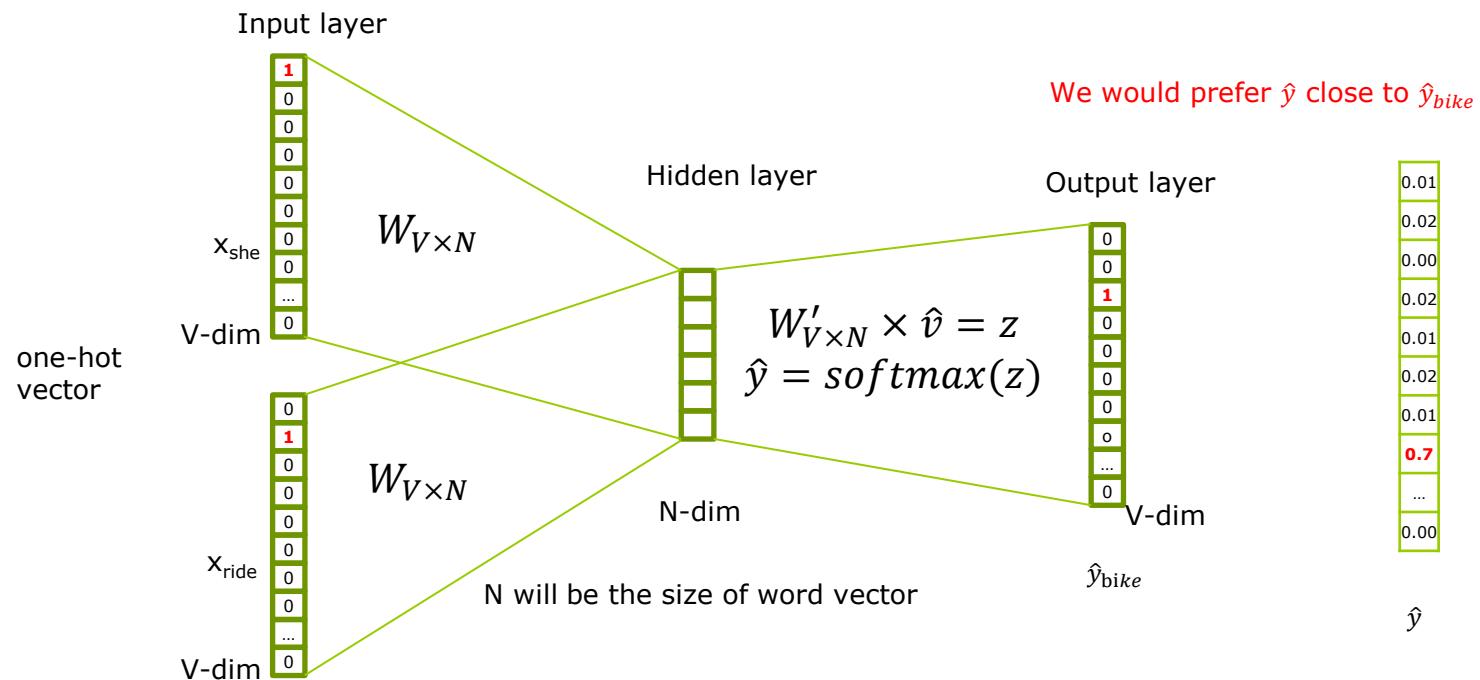
CBOW (Continuous Bag of Words)



CBOW (Continuous Bag of Words)



CBOW (Continuous Bag of Words)



Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

■ **Sisteme care învață singure**

- Arbori de decizie
- Rețele neuronale artificiale - transformers
- Mașini cu suport vectorial
- Algoritmi evolutivi

- Sisteme bazate pe reguli
- Sisteme hibride

C. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
 - Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversarială

-
- ❑ Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop
 - Prof. Dr. Radu Ionescu - <https://raduionescu.herokuapp.com/>

INTELIGENȚĂ ARTIFICIALĂ



Sisteme inteligente

Sisteme care învață singure

– generative AI –

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

c. Sisteme inteligente

■ Sisteme care învață singure

- Arbori de decizie
- **Rețele neuronale artificiale**
- kNN
- Algoritmi evolutivi
- Mașini cu suport vectorial

■ Sisteme bazate pe reguli

■ Sisteme hibride

B. Rezolvarea problemelor prin căutare

■ Definirea problemelor de căutare

■ Strategii de căutare

- Strategii de căutare neinformate
- Strategii de căutare informate
- Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
- Strategii de căutare adversială

Rețele neuronale artificiale

- ❑ Rețele neuronale dense (fully-connected)
- ❑ Rețele neuronale convolutive
 - Word embeddings
 - Transformers

Word embeddings

- Static (context-free)

- Dynamic (context-based)

Word embeddings

❑ Static (context-free)

- Word2vec (2013)
- GLoVe (2014)

The customer _____ was great

1

Projection (i.e. matrix multiplication)

Mean

Prediction

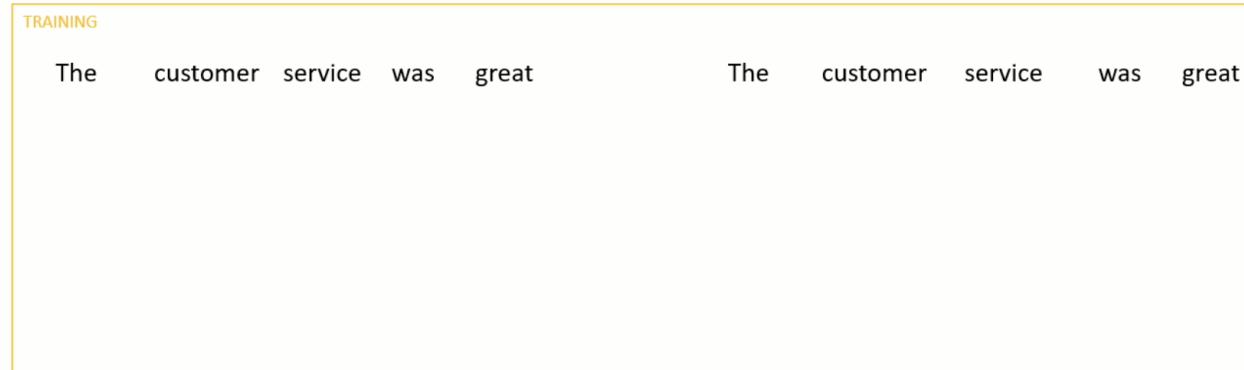
Word embeddings

❑ Static (context-free)

- Word2vec (2013)
- GLoVe (2014)

❑ Dynamic (context-based)

- ELMo (2018)



Forward LSTM
Backward LSTM
Prediction
Concatenation
Weighted average

Word embeddings

❑ Static (context-free)

- Word2vec (2013)
- GLoVe (2014)

❑ Dynamic (context-based)

- ELMo (2018)
- BERT (2019)

[CLS] The ____ service was great [SEP] They ____ very help ##ful [SEP]

Attention
Prediction

Transformers

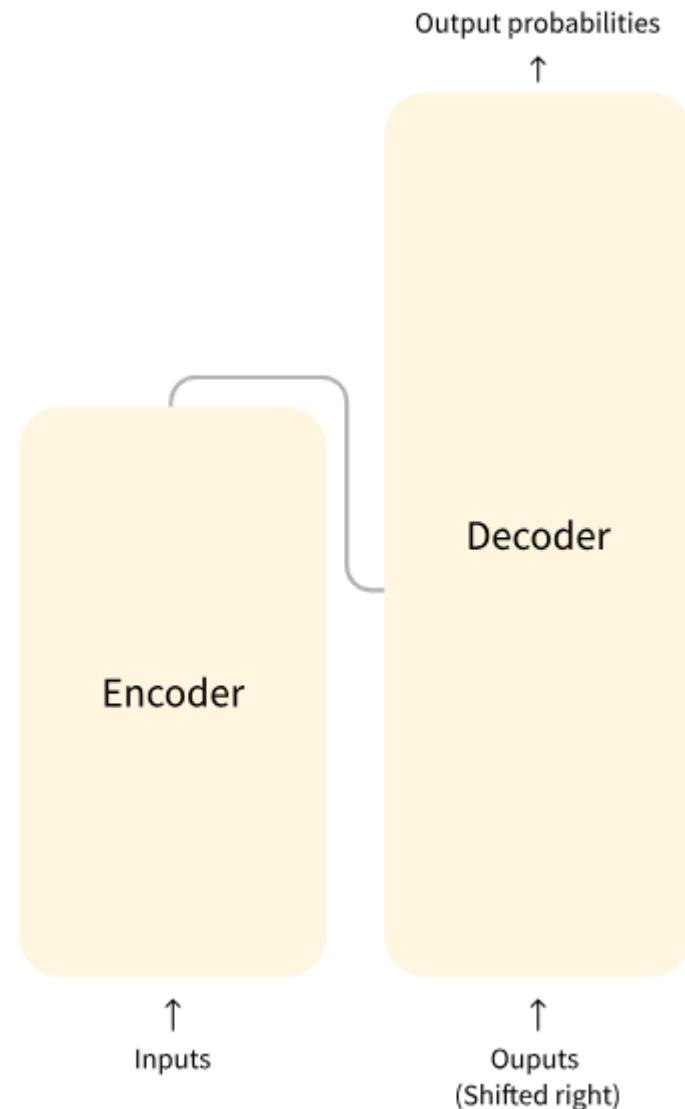
❑ Architecture

■ Encoder

- ❑ Primeste cuvinte
- ❑ Construieste reprezentar

■ Decoder

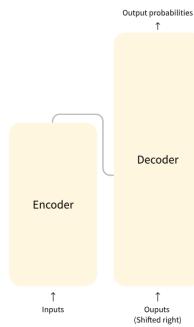
- ❑ Primeste
 - reprezentari (features)
 - Alte inputuri
- ❑ Genereaza sevenete de c



Transformers

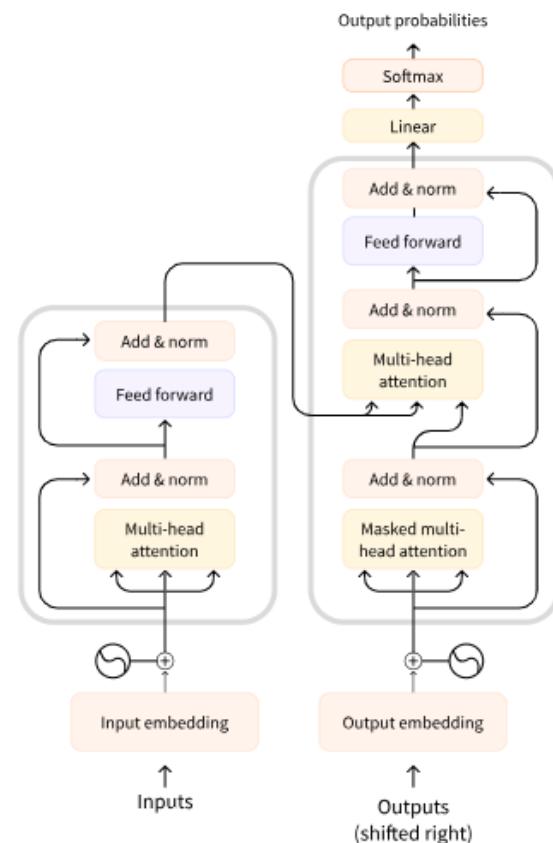
❑ Architecture

- Modele bazate doar pe encoder
 - ❑ Clasificare de propozitii
 - ❑ Clasificare de sentimente/emotii
 - ❑ Ex. BERT
- Modele bazate doar pe decoder
 - ❑ Generare de texte
 - ❑ Ex. GPT
- Modele bazate pe encoder-decoder (modele seq-to-seq)
 - ❑ Generare de texte care necesita un input (rezumat de text)
 - ❑ Ex. BART, T5



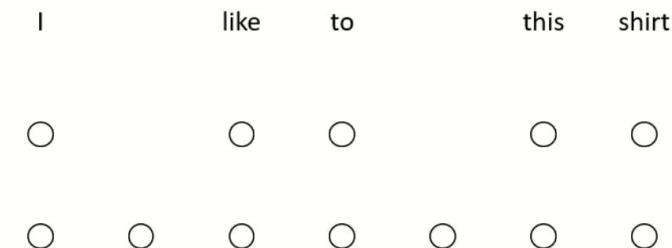
Transformers

□ Architecture

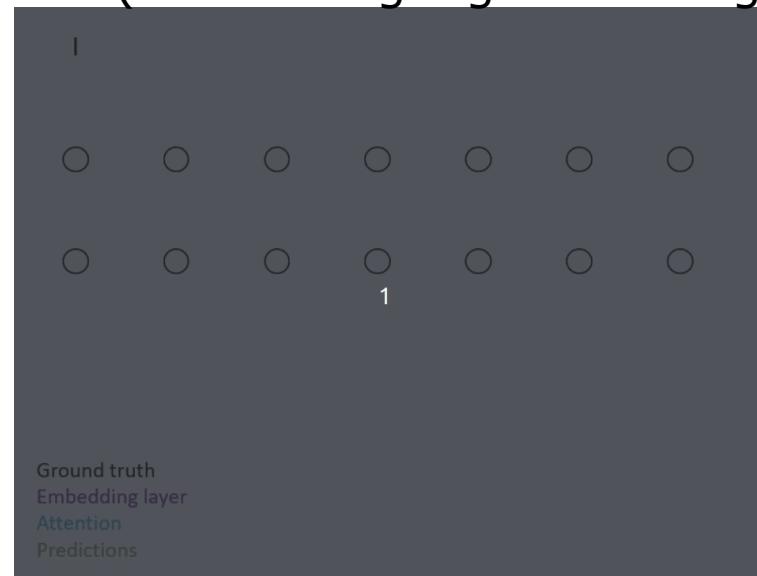


Transformers

- ❑ Architecture
- ❑ Pre-training – unsupervised
- Masked language modelling ■ Next word prediction
(causal language modeling)



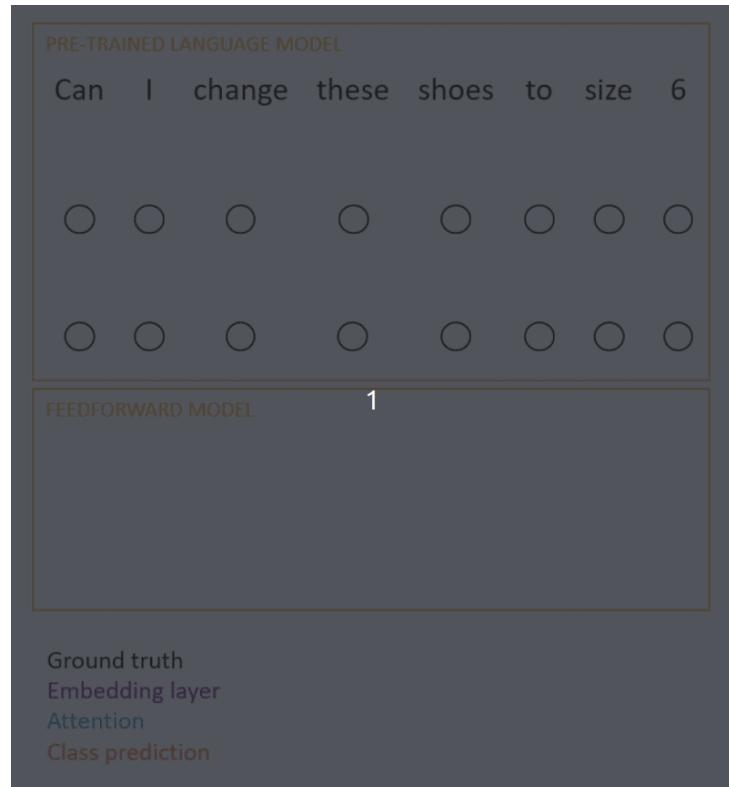
Ground truth
Embedding layer
Attention
Predictions



Ground truth
Embedding layer
Attention
Predictions

Transformers

- ❑ Architecture
- ❑ Pre-training – unsupervised
- ❑ Fine-tunning
 - Specific task
 - ❑ Another model -> Additional layers (params)



Transformers

- ❑ Architecture
 - ❑ Pre-training – unsupervised
 - ❑ Fine-tunning
 - ❑ Prompt-based learning
 - Simple
 - With examples (k-shot learning)

Identify the intent : Can I book a flight to London =>

A horizontal sequence of 20 small circles, arranged in two rows of 10 circles each. The circles are evenly spaced and have a thin black outline.

Ground truth
Embedding layer
Attention
Predictions

Translate French to English : chien => dog , chaise => chair , pomme =>

A 2x15 grid of 30 small circles, arranged in two rows of 15 circles each.

Ground truth
Embedding layer
Attention
Predictions

Mecanismul de “atentie” (attention)

flamanda

Broasca a mancat salata pentru ca ea era ...

delicioasa

Mecanismul de “atentie” (attention)

Broasca a mancat salata pentru ca ea era ...

flamanda

delicioasa



Mecanismul de “atentie” (attention)

Broasca a mancat salata pentru ca ea era ...

flamanda

delicioasa

În acvariu era o **broască** și un pește

S-a stricat **broasca** de la ușă

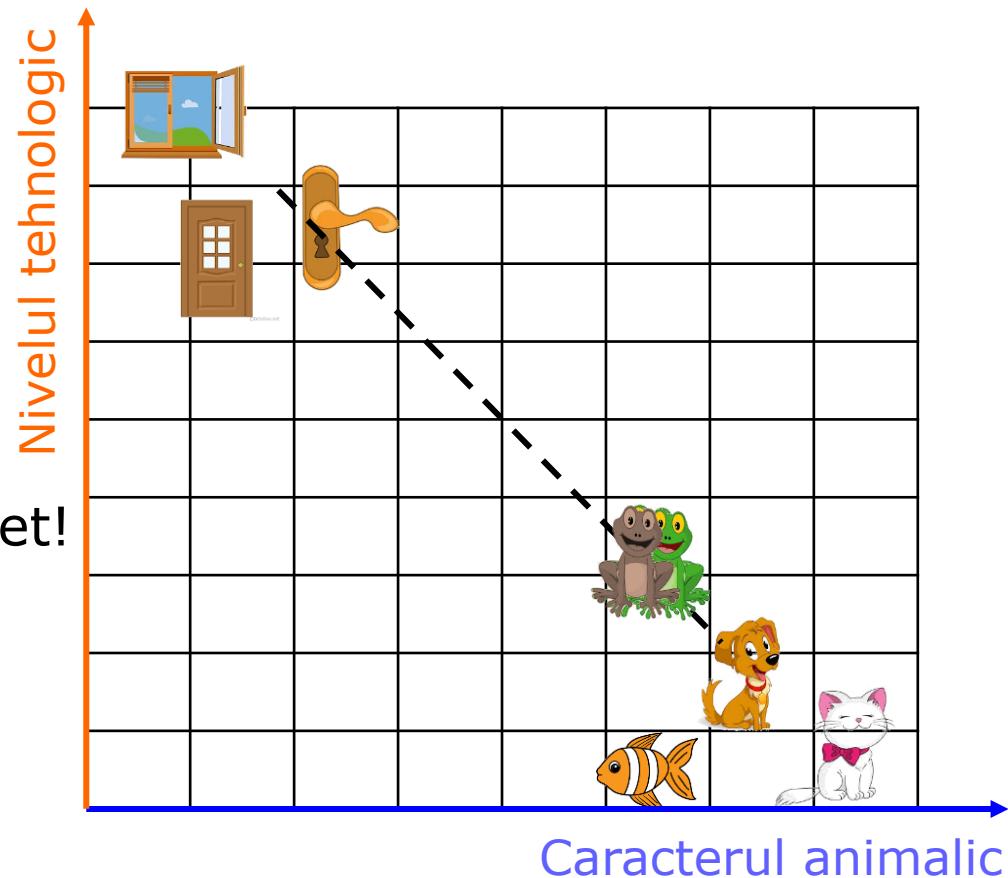
Mecanismul de “atentie” (attention)

■ Remember – embeddings

- cuvinte izolate precum: *geam, ușă, clanță, pisică, pește, câine, broască*
- Presupunem 2 atribute:
 - *caracterul animalic*
 - *nivelul tehnologic*

■ Attention

- Contextul e ca un magnet!
- Atrage cuvintele care se potrivesc!

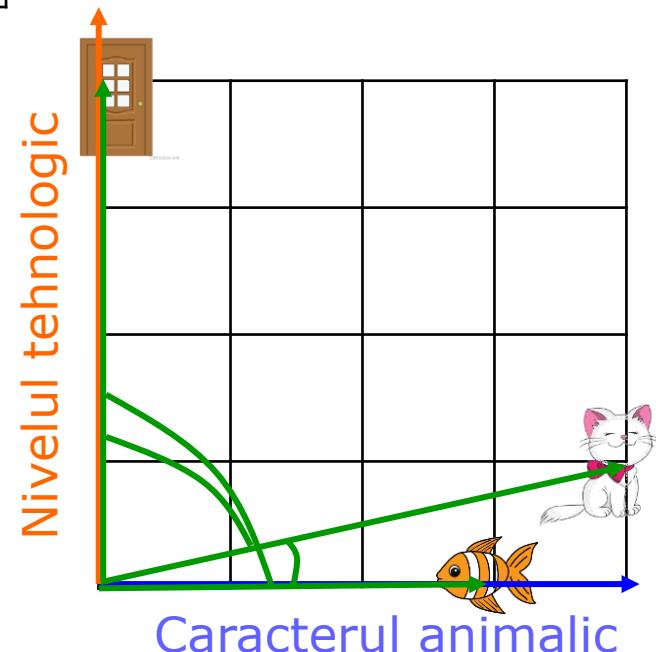


Mecanismul de “atentie” (attention)

- Similaritatea între cuvinte

Cuvântul	<i>caracterul animalic</i>	<i>nivelul tehnologic</i>
Pisică	4	1
Pește	3	0
Ușă	0	4

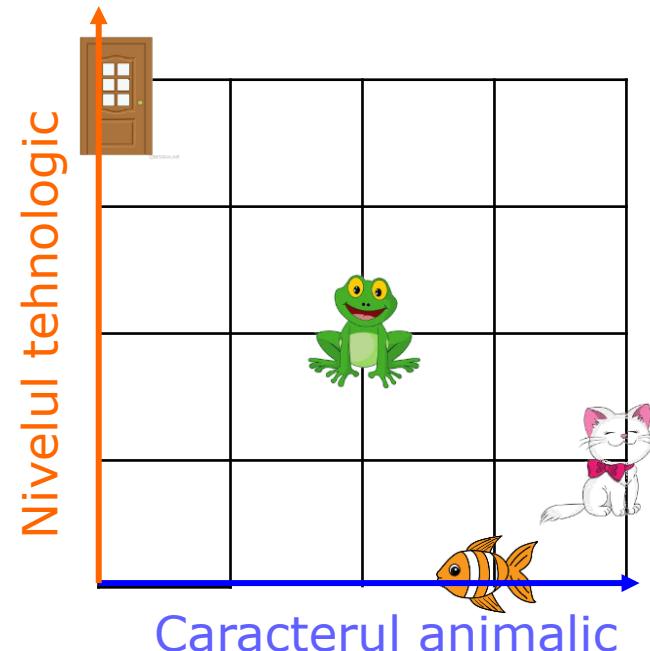
- sim (pisică, pește) = 12
 $\text{sim}([4,1], [3,0]) = 4*3 + 1*0 = 12$
- sim(pisică, ușă) = 4
 $\text{sim}([4,1],[0,4]) = 4*0 + 1*4 = 4$
- sim(pește, ușă) = 0
 $\text{sim}([3,0],[0,4]) = 3*0 + 0*4 = 0$



Mecanismul de “atentie” (attention)

- Contextul cuvintelor – matricea de afinitate
 - S-a stricat **broasca** de la **ușă**

	broască	ușă
broască		
ușă		



Mecanismul de “atentie” (attention)

- Contextul cuvintelor – matricea de afinitate
 - S-a stricat **broasca** de la **usă**

	broască	ușă
broască		
ușă		

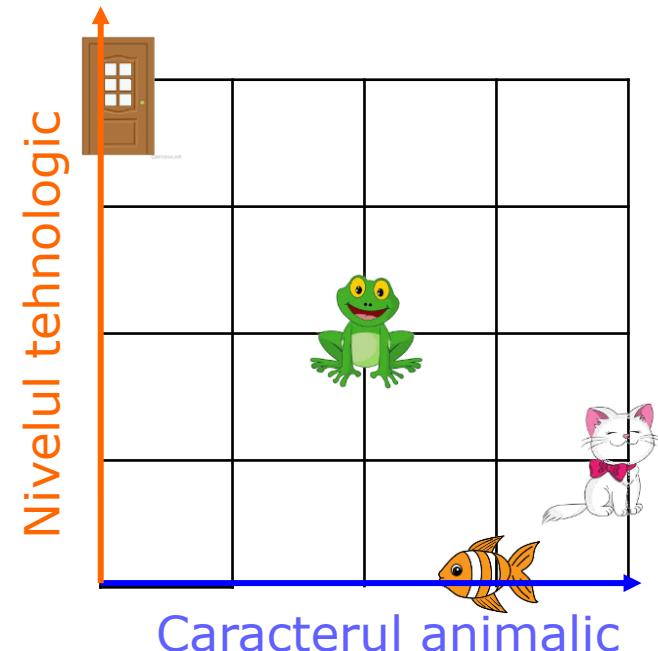
- $\text{sim}(\text{broască}, \text{broască}) = 1$

- $\text{sim}(\text{broască}, \text{ușă}) =$

$$\text{sim}([2,2], [0,4]) =$$

$$(2*0 + 2*4) / \sqrt{2} =$$

$$8 / \sqrt{2} = 5.65$$



Mecanismul de “atentie” (attention)

- Contextul cuvintelor – matricea de afinitate
 - S-a stricat **broasca** de la **usă**

	broască	ușă
broască	1	5.65
ușă	5.65	1

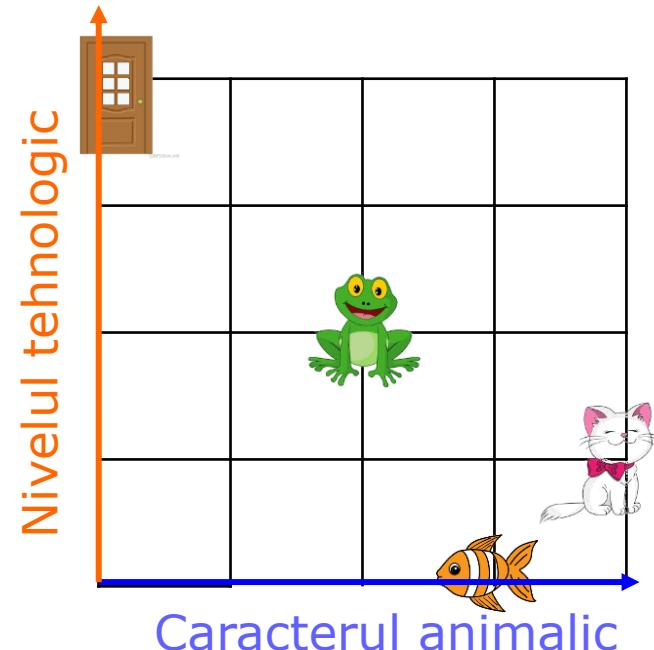
- $\text{sim}(\text{broască}, \text{broască}) = 1$

- $\text{sim}(\text{broască}, \text{ușă}) =$

$$\text{sim}([2,2], [0,4]) =$$

$$(2*0 + 2*4) / \sqrt{2} =$$

$$8 / \sqrt{2} = 5.65$$



Mecanismul de “atentie” (attention)

❑ Contextul cuvintelor – matricea de afinitate

- *S-a stricat **broasca** de la **ușă***

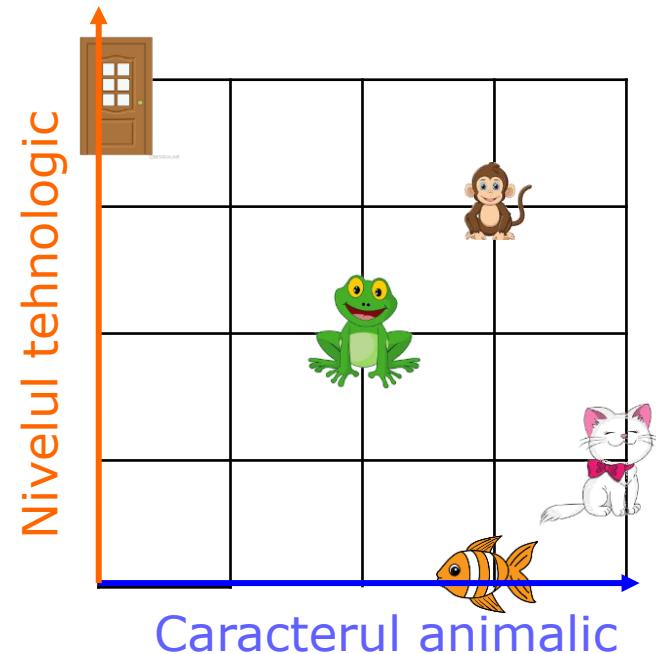
	broască	ușă
broască	1	5.65
ușă	5.65	1

❑ $\text{broască} = 1 * \text{broască} + 5.65 * \text{ușă}$

Mecanismul de “atentie” (attention)

- Contextul cuvintelor – matricea de afinitate
 - O **broască** s-a întâlnit cu o **maimuță**

	Broască	Maimuță
Broască		
Maimuță		

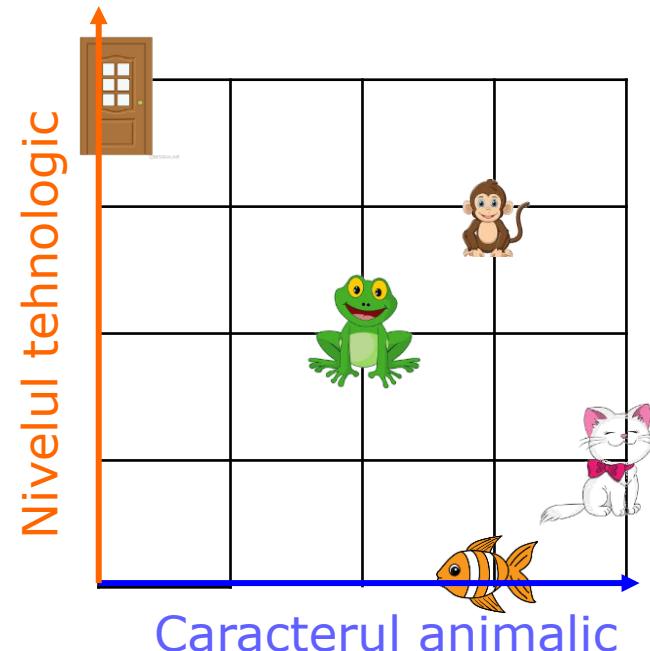


Mecanismul de “atentie” (attention)

- Contextul cuvintelor – matricea de afinitate
 - O **broască** s-a întâlnit cu o **maimuță**

	Broască	Maimuță
Broască		
Maimuță		

- $\text{sim}(\text{broască}, \text{broască}) = 1$
- $\text{sim}(\text{broască}, \text{maimuță}) =$
 $\text{sim}([2,2], [3,3]) =$
 $(2*3 + 2*3) / \sqrt{2} =$
 $12 / \sqrt{2} = 8.48$

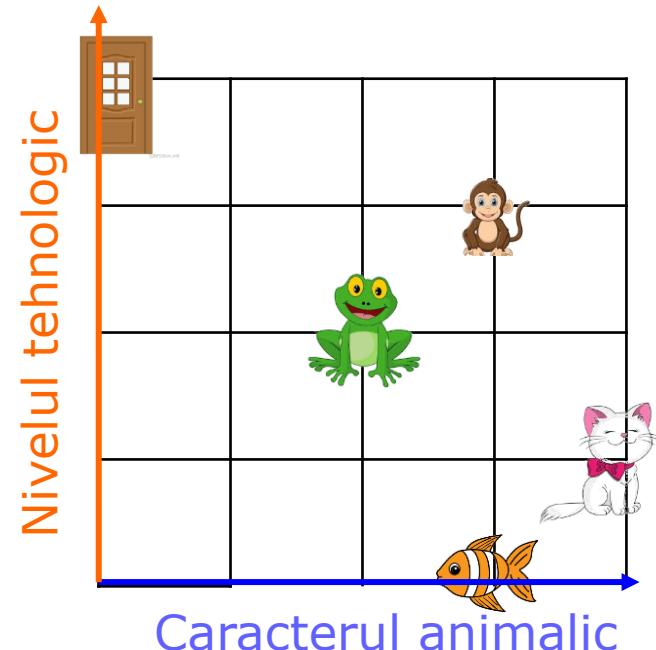


Mecanismul de “atentie” (attention)

- Contextul cuvintelor – matricea de afinitate
 - O **broască** s-a întâlnit cu o **maimuță**

	Broască	Maimuță
Broască	1	8.48
Maimuță	8.48	1

- $\text{sim}(\text{broască}, \text{broască}) = 1$
- $\text{sim}(\text{broască}, \text{maimuță}) = \text{sim}([2,2], [3,3]) = (2*3 + 2*3) / \sqrt{2} = 12 / \sqrt{2} = 8.48$



Mecanismul de “atentie” (attention)

❑ Contextul cuvintelor – matricea de afinitate

- O **broască** s-a întâlnit cu o **maimuță**

	broască	Maimuță
Broască	1	8.48
Maimuță	8.48	1

$$\square \text{broască} = 1 * \text{broască} + 8.48 * \text{maimuță}$$

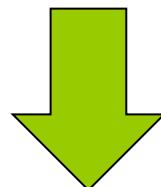
Mecanismul de “atentie” (attention)

□ Contextul cuvintelor – matricea de afinitate

- O **broască** s-a întâlnit cu o **maimuță**
- S-a stricat **broasca** de la **ușă**

broască = 1 * **broască** + 8.48 * maimuță

broască = 1 * **broască** + 5.65 * ușă



broască = ? * **broască** + ? * maimuță

broască = ? * **broască** + ? * ușă

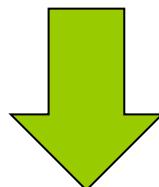
Mecanismul de “atentie” (attention)

□ Contextul cuvintelor – matricea de afinitate

- O **broască** s-a întâlnit cu o **maimuță**
- S-a stricat **broasca** de la **ușă**

broască = 1 * **broască** + 8.48 * maimuță

broască = 1 * **broască** + 5.65 * ușă



broască = 0.0005 * **broască** + 0.9994 * maimuță

broască = 0.0090 * **broască** + 0.9905 * ușă

Mecanismul de “atentie”

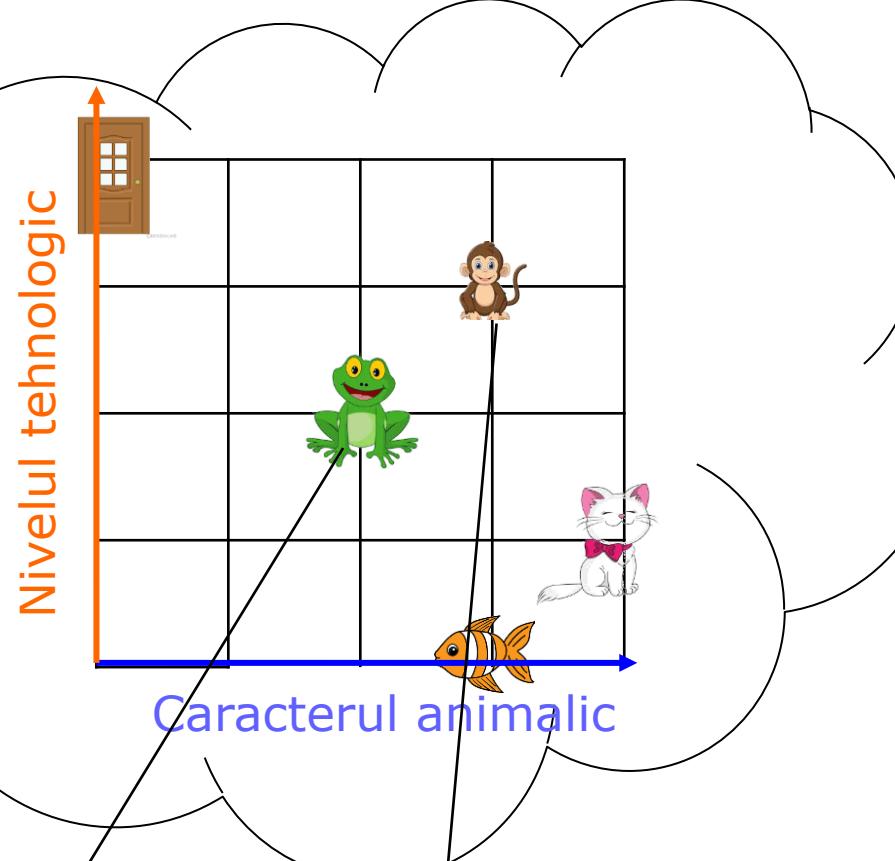
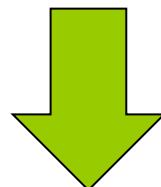
□ Contextul cuvintelor

■ O **broască** s-a în-

■ S-a stricat **broască**.

broască = 1 * **broască**

broască = 1 * **broască**



broască = 0.0005 * **broască** + 0.9994 * maimuță

broască = 0.0090 * **broască** + 0.9905 * ușă

broască = 0.0005 * [2,2] + 0.9994 * [3,3]

broască = 0.0090 * [2,2] + 0.9905 * [0,4]

Mecanismul de “atentie”

■ Contextul cuvintelor – matrice

■ O **broască** s-a întâlnit

■ S-a stricat **broasca** de

broască = 1 * **broască** + 4..

broască = 1 * **broască** + 5

broască = 0.0005 * **broască** + 0.9..

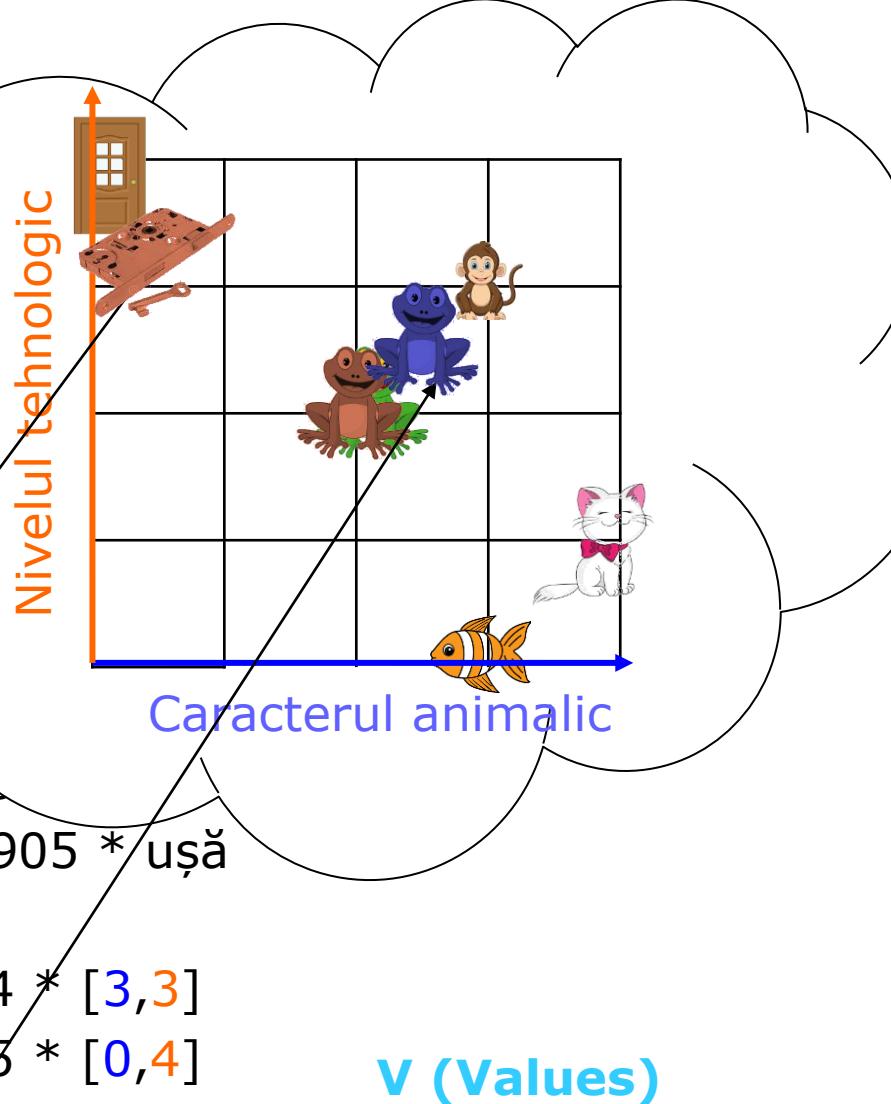
broască = 0.0090 * **broască** + 0.9905 * ușă

broască = 0.0005 * [2,2] + 0.9994 * [3,3]

broască = 0.0090 * [2,2] + 0.9905 * [0,4]

broască = [2.999, 2.999]

broască = [0.018, 3.981]



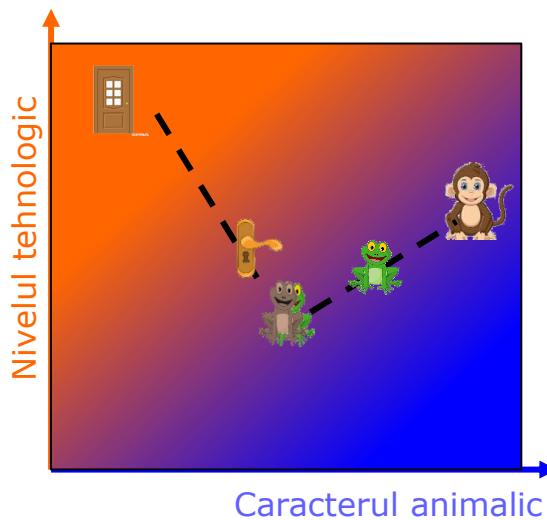
	Broască	Maimuță
Broască	0.0005	0.9994
Maimuță	0.9994	0.0005

Mecanismul de “atentie” (attention)

Un alt exemplu:

O **broască** s-a întâlnit cu o **maimuță**

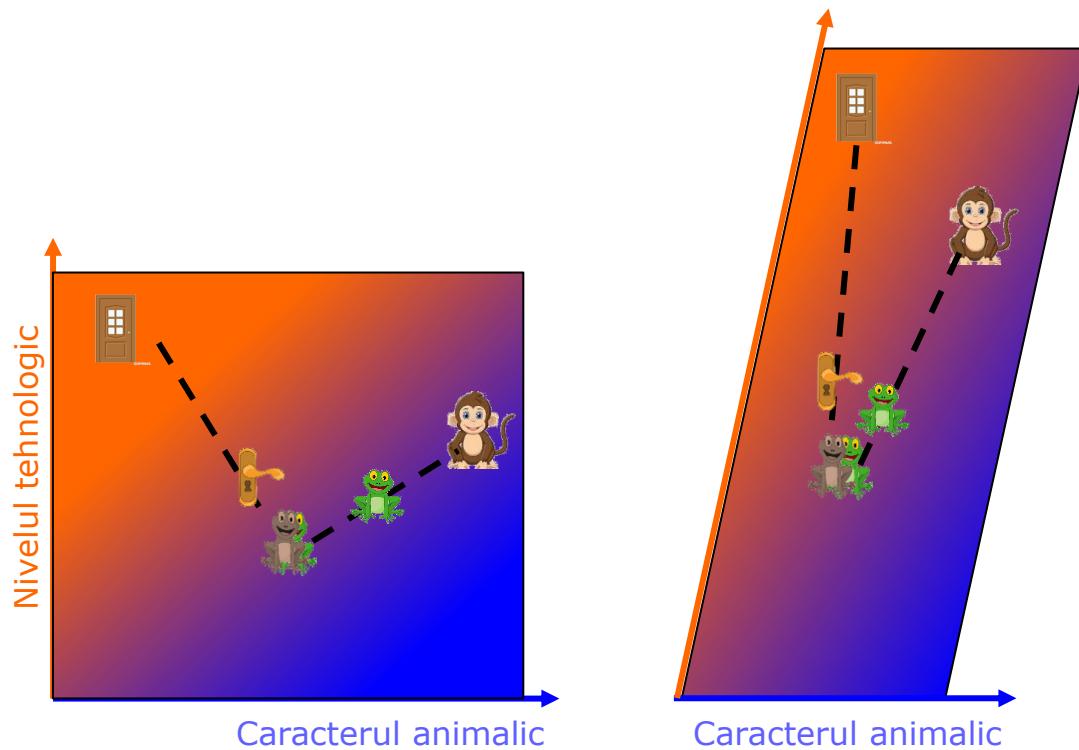
S-a stricat **broasca** de la **ușă**



Mecanismul de “atentie” (attention)

Un alt exemplu:

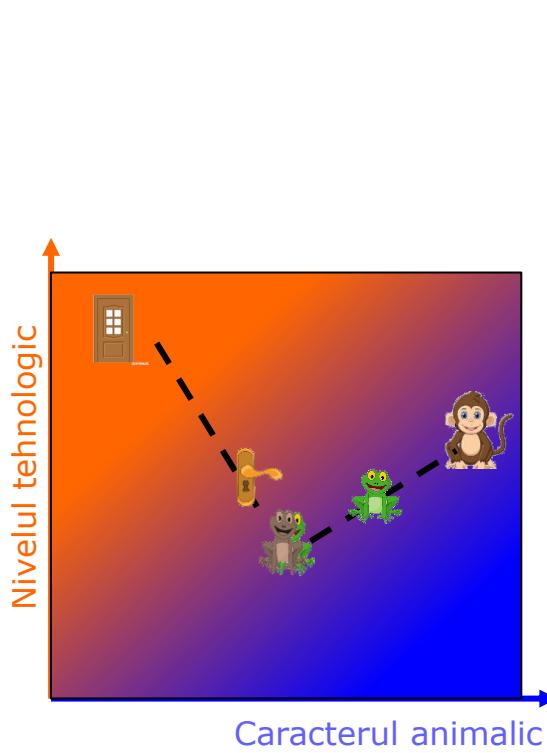
O **broască** s-a întâlnit cu o **maimuță**
S-a stricat **broasca** de la **ușă**



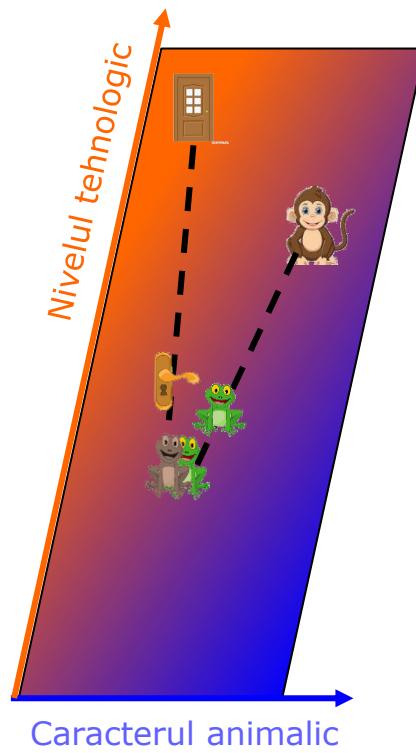
Mecanismul de “atentie” (attention)

Un alt exemplu:

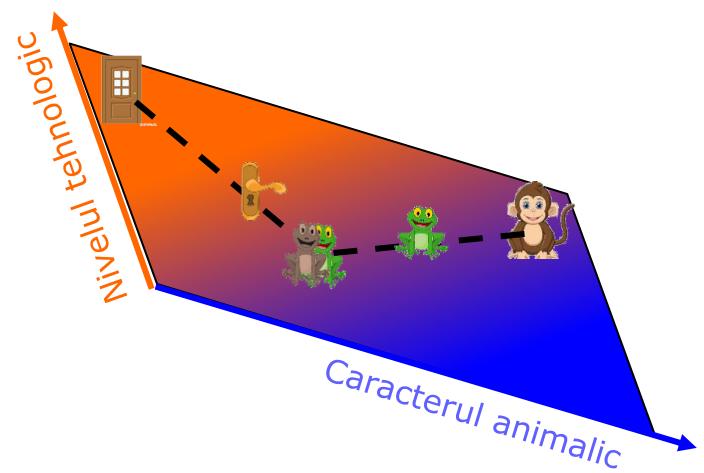
O **broască** s-a întâlnit cu o **maimuță**
S-a stricat **broasca** de la **ușă**



a) reprez1

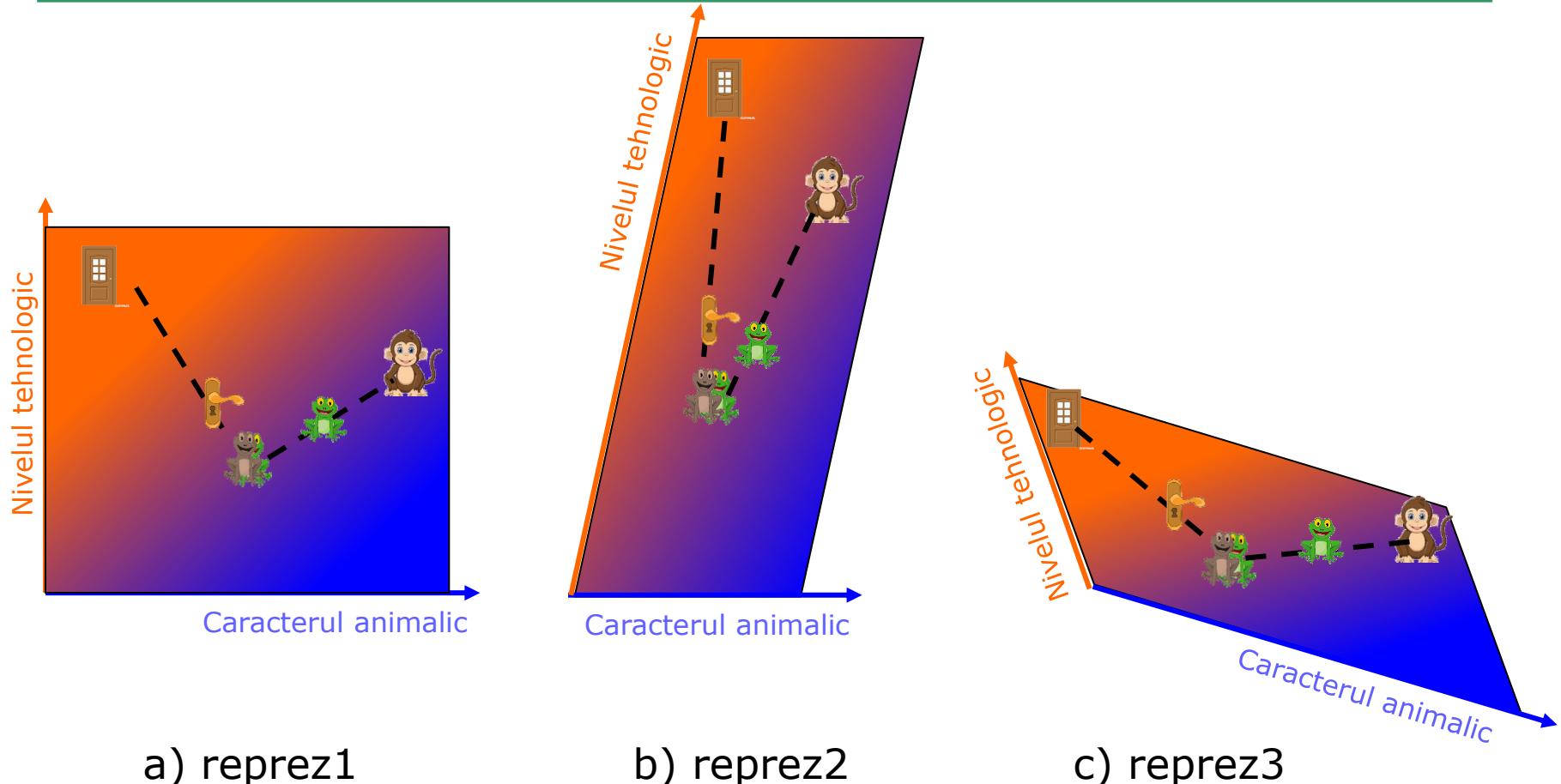


b) reprez2



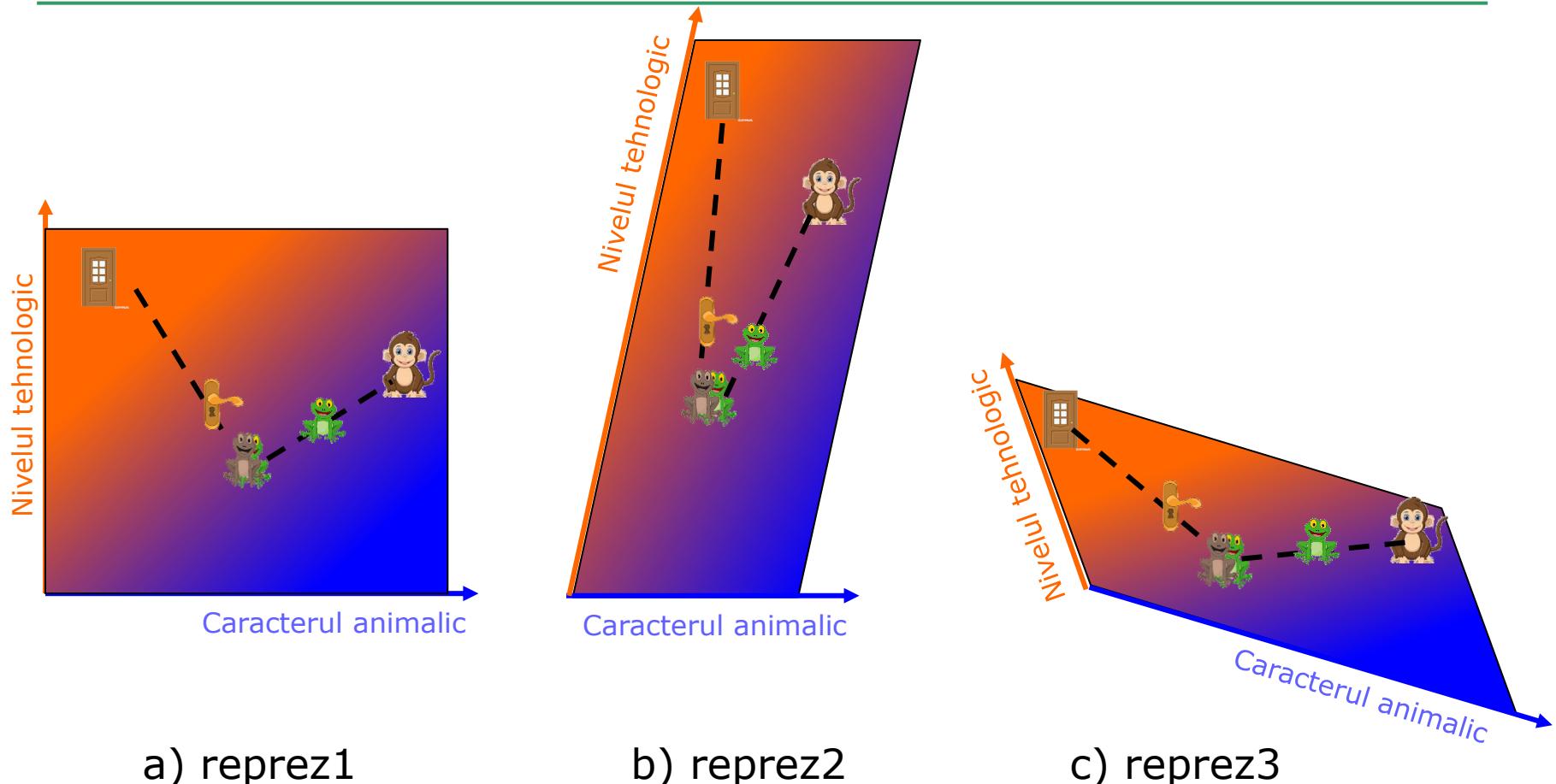
c) reprez3

Mecanismul de “atentie” (attention)



Q1: reprez1 = f(reprez2) = g(reprez3) ???
Da!!! Tranformare liniară!!!

Mecanismul de “atentie” (attention)



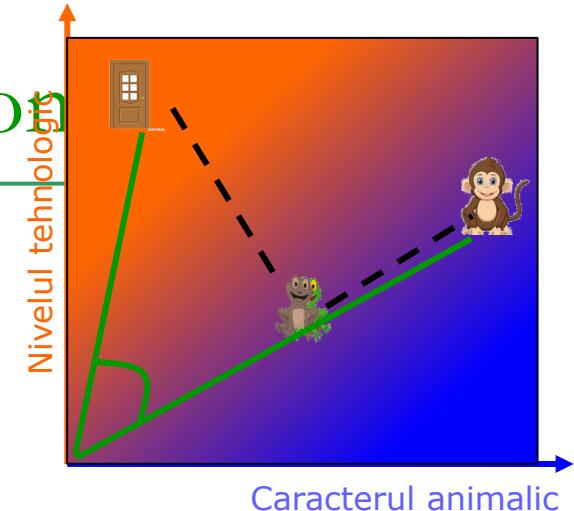
Q2: Care reprezentare e mai bună pentru a stabili similaritatea (deci a face diferența între sensurile lui broască)?

Reprezentarea 1 sau 3!

Mecanismul de “atentie” (attention)

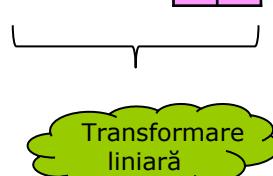
□ $\text{Sim}(\text{ușă}, \text{maimuță}) =$

$$\text{sim}(\begin{matrix} \text{purple} & \text{orange} \\ \text{purple} & \text{orange} \end{matrix}, \begin{matrix} \text{blue} & \text{orange} \\ \text{blue} & \text{orange} \end{matrix}) = \begin{matrix} \text{purple} & \text{orange} \\ \text{blue} & \text{orange} \end{matrix}$$



□ $\text{Sim}(\text{ușă}, \text{maimuță}) =$

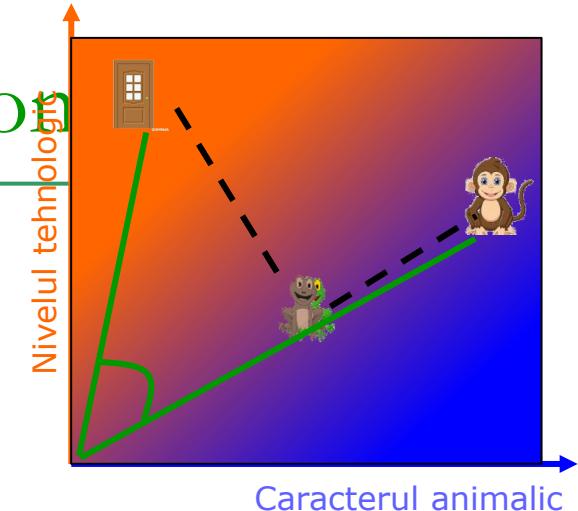
$$\text{sim}(\begin{matrix} \text{purple} & \text{orange} \\ \text{purple} & \text{orange} \end{matrix}, \begin{matrix} \text{blue} & \text{orange} \\ \text{blue} & \text{orange} \end{matrix}) = \begin{matrix} \text{purple} & \text{orange} \\ \text{blue} & \text{orange} \end{matrix} \quad \begin{matrix} \text{green} & \text{green} & \text{green} \\ \text{green} & \text{green} & \text{green} \\ \text{green} & \text{green} & \text{green} \end{matrix} \quad \begin{matrix} \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \end{matrix} \quad \begin{matrix} \text{blue} & \text{orange} \end{matrix}$$



Mecanismul de “atentie” (attention)

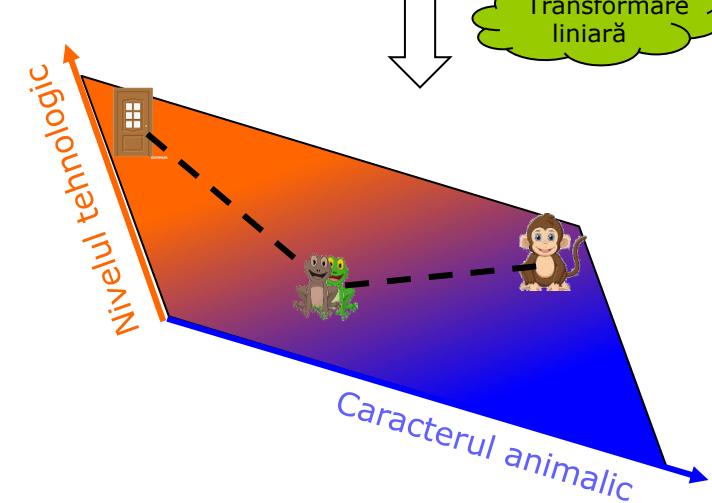
□ $\text{Sim}(\text{ușă}, \text{maimută}) =$

$$\text{sim}([\text{purple}, \text{orange}], [\text{blue}, \text{orange}]) = \begin{matrix} \text{purple} & \text{orange} \\ \text{blue} & \text{orange} \end{matrix}$$



□ $\text{Sim}(\text{ușă}, \text{maimută}) =$

$$\text{sim}([\text{purple}, \text{orange}], [\text{blue}, \text{orange}]) = \begin{matrix} \text{purple} & \text{orange} \\ \text{blue} & \text{orange} \end{matrix} \quad \begin{matrix} \text{green} \\ \text{green} \\ \text{green} \end{matrix} \quad \begin{matrix} \text{pink} \\ \text{pink} \\ \text{pink} \end{matrix} \quad \begin{matrix} \text{blue} \\ \text{orange} \end{matrix}$$



$$K (\text{Key}) = \begin{matrix} \text{green} & \text{green} & \text{green} \\ \text{green} & \text{green} & \text{green} \\ \text{green} & \text{green} & \text{green} \end{matrix}$$

$$Q (\text{Queries}) = \begin{matrix} \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \end{matrix}$$

Valorile optime - ANN

Mecanismul de “atentie”

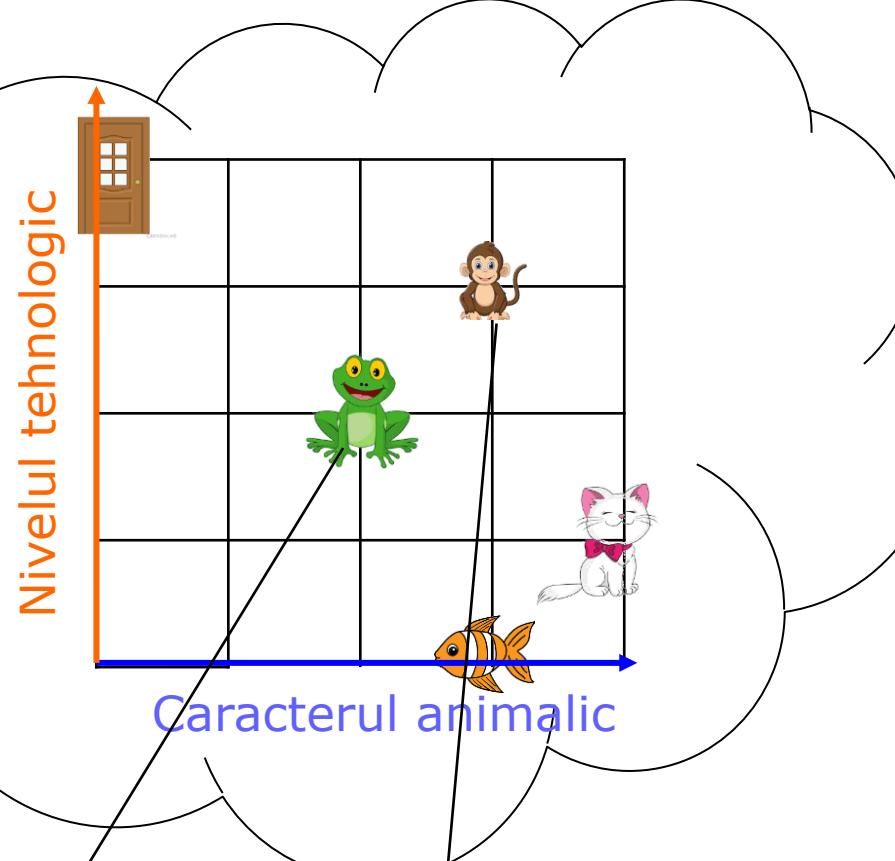
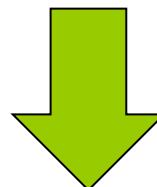
□ Contextul cuvintelor

■ O **broască** s-a în-

■ S-a stricat **broască**.

broască = 1 * **broască**

broască = 1 * **broască**



broască = 0.0005 * **broască** + 0.9994 * maimuță

broască = 0.0090 * **broască** + 0.9905 * ușă

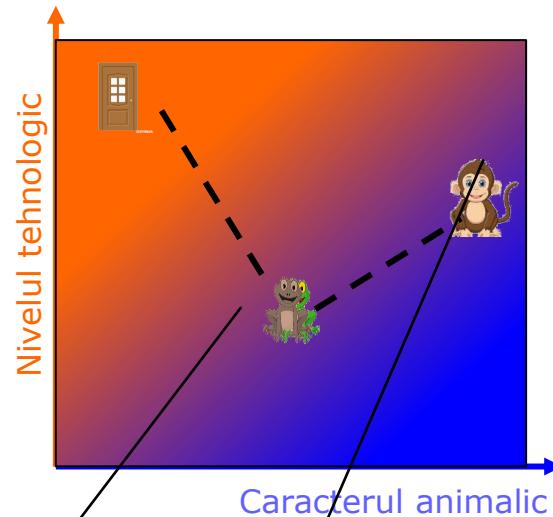
broască = 0.0005 * [2,2] + 0.9994 * [3,3]

broască = 0.0090 * [2,2] + 0.9905 * [0,4]

Mecanismul de “atentie”

□ Contextul cuvintelor

- O **broască** s-a în-
- S-a stricat **broască**.



broască = 1 * **broască** + 4.24 * maimuță

broască = 1 * **broască** + 5.65 * ușă

broască = 0.0005 * **broască** + 0.9994 * maimuță

broască = 0.0090 * **broască** + 0.9905 * ușă

broască = 0.0005 * [2,2] + 0.9994 * [3,3]

broască = 0.0090 * [2,2] + 0.9905 * [0,4]

Mecanismul de “atenție”

□ Contextul cuvintelor

- O **broască** s-a în-
- S-a stricat **broască**.



broască = 1 * **broască** + 4.24 * maimuță

broască = 1 * **broască** + 5.65 * ușă

broască = 0.0005 * **broască** + 0.9994 * maimuță

broască = 0.0090 * **broască** + 0.9905 * ușă

broască = [2.999, 2.999]

broască = [0.018, 3.981]

Mecanismul de “atentie” (attention mechanism)

□ Contextul cuvintelor –

- O **broască** s-a întâlnit
- S-a stricat **broasca** de

$$\text{broască} = 1 * \text{broască} + 4.24 * \text{maimuță}$$

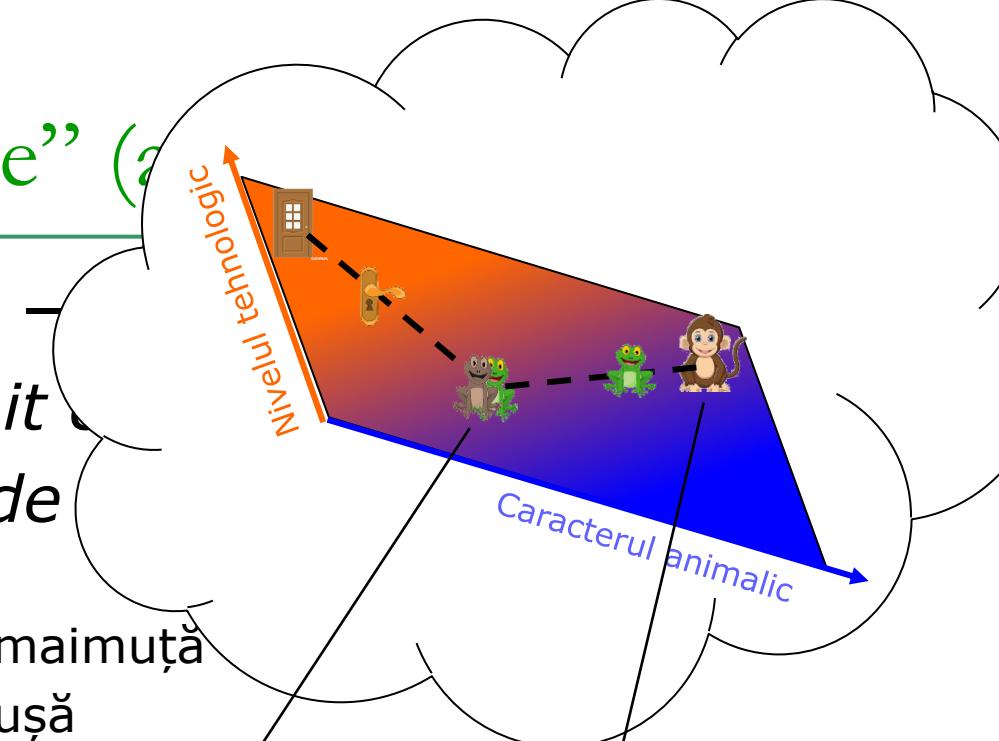
$$\text{broască} = 1 * \text{broască} + 5.65 * \text{ușă}$$

$$\text{broască} = 0.0005 * \text{broască} + 0.9994 * \text{maimuță}$$

$$\text{broască} = 0.0090 * \text{broască} + 0.9905 * \text{ușă}$$

$$\text{broască} = 0.0005 * [\text{broască}(KQ, KQ)] + 0.9994 * [\text{maimuță}(KQ, KQ)]$$

$$\text{broască} = 0.0090 * [\text{broască}(KQ, KQ)] + 0.9905 * [\text{ușă}(KQ, KQ)]$$



Mecanismul de “atentie” (2)

□ Contextul cuvintelor –

- O **broască** s-a întâlnit
- S-a stricat **broasca** de

broască = 1 * broască + 4.24 * maimuță

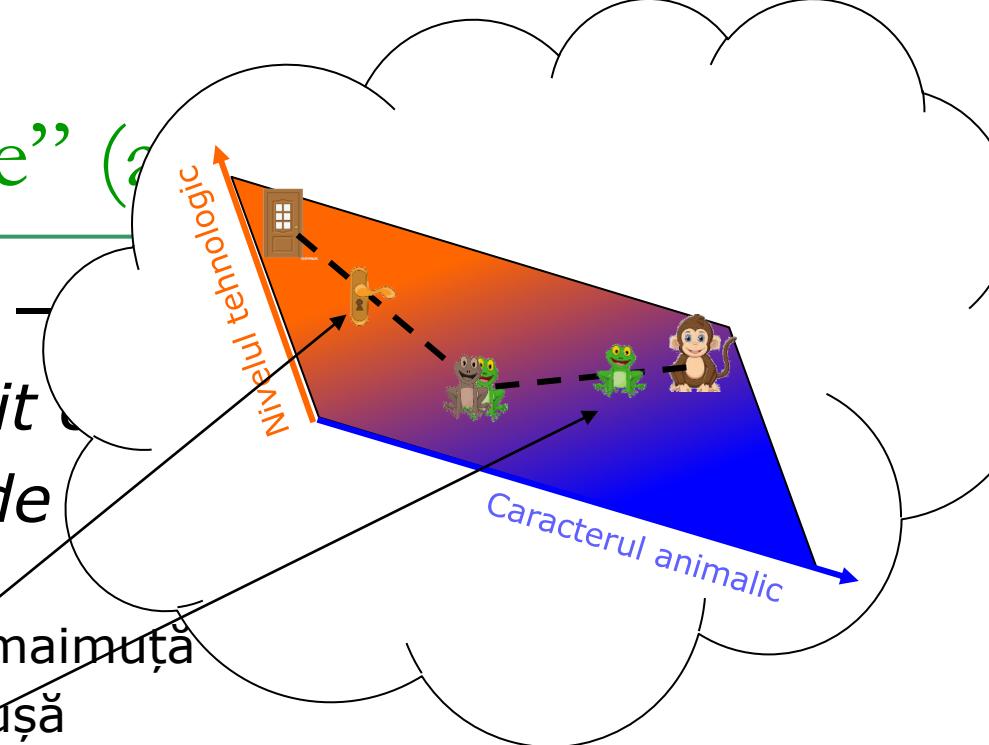
broască = 1 * broască + 5.65 * ușă

broască = 0.0005 * broască + 0.9994 * maimuță

broască = 0.0090 * broască + 0.9905 * ușă

broască = 0.0005 * [broască(KQ,KQ)] + 0.9994 * [maimuță(KQ,KQ)]

broască = 0.0090 * [broască(KQ,KQ)] + 0.9905 * [ușă(KQ,KQ)]



Mecanismul de “atentie” (attention mechanism)

■ Contextul cuvintelor – măsură de similaritate

- O **broască** s-a întâlnit
- S-a stricat **broasca** de

broască = 1 * broască + 4.24 * maimuță

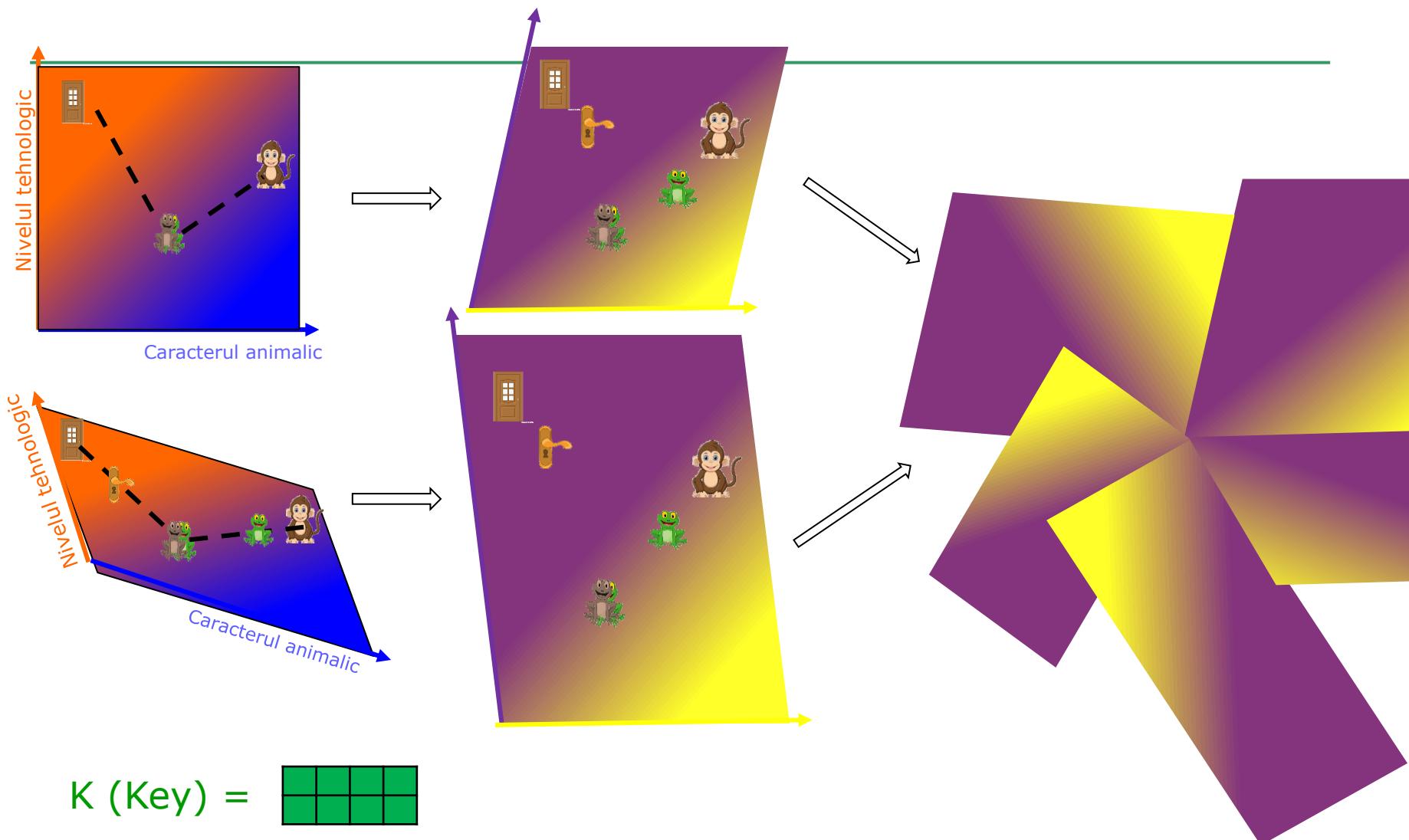
broască = 1 * broască + 5.65 * ușă

broască = 0.0005 * broască + 0.9994 * maimuță

broască = 0.0090 * broască + 0.9905 * ușă

broască = 0.0005 * [broască(**KQ**,**KQ**)] + 0.9994 * [maimuță(**KQ**,**KQ**)]

broască = 0.0090 * [broască(**KQ**,**KQ**)] + 0.9905 * [ușă(**KQ**,**KQ**)]



K (Key) =

Q (Queries) =

V (Values)

Mecanismul de “atentie” (attention)

- Identificarea celor mai bune embedding-uri pentru a stabili similaritatea intre elemente

- Se bazează pe feature-urile elementelor (e.g. Caracterul animalic, nivelul tehnologic)

$$K \text{ (Key)} = \begin{array}{|c|c|c|}\hline \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \hline \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \hline \textcolor{green}{\square} & \textcolor{green}{\square} & \textcolor{green}{\square} \\ \hline \end{array}$$

$$Q \text{ (Queries)} = \begin{array}{|c|c|c|}\hline \textcolor{magenta}{\square} & \textcolor{magenta}{\square} & \textcolor{magenta}{\square} \\ \hline \textcolor{magenta}{\square} & \textcolor{magenta}{\square} & \textcolor{magenta}{\square} \\ \hline \textcolor{magenta}{\square} & \textcolor{magenta}{\square} & \textcolor{magenta}{\square} \\ \hline \end{array}$$

- Identificarea celor mai bune embedding-uri pentru a prezice următorul cuvânt

- Se bazează pe co-apariția elementelor în același context

$$V \text{ (Values)} = \begin{array}{|c|c|c|}\hline \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \hline \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \hline \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \hline \end{array}$$

Mecanismul de “atentie” (attention)

- Input = embedding-uri (de lungime n) pt v cuvinte – matrice $v \times n$
- V – matrice $v \times v$ v – nr de cuvinte
- K – matrice $n \times d$ n – nr de features (lungimea unui embedding)
- Q – matrice $n \times d$ d – nr de features abstracte

Features	x_1	x_2	x_3	x_4
	2	0	0	2
	0	1	0	0
	0	2	1	0
	0	0	1	1
	2	0	0	0
	1	0	1	1

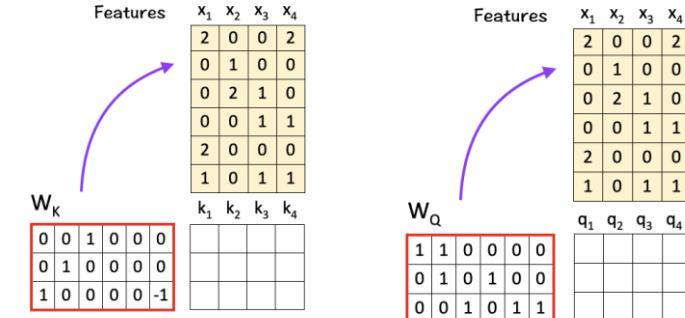
Mecanismul de “atentie” (attention)

- Input = embedding-uri (de lungime n) pt v cuvinte – matrice $v \times n$
- V – matrice $v \times v$ v – nr de cuvinte
- K – matrice $n \times d$ n – nr de features (lungimea unui embedding)
- Q – matrice $n \times d$ d – nr de features abstracte

Features	x_1	x_2	x_3	x_4
	2	0	0	2
	0	1	0	0
	0	2	1	0
	0	0	1	1
	2	0	0	0
	1	0	1	1

Transformare

$$\begin{aligned} \text{input} &\rightarrow \text{input} \times K \times Q^T \times \text{input}^T \times V \times \text{input} \rightarrow \text{features} \\ (v,n) &\rightarrow ((v,n) \times (n,d)) \times ((d,n) \times (n,v)) \times ((v,v) \times (v,n)) \\ (v,n) &\rightarrow (v,d) \quad \times \quad (d,v) \quad \times \quad (v,n) \end{aligned}$$



Mecanismul de “atentie” (attention)

- Input = embedding-uri (de lungime n) pt v cuvinte – matrice $v \times n$
- V – matrice $v \times v$ v – nr de cuvinte
- K – matrice $n \times d$ n – nr de features (lungimea unui embedding)
- Q – matrice $n \times d$ d – nr de features abstracte

Features	x_1	x_2	x_3	x_4
	2	0	0	2
	0	1	0	0
	0	2	1	0
	0	0	1	1
	2	0	0	0
	1	0	1	1

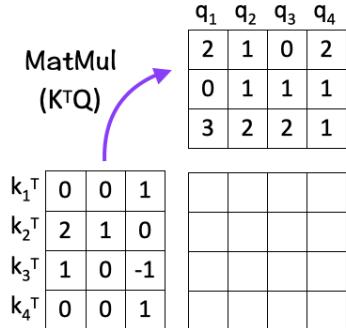
Transformare

$$\text{input} \rightarrow \text{input} \times K \times Q^T \times \text{input}^T \times V \times \text{input} \rightarrow \text{features}$$

$$(v,n) \rightarrow ((v,n) \times (n,d)) \times ((d,n) \times (n,v)) \times ((v,v) \times (v,n))$$

$$(v,n) \rightarrow (v,d) \quad \times \quad (d,v) \quad \times \quad (v,n)$$

$$(v,n) \rightarrow \quad \quad \quad (v,v) \quad \quad \quad \times \quad \quad \quad (v,n)$$



Mecanismul de “atentie” (attention)

- Input = embedding-uri (de lungime n) pt v cuvinte – matrice $v \times n$
- V – matrice $v \times v$ v – nr de cuvinte
- K – matrice $n \times d$ n – nr de features (lungimea unui embedding)
- Q – matrice $n \times d$ d – nr de features abstracte

Features	x_1	x_2	x_3	x_4
	2	0	0	2
	0	1	0	0
	0	2	1	0
	0	0	1	1
	2	0	0	0
	1	0	1	1

Transformare

$$\text{input} \rightarrow \text{input} \times K \times Q^T \times \text{input}^T \times V \times \text{input} \rightarrow \text{features}$$

$$(v,n) \rightarrow ((v,n) \times (n,d)) \times ((d,n) \times (n,v)) \times ((v,v) \times (v,n))$$

$$(v,n) \rightarrow (v,d) \quad \times \quad (d,v) \quad \times \quad (v,n)$$

$$(v,n) \rightarrow \quad \quad \quad (v,v) \quad \quad \quad \times \quad \quad (v,n)$$

Scale

$$\frac{\square}{\sqrt{dk}} \approx \frac{\square}{\sqrt{2}}$$

1	1	1	0
2	1	0	2
0	0	-1	0
1	1	1	0

Scalare

$$\text{input} \rightarrow \text{input} \times K \times Q^T \times \text{input}^T \times V \times \text{input} \rightarrow \text{features}$$

$$(v,n) \rightarrow ((v,n) \times (n,d)) \times ((d,n) \times (n,v)) \times ((v,v) \times (v,n))$$

$$(v,n) \rightarrow (v,d) \quad \times \quad (d,v) \quad \times \quad (v,n)$$

$$(v,n) \rightarrow \quad \quad \quad (v,v) / \sqrt{d} \quad \quad \quad \times \quad \quad (v,n)$$

Normalizare

$$\text{input} \rightarrow \text{input} \times K \times Q^T \times \text{input}^T \times V \times \text{input} \rightarrow \text{features}$$

$$(v,n) \rightarrow ((v,n) \times (n,d)) \times ((d,n) \times (n,v)) \times ((v,v) \times (v,n))$$

$$(v,n) \rightarrow (v,d) \quad \times \quad (d,v) \quad \times \quad (v,n)$$

$$(v,n) \rightarrow \text{softmax}((v,v) / \sqrt{d}) \quad \times \quad (v,n)$$

Mecanismul de “atentie” (attention)

- Input = embedding-uri (de lungime n) pt v cuvinte – matrice $v \times n$
- V – matrice $v \times v$ v – nr de cuvinte
- K – matrice $n \times d$ n – nr de features (lungimea unui embedding)
- Q – matrice $n \times d$ d – nr de features abstracte

Features	x_1	x_2	x_3	x_4
	2	0	0	2
	0	1	0	0
	0	2	1	0
	0	0	1	1
	2	0	0	0
	1	0	1	1

Transformare

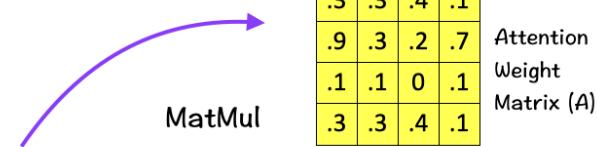
input \rightarrow input $\times K \times Q^T \times \text{input}^T \times V \times \text{input} \rightarrow \text{features}$

$(v, n) \rightarrow ((v, n) \times (n, d)) \times ((d, n) \times (n, v)) \times ((v, v)$

$(v, n) \rightarrow (v, d) \quad \times \quad (d, v) \quad \times$

$(v, n) \rightarrow (v, v) \quad \quad \quad \times$

$(v, n) \rightarrow \quad \quad \quad (v, n)'$



Scalare

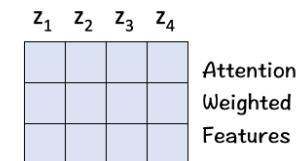
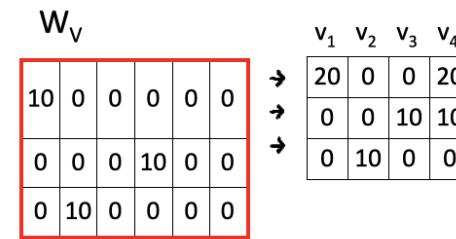
input \rightarrow input $\times K \times Q^T \times \text{input}^T \times V \times \text{input} \rightarrow \text{fe}$

$(v, n) \rightarrow ((v, n) \times (n, d)) \times ((d, n) \times (n, v)) \times ((v, v)$

$(v, n) \rightarrow (v, d) \quad \times \quad (d, v) \quad \times$

$(v, n) \rightarrow (v, v) / \sqrt{v} \quad \quad \quad \times$

$(v, n) \rightarrow \quad \quad \quad (v, n)'$



Normalizare

input \rightarrow input $\times K \times Q^T \times \text{input}^T \times V \times \text{input} \rightarrow \text{features}$

$(v, n) \rightarrow ((v, n) \times (n, d)) \times ((d, n) \times (n, v)) \times ((v, v) \times (v, n))$

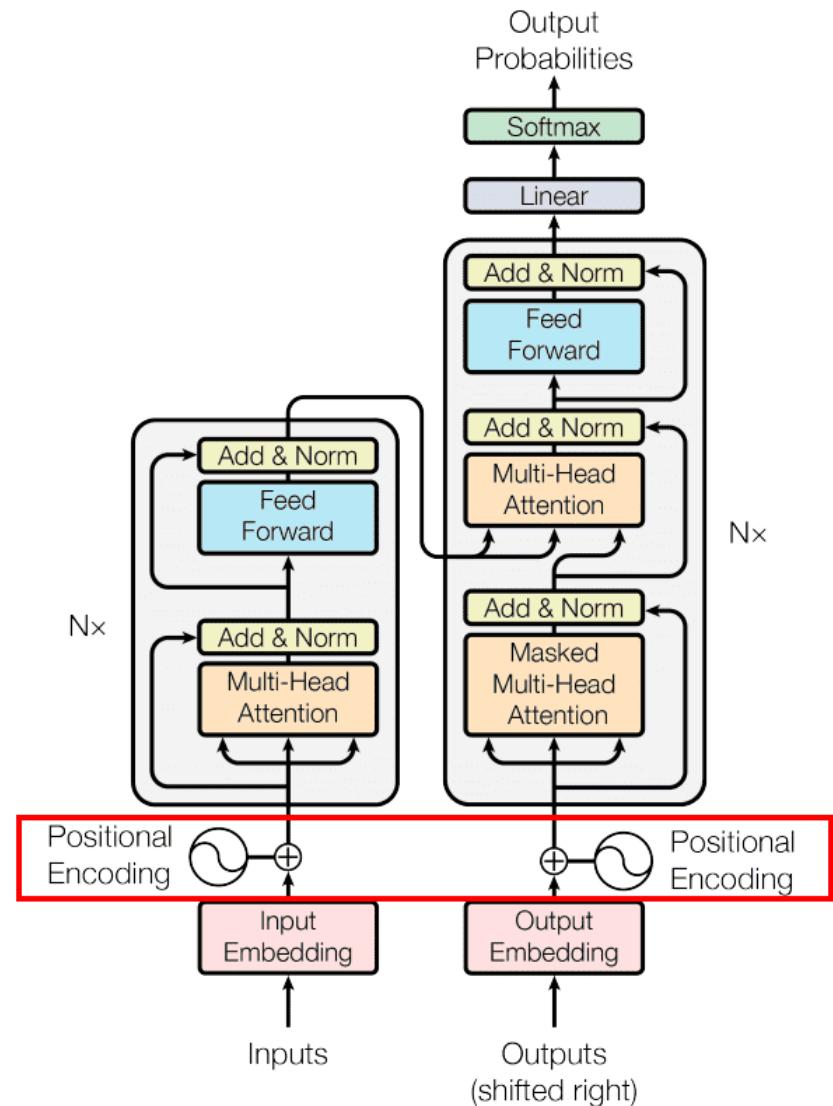
$(v, n) \rightarrow (v, d) \quad \times \quad (d, v) \quad \times \quad (v, n)$

$(v, n) \rightarrow \text{softmax}((v, v) / \sqrt{v}) \quad \times \quad (v, n)$

$(v, n) \rightarrow \quad \quad \quad (v, n)'$

Transformer

Positional Embedding



Positional embedding

bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [8].

In this work, we use sine and cosine functions of different frequencies:

$$\begin{aligned} PE_{(pos,2i)} &= \sin(pos/10000^{2i/d_{\text{model}}}) \\ PE_{(pos,2i+1)} &= \cos(pos/10000^{2i/d_{\text{model}}}) \end{aligned}$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k , PE_{pos+k} can be represented as a linear function of PE_{pos} .

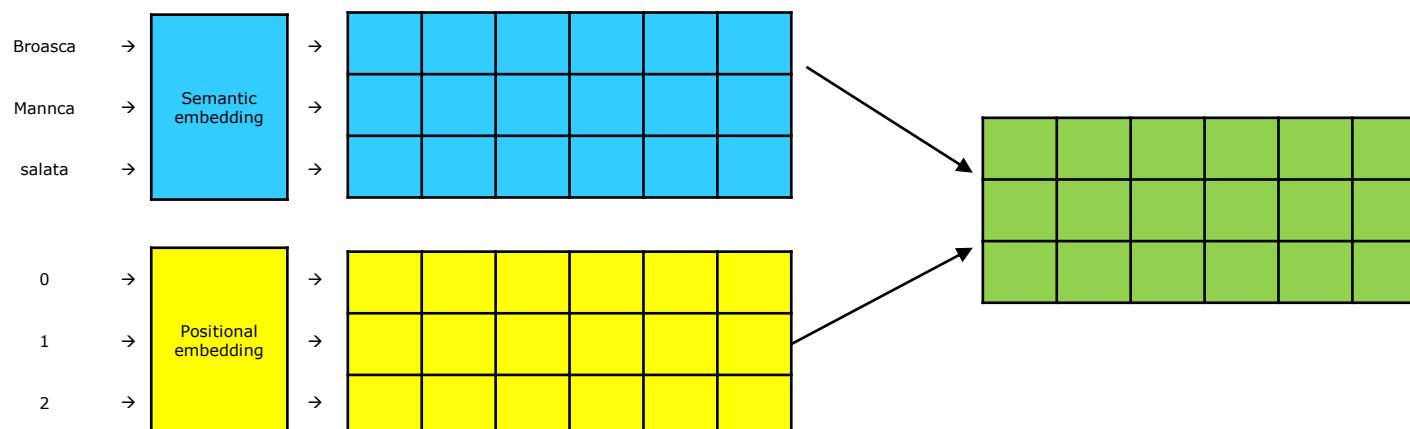
Positional embedding

- ❑ Cuvinte → Embeddings
- ❑ Pentru o propoziție cu n cuvinte, se obțin n embedding-uri (de lungime d)
 - Care țin cont de semantica cuvintelor
 - ❑ Word Embeddings
 - descoperite / învățate
 - prin ML (Word2Vec, GloVe, etc.)
 - Care țin cont de poziția (absolută sau relativă) a cuvintelor în propoziției
 - ❑ Positional Embeddings
 - calculate
 - Cum?

Positional embedding

❑ Exemplu

- O propoziție cu n cuvinte
- Fiecăruia cuvânt i se asociază
 - ❑ Un embedding semantic SE de lungime d
 - ❑ Un embedding pozitional PE de lungime d
 - ❑ Embedding-ul final corespunzător cuvântului va fi suma celor 2 (WE + PE) $\rightarrow n \times d$ valori



Positional embedding

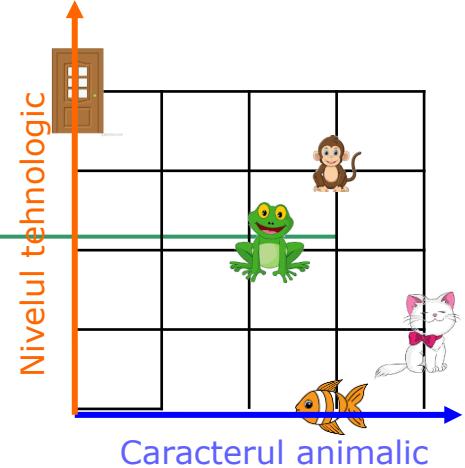
□ Ce valori conține PE?

■ Principii

- Embedding-urile trebuie să păstreze distanța originală între cuvinte
 - $\text{dist}(\text{Emb}(\text{câine}), \text{Emb}(\text{pisică})) < \text{dist}(\text{Emb}(\text{câine}), \text{Emb}(\text{fereastră}))$
 - $\text{Dist}(\text{Emb}(\text{word}_k), \text{Emb}(\text{word}_{k+1})) = \text{Dist}(\text{Emb}(\text{word}_p), \text{Emb}(\text{word}_{p+1}))$
- Ortogonalitate - Embedding-urile cuvintelor ne-conectate să fie perpendiculare
 - $\text{Emb}(\text{uşă}) \perp \text{Emb}(\text{pește})$
 - Similaritate (uşă , pește) = 0
- Embedding-uri independente de lungimea propoziției
 - → putere de generalizare (train vs. test)
 - → *bounded*

■ Reprezentări / valori posibile

- Poziția efectivă (1,2,3,4)
- Poziția în reprezentare 1-hot encoding
- Poziția în reprezentare binară
- Poziția în reprezentare polară



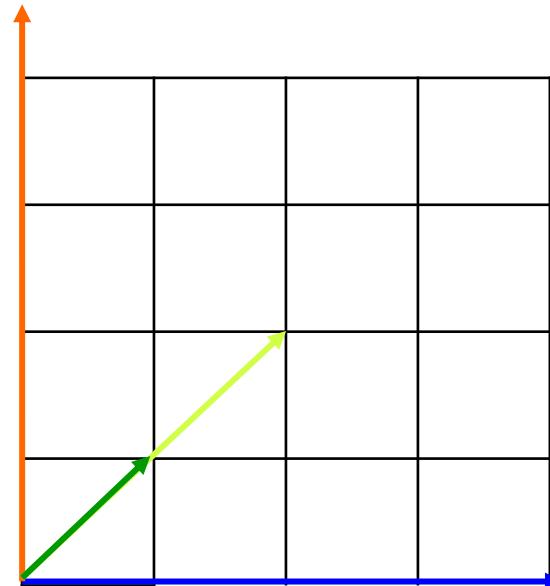
Pozitia in propozitie

❑ Exemplu

- O propoziție cu 5 cuvinte și $d = 1$
- $\text{PE}(\text{cuv}) = \text{poziția cuvântului}$

Dist	w0	w1	w2	w3	w4
w0	0	1	2	3	4
w1	1	0	1	2	3
w2	2	1	0	1	2
w3	3	2	1	0	1
w4	4	3	2	1	0

0
1
2
3
4



sim($\text{PE}(w_i)$, $\text{PE}(w_j)$)!=0, orice i,j

Pozitia normalizată în propozitie

❑ Exemplu

- O propoziție cu 5 cuvinte și $d = 1$
- $\text{PE(cuv)} = \text{poziția cuvântului}$

0/5
1/5
2/5
3/5
4/5

- Problema: al k -lea cuvânt într-o propoziție de N cuvinte are alt embedding decât al k -lea cuvânt într-o propoziție cu M cuvinte

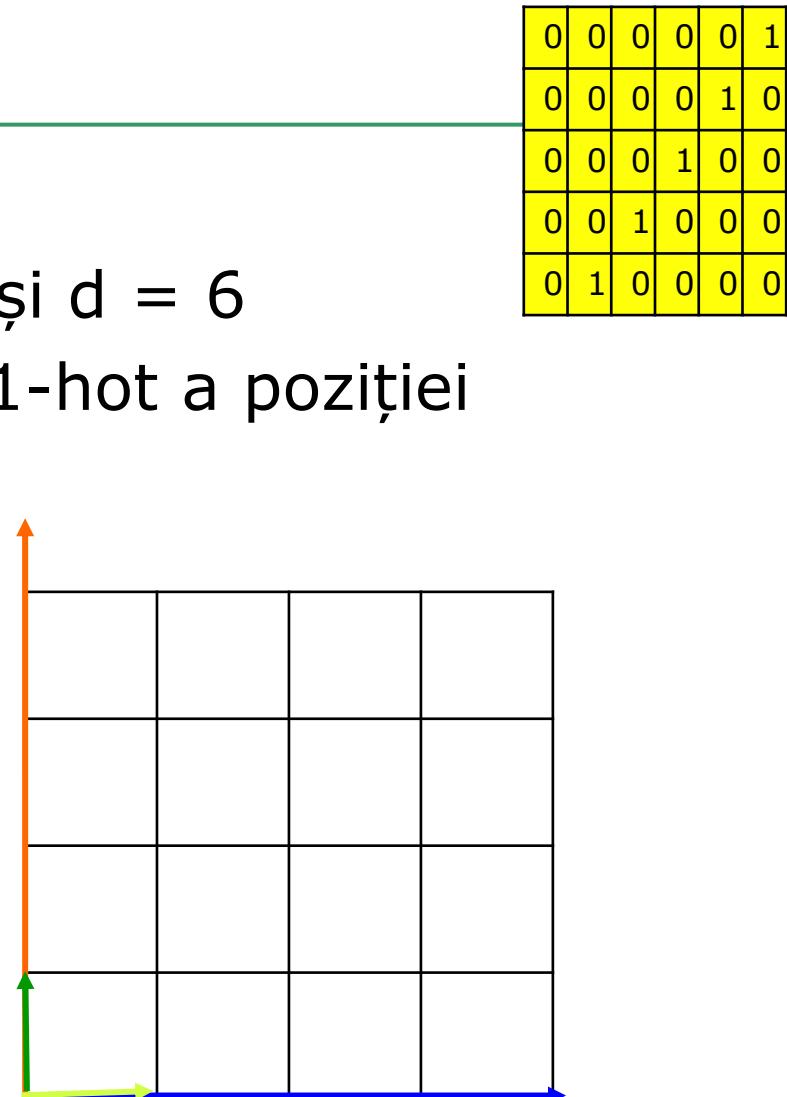
1-hot encoding

❑ Exemplu

- O propoziție cu 5 cuvinte și $d = 6$
- $\text{PE}(\text{cuv}) = \text{reprezentarea 1-hot a poziției cuvântului}$

Dist	w0	w1	w2	w3	w4
w0	0	1.4	1.4	1.4	1.4
w1	1.4	0	1.4	1.4	1.4
w2	1.4	1.4	0	1.4	1.4
w3	1.4	1.4	1.4	0	1.4
w4	1.4	1.4	1.4	1.4	0

$\text{sim}(\text{PE}(w_i), \text{PE}(w_j))=0$, orice i,j



Binary encoding

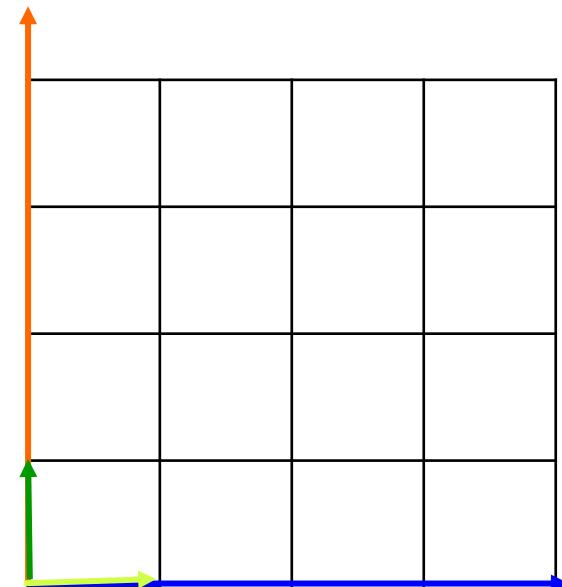
Exemplu

- O propoziție cu 5 cuvinte și $d = 6$
- $\text{PE}(\text{cuv}) = \text{reprezentarea binară a poziției cuvântului}$

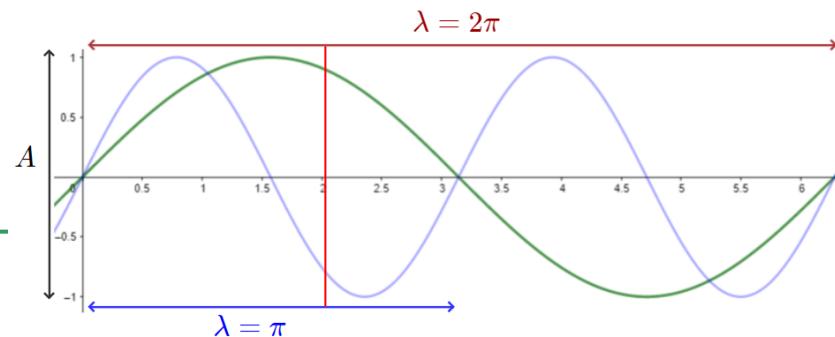
Dist	w0	w1	w2	w3	w4
w0	0	1	1	$\sqrt{2}$	1
w1	1	0	$\sqrt{2}$	1	$\sqrt{2}$
w2	1	$\sqrt{2}$	0	1	$\sqrt{2}$
w3	$\sqrt{2}$	1	1	0	$\sqrt{3}$
w4	1	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{3}$	0

există i, j a.î. $\text{sim}(\text{PE}(w_i), \text{PE}(w_j))=0$

0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	0	1	1
0	0	0	1	0	0



Sin-based encoding

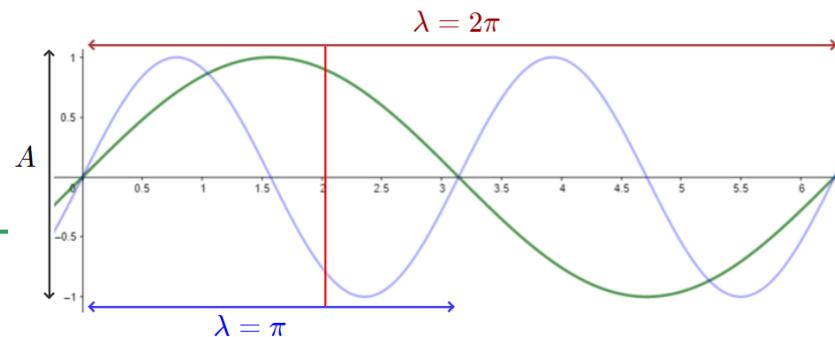


Exemplu

- O propoziție cu 5 cuvinte și $d = 6$
- PE(cuv) = valorile functiei sin pentru diferite argumente (frecvențe sau lungimi de undă)
 - E.g. $\sin(2\pi \text{ pos} / \lambda_i)$, $i = 0, 1, 2, \dots, d-1$

pos	$\lambda = \pi$	$\lambda = 2\pi$	$\lambda = 3\pi$	$\lambda = 4\pi$	$\lambda = 5\pi$	$\lambda = 6\pi$
0 →	$\sin(2*0)$	$\sin(0)$	$\sin(2/3*0)$	$\sin(2/4*0)$	$\sin(2/5*0)$	$\sin(2/6*0)$
1 →	$\sin(2*1)$	$\sin(1)$	$\sin(2/3*1)$	$\sin(2/4*1)$	$\sin(2/5*1)$	$\sin(2/6*1)$
2 →	$\sin(2*2)$	$\sin(2)$	$\sin(2/3*2)$	$\sin(2/4*2)$	$\sin(2/5*2)$	$\sin(2/6*2)$
3 →	$\sin(2*3)$	$\sin(3)$	$\sin(2/3*3)$	$\sin(2/4*3)$	$\sin(2/5*3)$	$\sin(2/6*3)$
4 →	$\sin(2*4)$	$\sin(4)$	$\sin(2/3*4)$	$\sin(2/4*4)$	$\sin(2/5*4)$	$\sin(2/6*4)$

Sin-based encoding



Exemplu

- O propoziție cu 5 cuvinte și $d = 6$
- PE(cuv) = valorile functiei sin pentru diferite argumente (frecvențe sau lungimi de undă)
 - E.g. $\sin(2\pi \text{ pos} / \lambda_i)$, $i = 0, 1, 2, \dots, d-1$

pos	$\lambda = \pi$	$\lambda = 2\pi$	$\lambda = 3\pi$	$\lambda = 4\pi$	$\lambda = 5\pi$	$\lambda = 6\pi$
0 →	$\sin(0)$	$\sin(0)$	$\sin(0)$	$\sin(0)$	$\sin(0)$	$\sin(0)$
1 →	$\sin(2*1)$	$\sin(1)$	$\sin(2/3*1)$	$\sin(2/4*1)$	$\sin(2/5*1)$	$\sin(2/6*1)$
2 →	$\sin(2*2)$	$\sin(2)$	$\sin(2/3*2)$	$\sin(2/4*2)$	$\sin(2/5*2)$	$\sin(2/6*2)$
3 →	$\sin(2*3)$	$\sin(3)$	$\sin(2/3*3)$	$\sin(2/4*3)$	$\sin(2/5*3)$	$\sin(2/6*3)$
4 →	$\sin(2*4)$	$\sin(4)$	$\sin(2/3*4)$	$\sin(2/4*4)$	$\sin(2/5*4)$	$\sin(2/6*4)$

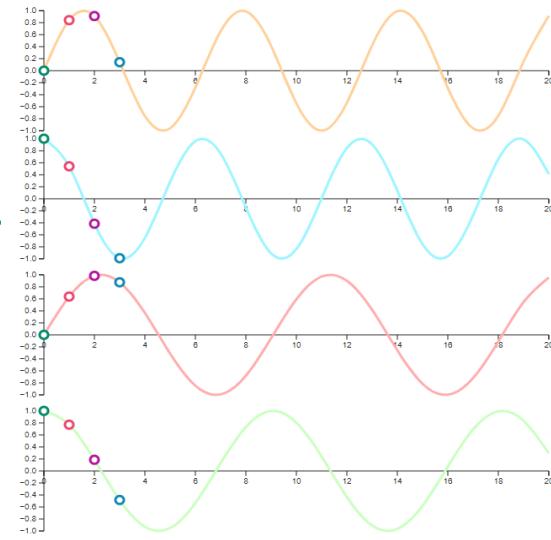
$$\text{sim}(\text{PE}(\text{word}_0), \text{PE}(\text{word}_k)) = 0!!!$$

Sin&cos-based encoding

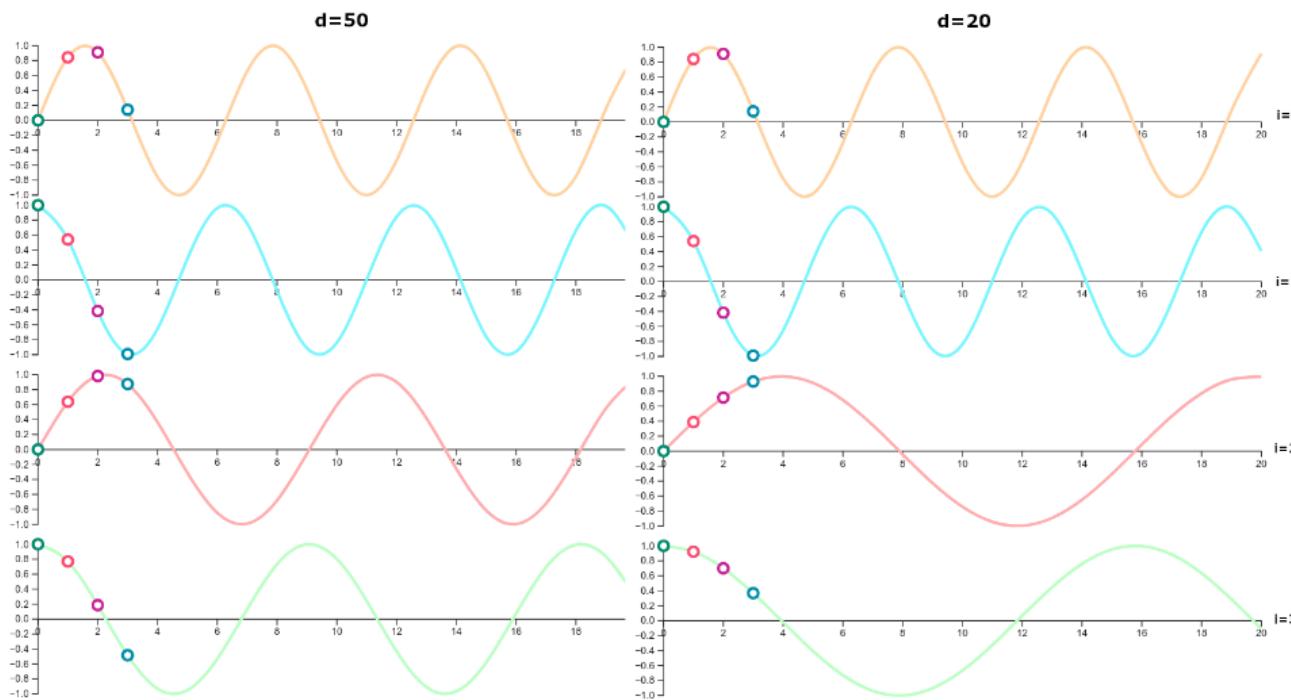
❑ Exemplu

- O propoziție cu 5 cuvinte și $d = 6$
- PE(cuv) = perechi (sin, cos) pentru diferite argumente (frecvențe sau lungimi de undă)
 - ❑ E.g. $(\sin(\text{pos} / 10000^{2i/d}), \cos(\text{pos} / 10000^{2i/d})), i = 0, 1, 2, \dots, d-1$

pos	i=0	i=1	i=2	i=3	i=4	i=5
0 →	$\sin(0)$	$\cos(0)$	$\sin(0)$	$\cos(0)$	$\sin(0)$	$\cos(0)$
1 →	$\sin(1)$	$\cos(1)$	$\sin(1/10000^{4/6})$	$\cos(1/10000^{4/6})$	$\sin(1/10000^{8/6})$	$\cos(1/10000^{8/6})$
2 →	$\sin(2)$	$\cos(2)$	$\sin(2/10000^{4/6})$	$\cos(2/10000^{4/6})$	$\sin(2/10000^{8/6})$	$\cos(2/10000^{8/6})$
3 →	$\sin(3)$	$\cos(3)$	$\sin(3/10000^{4/6})$	$\cos(3/10000^{4/6})$	$\sin(3/10000^{8/6})$	$\cos(3/10000^{8/6})$
4 →	$\sin(4)$	$\cos(4)$	$\sin(4/10000^{4/6})$	$\cos(4/10000^{4/6})$	$\sin(4/10000^{8/6})$	$\cos(4/10000^{8/6})$



Sin&cos-based encoding



-
- ❑ [https://arxiv.org/pdf/1706.03762](https://arxiv.org/pdf/1706.03762.pdf)
 - ❑ <https://github.com/karpathy/nanoGPT>
 - ❑ <https://github.com/karpathy/ng-video-lecture>
 - ❑ <https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3cfvtXnx-?usp=sharing>
 - ❑ <https://paperswithcode.com/paper/attention-is-all-you-need>
 - ❑ <https://www.youtube.com/watch?v=kCc8FmEb1nY>

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

■ **Sisteme care învață singure**

- Arbori de decizie
- Rețele neuronale artificiale - transformers
- Mașini cu suport vectorial
- Algoritmi evolutivi

- Sisteme bazate pe reguli
- Sisteme hibride

C. Rezolvarea problemelor prin căutare

■ Definirea problemelor de căutare

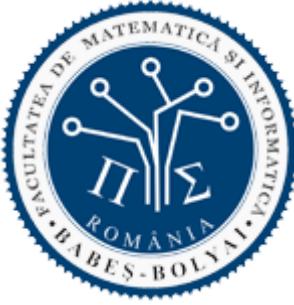
■ Strategii de căutare

- Strategii de căutare neinformate
- Strategii de căutare informate
- Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
- Strategii de căutare adversarială

-
- ❑ Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop
 - Prof. Dr. Radu Ionescu - <https://raduionescu.herokuapp.com/>



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare
Strategii de căutare neinformată

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

c. Sisteme inteligente

- Sisteme care învață singure

- Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi

- Sisteme bazate pe reguli

- Sisteme hibride

B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare

- Strategii de căutare

- Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

Sumar

- Probleme
- Rezolvarea problemelor
 - Pași în rezolvarea problemelor
- Rezolvarea problemelor prin căutare
 - Pași în rezolvarea problemelor prin căutare
 - Tipuri de strategii de căutare

Materiale de citit și legături utile

- capitolele I.1, I.2, II.3 și II.4 din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*

- capitolele 1 - 4 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*

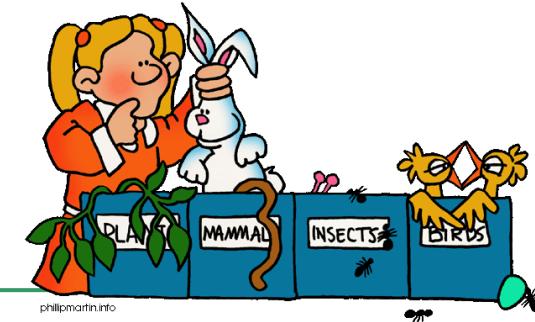
- capitolele 2.1 – 2.5 din <http://www-g.eng.cam.ac.uk/mmg/teaching/artificialintelligence/>

Probleme



- Două mari categorii de probleme:
 - Rezolvabile în mod determinist
 - Calculul sinusului unui unghi sau a rădăcinii pătrate dintr-un număr
 - Rezolvabile în mod stocastic
 - Probleme din lumea reală → proiectarea unui ABS
 - Presupun căutarea unei soluții → metode ale IA

Probleme



□ Tipologie

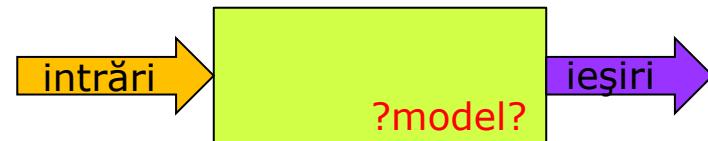
■ Probleme de căutare/optimizare

- Planificare, proiectarea sateliților



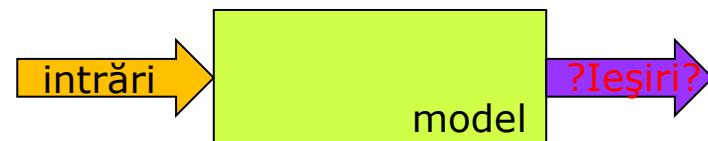
■ Probleme de modelare

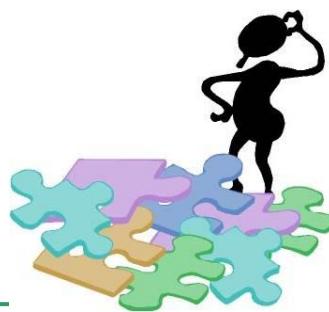
- Predicții, clasificări



■ Probleme de simulare

- Teoria jocurilor economice

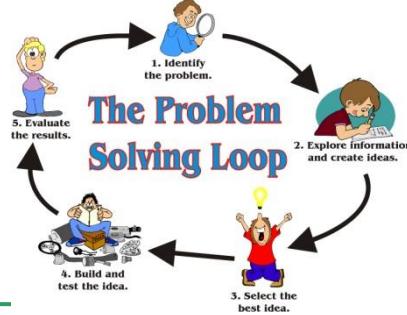




Rezolvarea problemelor

- Constă în identificarea unei soluții
 - În informatică (IA) → proces de căutare
 - În inginerie și matematică → proces de optimizare

- Cum?
 - Reprezentarea soluțiilor (partiale) → puncte în spațiul de căutare
 - Proiectarea unor operatori de căutare → transformă o posibilă soluție în altă soluție



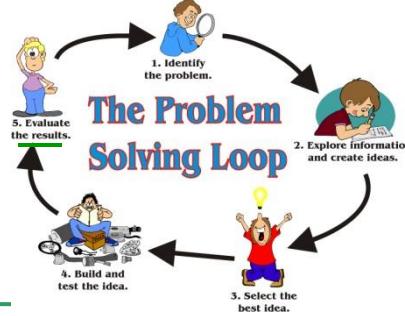
Pasi în rezolvarea problemelor

- Definirea problemei
- Analiza problemei
- Alegerea unei tehnici de rezolvare
 - căutare
 - reprezentarea cunoștințelor
 - abstractizare



Rezolvarea problemelor prin căutare

- bazată pe urmărirea unor obiective
- compusă din acțiuni care duc la îndeplinirea unor obiective
 - fiecare acțiune modifică o anumită stare a problemei
- succesiune de acțiuni care transformă starea inițială a problemei în stare finală

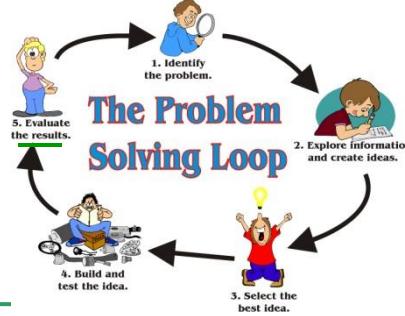


Pași în rezolvarea problemelor prin căutare

Definirea problemei

Definirea problemei implică stabilirea:

- unui spațiu de stări
 - toate configurațiile posibile, fără enumerarea obligatorie a tuturor configurațiilor
 - reprezentare
 - explicită – construirea (în memorie) a tuturor stărilor posibile
 - implicită – utilizarea unor structuri de date și a unor funcții (operatori)
- unei/unor stări inițiale
- unei/unor stări finale - obiectiv
- unui/unor drum(uri)
 - succesiuni de stări
- unui set de reguli (acțiuni)
 - funcții succesor (operatori) care precizează starea următoare a unei stări
 - funcții de cost care evaluatează
 - trecerea dintr-o stare în alta
 - un întreg drum
 - funcții obiectiv care verifică dacă s-a ajuns într-o stare finală



Pași în rezolvarea problemelor prin căutare

Definirea problemei

□ Exemple

■ Joc puzzle cu 8 piese

- spațiul stărilor – configurații ale tablei de joc cu 8 piese

- starea inițială – o configurație oarecare

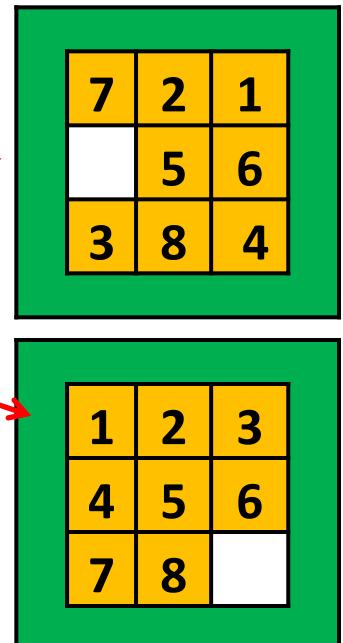
- starea finală – o configurație cu piesele aranjate într-o anumită ordine

- Reguli → acțiuni albe

- Condiții: mutarea în interiorul tablei

- Transformări: spațiul alb se mișcă în sus, în jos, la stânga sau la dreapta

- soluția – secvența optimă de acțiuni albe



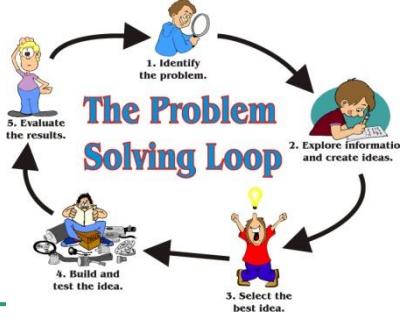
Pași în rezolvarea problemelor prin căutare

Definirea problemei

□ Exemple

■ Joc cu n dame

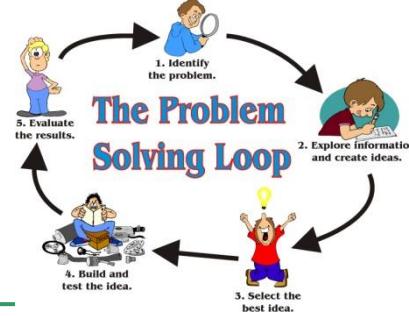
- Spațiul stărilor – configurații ale tablei de joc cu n regine
 - Starea inițială – o configurație fără regine
 - Starea finală – o configurație cu n regine care nu se atacă
 - Reguli → amplasarea unei regine pe tablă
 - Condiții: regina amplasată nu este atacată de nici o regină existentă pe tablă
 - Transformări: amplasarea unei noi regine într-o căsuță de pe tabla de joc
 - Solutia – amplasarea optimă a reginelor pe tablă



	a	b	c	d	e	f	g	h	
1	■			■	■	■	■		1
2				■	■	■	■	■	2
3	■				■				3
4		■				■			4
5			■						5
6				■			■		6
7	■				■				7
8		■				■			8
	a	b	c	d	e	f	g	h	

Pași în rezolvarea problemelor prin căutare

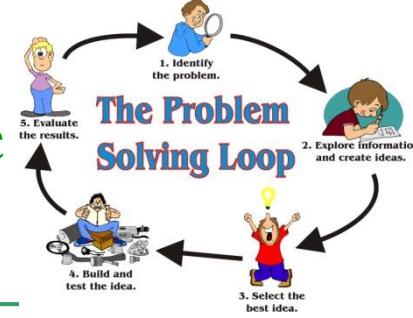
Analiza problemei



- Se poate descompune problema?
 - Sub-problemele sunt independente sau nu?
- Universul stărilor posibile este predictibil?
- Se dorește obținerea oricărei soluții sau a unei soluții optime?
- Soluția dorită constă într-o singură stare sau într-o succesiune de stări?
- Sunt necesare multiple cunoștințe pentru a limita căutarea sau chiar pentru a identifica soluția?
- Problema este conversațională sau solitară?
 - Este sau nu nevoie de interacțiune umană pentru rezolvarea ei?

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



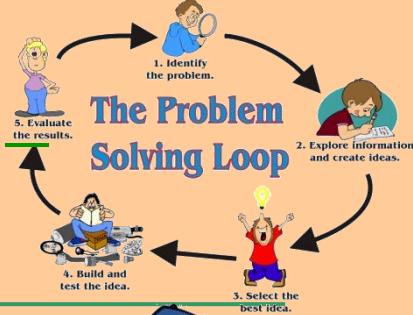
- Rezolvarea prin utilizarea regulilor (în combinație cu o strategie de control) de deplasare în spațiul problemei până la găsirea unui drum între starea inițială și cea finală

- Rezolvare prin căutare
 - Examinarea sistematică a stărilor posibile în vederea identificării
 - unui drum de la o stare inițială la o stare finală
 - unei stări optime
 - Spațiul stărilor = toate stările posibile + operatorii care definesc legăturile între stări



Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

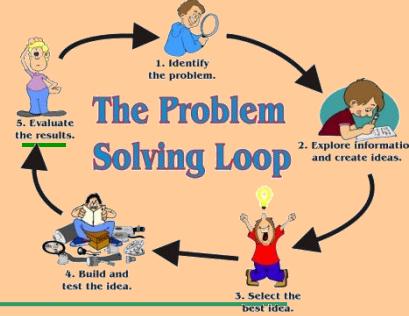


□ Rezolvare prin căutare

- Strategii de căutare multiple → cum alegem o strategie?
 - Complexitatea computațională (temporală și spațială)
 - Completitudine → algoritmul se sfârșește întotdeauna și găsește o soluție (dacă ea există)
 - Optimalitate → algoritmul găsește soluția optimă (costul optim al drumului de la starea inițială la starea finală)

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



□ Rezolvare prin căutare

- Strategii de căutare multiple → cum alegem o strategie?

- Complexitatea computațională (temporală și spațială)

- Performanța strategiei depinde de:

- Timpul necesar rulării
 - Spațiul (memoria) necesară rulării
 - Mărimea intrărilor algoritmului
 - Viteza calculatorului
 - Calitatea compilatorului

} Factori interni

} Factori externi

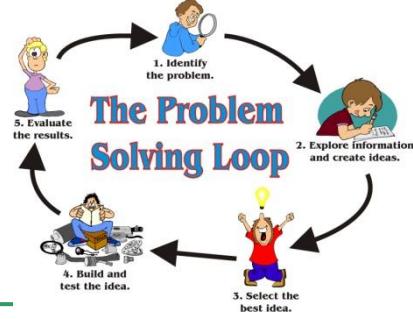
- Se măsoară cu ajutorul complexității → Eficiență computațională

- Spațială → memoria necesară identificării soluției
 - $S(n)$ – cantitatea de memorie utilizată de cel mai bun algoritm A care rezolvă o problemă de decizie f cu n date de intrare
 - Temporală → timpul necesar identificării soluției
 - $T(n)$ – timpul de rulare (numărul de pași) al celui mai bun algoritm A care rezolvă o problemă de decizie f cu n date de intrare



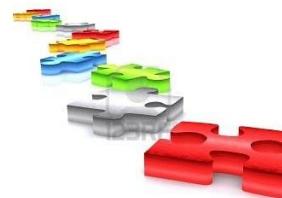
Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



- Rezolvarea problemelor prin căutare poate consta în:

- Construirea progresivă a soluției



- Identificarea soluției potențiale optime

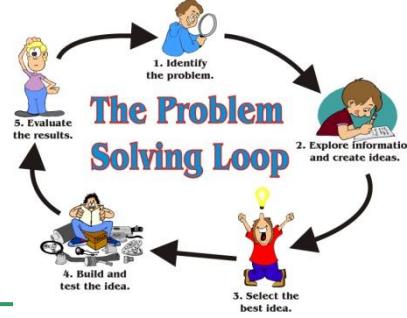


Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

□ Rezolvarea problemelor prin căutare poate consta în:

- Construirea progresivă a soluției
 - Componentele problemei
 - Stare inițială
 - Operatori (funcții succesor)
 - Stare finală
 - Soluția = un drum (de cost optim) de la starea inițială la starea finală
 - Spațiul de căutare
 - Mulțimea tuturor stărilor în care se poate ajunge din starea inițială prin aplicarea operatorilor
 - stare = o componentă a soluției
 - Exemple
 - Problema comisului voiajor
 - Algoritmi
 - Ideea de bază: se începe cu o componentă a soluției și se adaugă noi componente până se ajunge la o soluție completă
 - Recursivi → se re-aplică până la îndeplinirea unei condiții
 - Istoricul căutării (drumul parcurs de la starea inițială la starea finală) este reținut în liste de tip LIFO sau FIFO
 - Avantaje
 - Nu necesită cunoștințe (informații inteligente)

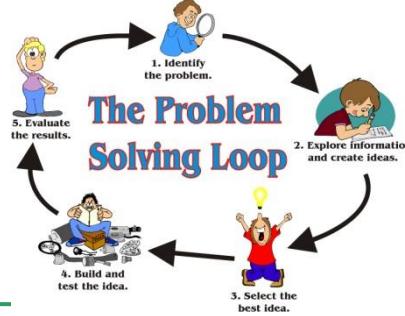


Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

□ Rezolvarea problemelor prin căutare poate consta în:

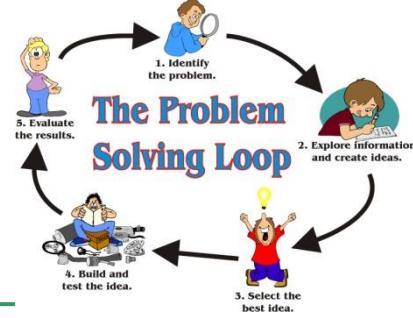
- Identificarea soluției potențiale optime
 - Componentele problemei
 - Condiții (constrângeri) pe care trebuie să le satisfacă (parțial sau total) soluția
 - Funcție de evaluare a unei soluții potențiale → identificarea optimului
 - Spațiul de căutare
 - mulțimea tuturor soluțiilor potențiale complete
 - Stare = o soluție completă
 - Exemple
 - Problema celor 8 regine
 - Algoritmi
 - Ideea de bază: se începe cu o stare care nu respectă anumite constrângeri pentru a fi soluție optimă și se efectuează modificări pentru a elimina aceste violări
 - Iterativi → se memorează o singură stare și se încercă îmbunătățirea ei
 - Istoricul căutării nu este reținut
 - Avantaje
 - Simplu de implementat
 - Necesară puțină memorie
 - Poate găsi soluții rezonabile în spații de căutare (continue) foarte mari pentru care alți algoritmi sistematici nu pot fi aplicati



www.shutterstock.com - 36774760

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



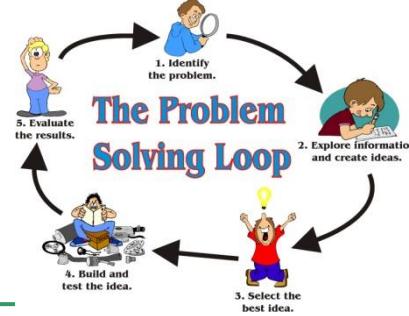
□ Rezolvarea problemelor prin căutare presupune

- algoritmi cu o complexitate ridicată (probleme NP-complete)
- căutarea într-un spațiu exponențial



Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

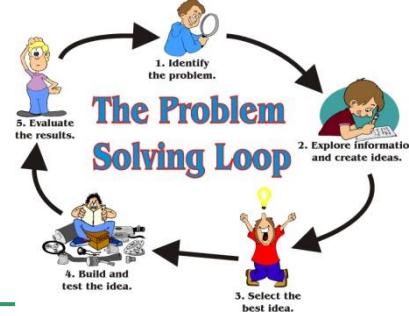


□ Tipologia strategiilor de căutare

- În funcție de modul de **generare** a soluției
 - Căutare **constructivă**
 - Construirea progresivă a soluției
 - Ex. TSP
 - Căutare **perturbativă**
 - O soluție candidat este modificată în vederea obținerii unei noi soluții candidat
 - Ex. SAT - Propositional Satisfiability Problem
- În funcție de modul de **traversare** a spațiului de căutare
 - Căutare **sistematică**
 - Traversarea completă a spațiului
 - Ideintificarea soluției dacă ea există → algoritmi compleți
 - Căutare **locală**
 - Traversarea spațiului de căutare dintr-un punct în alt punct vecin → algoritmi incompleți
 - O stare a spațiului poate fi vizitată de mai multe ori
- În funcție de elementele de **certitudine** ale căutării
 - Căutare **deterministă**
 - Algoritmi de identificare exactă a soluției
 - Căutare **stocastică**
 - Algoritmi de aproximare a soluției
- În funcție de stilul de **explorare** a spațiului de căutare
 - Căutare **secvențială**
 - Căutare **paralelă**

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

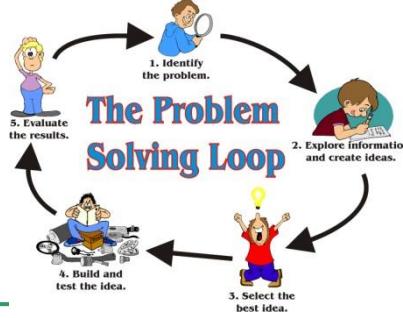


Tipologia strategiilor de căutare

- În funcție de **scopul** urmărit
 - Căutare **uni-obiectiv**
 - Soluția trebuie să satisfacă o singură condiție/constrângere
 - Căutare **multi-obiectiv**
 - Soluția trebuie să satisfacă mai multe condiții (obiective)
- În funcție de **numărul de soluții optime**
 - Căutare **uni-modală**
 - Există o singură soluție optimă
 - Căutare **multi-modală**
 - Există mai multe soluții optime
- În funcție de tipul de **algoritm** folosit
 - Căutare de-a lungul unui **număr finit de pași**
 - Căutare **iterativă**
 - Algoritmii converg către soluție
 - Căutare **euristică**
 - Algoritmii oferă o aproximare a soluției
- În funcție de **mecanismul** căutării
 - Căutare **tradițională**
 - Căutare **modernă**
- În funcție de **locul** în care se desfășoară căutarea în spațiul de căutare
 - Căutare **locală**
 - Căutare **globală**

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



Tipologia strategiilor de căutare

□ În funcție de tipul **(linearitatea) constrângerilor**

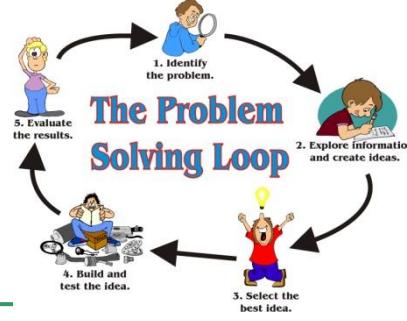
- Căutare **liniară**
- Căutare **neliniară**
 - Clasică (deterministă)
 - Directă – bazată doar pe evaluarea funcției obiectiv
 - Indirectă – bazată și pe derivata (I si/sau II) a funcției obiectiv
 - Enumerativă
 - În funcție de modul de stabilire a soluției
 - Ne-informată → soluția se știe doar când ea coincide cu starea finală
 - Informată → se lucrează cu o funcție de evaluare a unei soluții parțiale
 - În funcție de tipul spațiului de căutare
 - Completă – spațiu finit (dacă soluția există, ea poate fi găsită)
 - Incompletă – spațiu infinit
 - Stocastică
 - Bazată pe elemente aleatoare

□ În funcție de **agenții** implicați în căutare

- Căutare realizată de un **singur agent** → fără piedici în atingerea obiectivelor
- Căutare **adversarială** → adversarul aduce o incertitudine în realizarea obiectivelor

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare

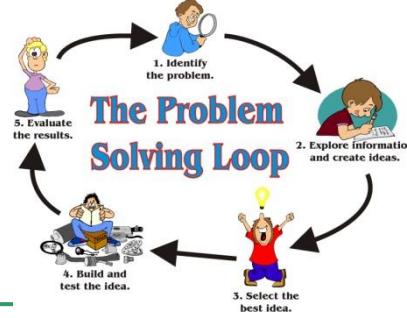


Exemplu

- Tipologia strategiilor de căutare
 - În funcție de modul de generare a soluției
 - **Căutare constructivă**
 - Căutare perturbativă
 - În funcție de modul de traversare a spațiului de căutare
 - **Căutare sistematică**
 - Căutare locală
 - În funcție de elementele de certitudine ale căutării
 - **Căutare deterministă**
 - Căutare stocastică
 - În funcție de stilul de explorare a spațiului de căutare
 - **Căutare secvențială**
 - Căutare paralelă

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



Exemplu

□ Problema “capra, varza și lupul”

■ Se dau:

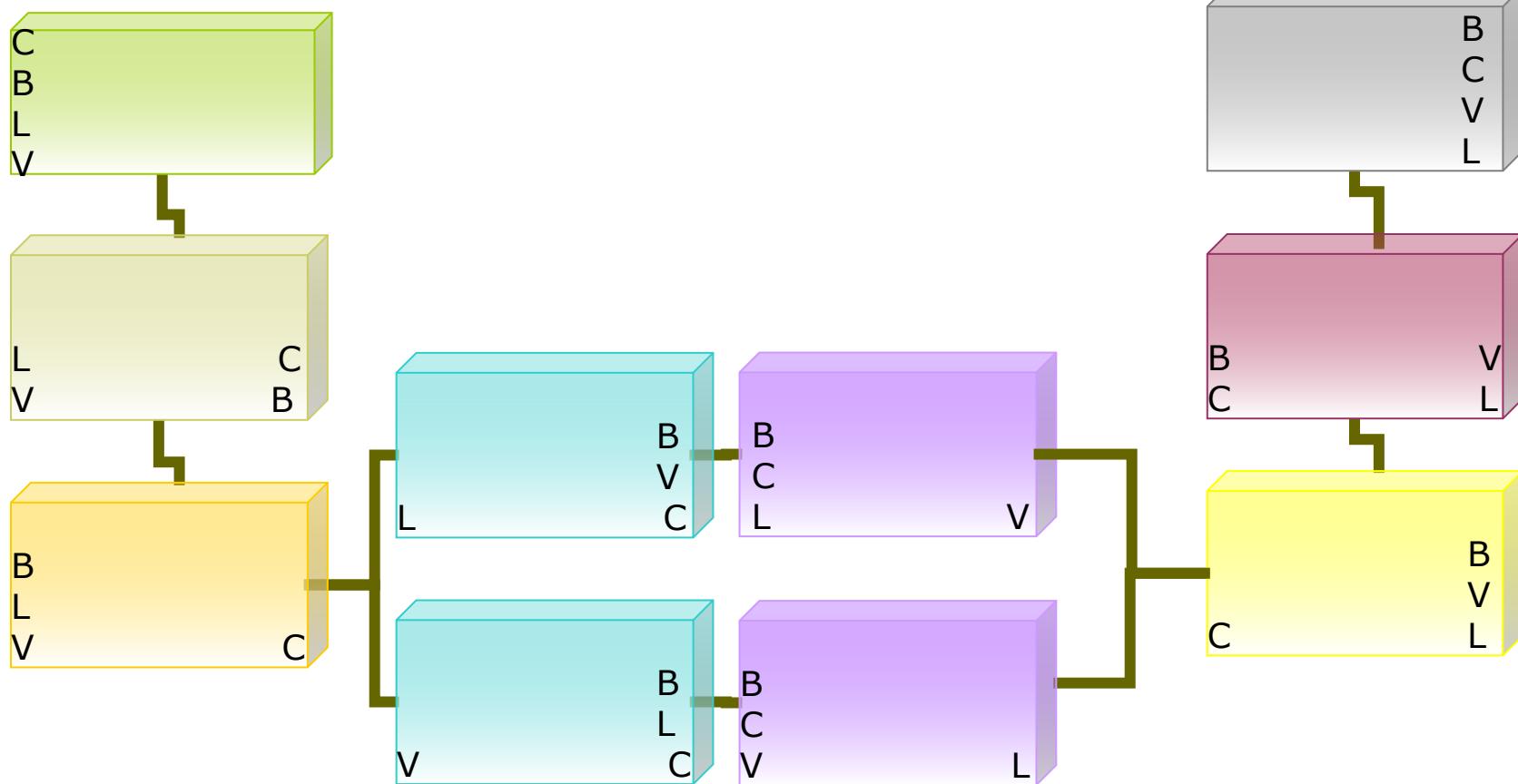
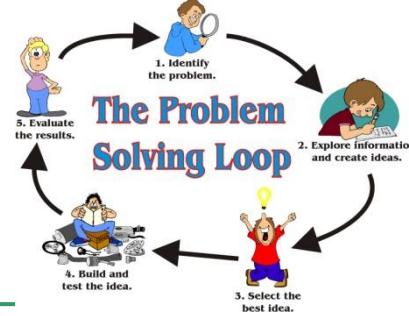
- o capră, o varză și un lup pe malul unui râu
- o barcă cu barcagiu

■ Se cere

- Să se traverseze toți pasagerii pe malul celălalt al râului
- cu următoarele condiții
 - în barcă există doar 2 locuri
 - nu pot rămâne pe același mal
 - capra și varza
 - lupul și capra

Pași în rezolvarea problemelor prin căutare

Alegerea unei tehnici de rezolvare



Strategii de căutare – Elemente fundamentale



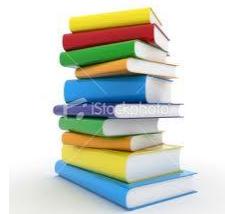
- Tipuri abstracte de date (TAD)
 - TAD listă → structură liniară
 - TAD arbore → structură arborescentă (ierarhică)
 - TAD graf → structură de graf

- TAD
 - Domeniu și operații
 - Reprezentare

Strategii de căutare – Elemente fundamentale – TAD Listă



- Domeniu
 - $D = \{l \mid l = (el_1, el_2, \dots), \text{ unde } el_i, i=1,2,3\dots, \text{ sunt de același tip } TE \text{ (tip element) și fiecare element } el_i, i=1,2,3\dots, \text{ are o poziție unică în } l \text{ de tip } TP \text{ (TipPoziție)}\}$
- Operații
 - Creare(l)
 - Prim(l)
 - Ultim(l)
 - Următor(l,p)
 - Anterior(l,p)
 - Valid(l,p)
 - getElement(l,p)
 - getPoziție(l,e)
 - Modifică(l,p,e)
 - AdăugareLaÎnceput(l,e)
 - AdăugareLaSfârșit(l,e)
 - AdăugareDupă(l,p,e)
 - AdăugareÎnainte(l,p,e)
 - Eliminare(l,p)
 - Căutare(l,e)
 - Vidă(l)
 - Dimensiune(l)
 - Distrugere(l)
 - getIterator(l)
- Reprezentare
 - Vectorială
 - Liste (simplu sau dublu) înlănțuite, etc
- Cazuri particulare
 - Stivă – LIFO
 - Coadă – FIFO
 - Coadă cu priorități



Strategii de căutare – Elemente fundamentale – TAD Listă



- Domeniu
 - $D = \{l \mid l = (el_1, el_2, \dots), \text{ unde } el_i, i=1,2,3\dots, \text{ sunt de același tip } TE \text{ (tip element) și fiecare element } el_i, i=1,2,3\dots, \text{ are o poziție unică în } l \text{ de tip } TP \text{ (TipPoziție)}\}$
- Operații
 - Creare(l)
 - Prim(l)
 - Ultim(l)
 - Următor(l,p)
 - Anterior(l,p)
 - Valid(l,p)
 - getElement(l,p)
 - getPoziție(l,e)
 - Modifică(l,p,e)
 - AdăugareLaÎnceput(l,e)
 - AdăugareLaSfârșit(l,e)
 - AdăugareDupă(l,p,e)
 - AdăugareÎnainte(l,p,e)
 - Eliminare(l,p)
 - Căutare(l,e)
 - Vidă(l)
 - Dimensiune(l)
 - Distrugere(l)
 - getIterator(l)
- Reprezentare
 - Vectorială
 - Liste (simplu sau dublu) înlănțuite, etc
- Cazuri particulare
 - Stivă – LIFO
 - Coadă – FIFO
 - Coadă cu priorități



Strategii de căutare – Elemente fundamentale – TAD Listă



- Domeniu
 - $D = \{l \mid l = (el_1, el_2, \dots), \text{ unde } el_i, i=1,2,3\dots, \text{ sunt de același tip } TE \text{ (tip element) și fiecare element } el_i, i=1,2,3\dots, \text{ are o poziție unică în } l \text{ de tip } TP \text{ (TipPoziție)}\}$
- Operații
 - Creare(l)
 - Prim(l)
 - Ultim(l)
 - Următor(l,p)
 - Anterior(l,p)
 - Valid(l,p)
 - getElement(l,p)
 - getPoziție(l,e)
 - Modifică(l,p,e)
 - AdăugareLaÎnceput(l,e)
 - AdăugareLaSfârșit(l,e)
 - AdăugareDupă(l,p,e)
 - AdăugareÎnainte(l,p,e)
 - Eliminare(l,p)
 - Căutare(l,e)
 - Vidă(l)
 - Dimensiune(l)
 - Distrugere(l)
 - getIterator(l)
- Reprezentare
 - Vectorială
 - Liste (simplu sau dublu) înlántuite, etc
- Cazuri particulare
 - Stivă – LIFO
 - Coadă – FIFO
 - Coadă cu priorități



Strategii de căutare – Elemente fundamentale – TAD Graf



- Domeniu – container de noduri și legături între noduri
 - $D = \{nod_1, nod_2, \dots, nod_n, leg_1, leg_2, \dots, leg_m\}$, unde nod_i , cu $i=1,2,\dots,n$ sunt noduri, iar leg_i , cu $i=1,2,\dots,m$ sunt muchii între noduri}
- Operații
 - creare
 - creareNod
 - traversare
 - getIterator
 - distrugere
- Reprezentare
 - Lista muchilor
 - Lista de adiacență (Tradițională și Modernă)
 - Matricea de adiacență (Tradițională și Modernă)
 - Matricea de incidentă
- Cazuri particulare
 - Grafuri orientate și neorientate
 - Grafuri simple sau multiple
 - Grafuri conexe sau nu
 - Grafuri complete sau nu
 - Grafuri cu sau fără cicluri (aciclice \rightarrow păduri, arbori)

Strategii de căutare – Elemente fundamentale – TAD Arbore



- Domeniu – container de noduri si legături între noduri
 - $D = \{nod_1, nod_2, \dots, nod_n, leg_1, leg_2, \dots, leg_m\}$, unde nod_i , cu $i=1,2,\dots,n$ sunt noduri, iar leg_i , cu $i=1,2,\dots,m$ sunt muchii între noduri astfel încât să nu se formeze cicluri}
- Operații
 - creare
 - creareFrunză
 - adăugareSubarbore
 - getInfoRădăcină
 - getSubarbore
 - traversare
 - getIterator
 - distrugere
- Reprezentare
 - Vectorială
 - Liste înlăntuite ale descendentilor
- Cazuri particulare
 - Arbo里 binari (de căutare)
 - Arbo里 n-ari

Strategii de căutare – Elemente fundamentale – parcurgerea grafelor



□ Drumuri

- drum (*path*)
 - nodurile nu se pot repeta
- *trail*
 - muchiile nu se pot repeta
- *walk*
 - fără restricții
- drum închis
 - nodul inițial = nodul final
- circuit
 - un *trail* închis
- ciclu
 - un *path* închis



Strategii de căutare neinformate (SCnI)

□ Caracteristici

- nu se bazează pe informații specifice problemei
- sunt generale
- strategii oarbe
- strategii de tipul forței brute

□ Topologie

- În funcție de ordinea expandării stărilor în spațiul de căutare:
 - SCnI în structuri liniare
 - căutare liniară
 - căutare binară
 - SCnI în structuri ne-liniare
 - căutare în lățime (breadth-first)
 - căutare de cost uniform (branch and bound)
 - căutare în adâncime (depth-first)
 - căutare în adâncime limitată (limited depth-first)
 - căutare în adâncime iterativă (iterative deepening depth-first)
 - căutare bidirectională

SCnI în structuri liniare

Căutare liniară



□ Aspecte teoretice

- Se verifică fiecare element al unei liste până la identificarea celui dorit
- Lista de elemente poate fi ordonată sau nu

□ Exemplu

- Lista = (2, 3, 1, ,7, 5)
- Elemt = 7

□ Algoritm

```
bool LS(elem, list){  
    found = false;  
    i = 1;  
    while ((!found) && (i <= list.length)) {  
        if (elem == list[i])  
            found = true;  
        else  
            i++;  
    } //while  
    return found;  
}
```

SCnI în structuri liniare

Căutare liniară



□ Analiza căutării

- Complexitate temporală
 - Cel mai bun caz: $elem = list[1] \Rightarrow O(1)$
 - Cel mai slab caz: $elem \notin list \Rightarrow T(n) = n + 1 \Rightarrow O(n)$
 - Cazul mediu: $T(n) = (1 + 2 + \dots + n + (n+1))/(n+1) \Rightarrow O(n)$
- Complexitate spațială
 - $S(n) = n$
- Completitudine
 - da
- Optimalitate
 - da

□ Avantaje

- Simplitate, complexitate temporală bună pentru structuri mici
- Structura nu trebuie sortată în prealabil

□ Dezavantaje

- complexitate temporală foarte mare pentru structuri mari

□ Aplicații

- Căutări în baze de date reale

SCnI în structuri liniare

Căutare binară



□ Aspecte teoretice

- Localizarea unui element într-o listă ordonată
- Strategie de tipul *Divide et Conquer*

□ Exemplu

- List = (2, 3, 5, 6, 8, 9, 13, 16, 18), Elem = 6
 - List = (2, 3, 5, 6, 8, 9, 13, 16, 18)
 - List = (2, 3, 5, 6)
 - List = (5, 6)
 - List = (6)

□ Algoritm

```
bool BS(elem, list){  
    found = false;  
    left = 1;  
    right = list.length;  
    while((left < right) && (!found)){  
        middle = left + (right - left)/2;  
        if (element == list[middle])  
            found = true;  
        else  
            if (element < list[middle])  
                right = middle - 1;  
            else  
                left = middle + 1;  
    } //while  
    return found;  
}
```

SCnI în structuri liniare

Căutare binară



Analiza căutării

- Complexitate temporală $T(n) = 1$, pt $n = 1$ și $T(n) = T(n/2) + 1$, altfel
Pp. că $n = 2k \Rightarrow k = \log_2 n$
Pp. că $2k < n < 2k+1 \Rightarrow k < \log_2 n < k + 1$
$$\begin{aligned} T(n) &= T(n/2) + 1 \\ T(n/2) &= T(n/2^2) + 1 \\ &\dots \\ T(n/2^{k-1}) &= T(n/2^k) + 1 \end{aligned}$$

$$T(n) = k + 1 = \log_2 n + 1$$
- Complexitate spațială – $S(n) = n$
- Completitudine – da
- Optimalitate – da

Avantaje

- Complexitate temporală redusă față de căutarea liniară

Dezavantaje

- Lucrul cu vectori (trebuie accesate elemente indexate) sortați

Aplicații

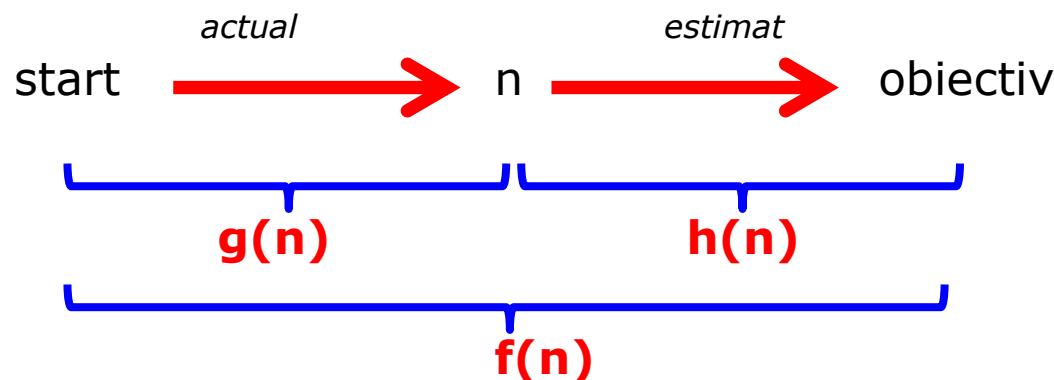
- Jocul ghicirii unui număr
- Căutare într-o carte de telefon/dicționar

SC în structuri arborescente

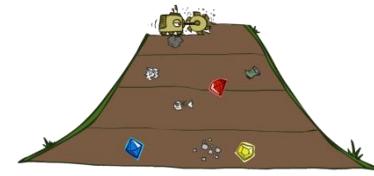


❑ Noțiuni necesare

- $f(n)$ – funcție de evaluare pentru estimarea costului soluției prin nodul (starea) n
- $h(n)$ – funcție heuristică pentru estimarea costului drumului de la nodul n la nodul obiectiv
- $g(n)$ – funcție de cost pentru estimarea costului drumului de la nodul de start până la nodul n
- $f(n) = g(n) + h(n)$



SCnI în structuri arborescente căutare în lățime (breadth-first search – BFS)



□ Aspecte teoretice

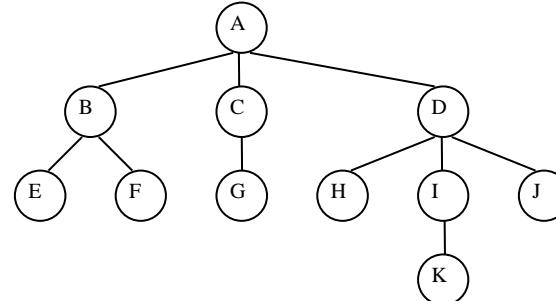
- Toate nodurile aflate la adâncimea d se expandează înaintea nodurilor aflate la adâncimea $d+1$
- Toate nodurile fii obținute prin expandarea nodului curent se adaugă într-o *listă* de tip *FIFO* (*coadă*)

□ Exemplu

- Ordinea vizitării: A, B, C, D, E, F, G, H, I, J, K

□ Algoritm

```
bool BFS(elem, list){  
    found = false;  
    visited = ∅;  
    toVisit = {start};           //FIFO list  
    while((toVisit != ∅) && (!found)){  
        node = pop(toVisit);  
        visited = visited ∪ {node};  
        if (node == elem)  
            found = true;  
        else{  
            aux = ∅;  
            for all (unvisited) children of node do  
                aux = aux ∪ {child};  
            }  
            toVisit = toVisit ∪ aux;  
    } //while  
    return found;  
}
```



Vizitate deja	De vizitat
∅	A
A	B, C, D
A, B	C, D, E, F
A, B, C	D, E, F, G
A, B, C, D	E, F, G, H, I, J
A, B, C, D, E	F, G, H, I, J
A, B, C, D, E, F	G, H, I, J
A, B, C, D, E, F, G	H, I, J
A, B, C, D, E, F, G, H	I, J
A, B, C, D, E, F, G, H, I	J, K
A, B, C, D, E, F, G, H, I, J	K
A, B, C, D, E, F, G, H, I, J, K	∅

SCnI în structuri arborescente căutare în lățime (breadth-first search – BFS)



Analiza căutării:

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d - lungimea (adâncimea) soluției
 - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Complexitate spațială
 - $S(n) = T(n)$
- Completitudine
 - Dacă soluția există, atunci BFS o găsește
- Optimalitate
 - nu

Avantaje

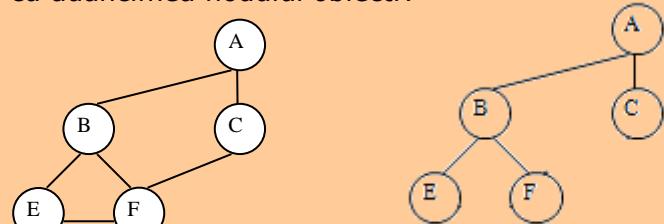
- Găsirea drumului de **lungime** minimă până la nodul obiectiv (soluția cea mai puțin adâncă)

Dezavantaje

- Generarea și stocarea unui arbore a cărui mărime crește exponențial cu adâncimea nodului obiectiv
- Complexitate temporală și spațială exponențială
- **Experimentul Russel&Norvig????**
- Funcțional doar pentru spații de căutare mici

Aplicații

- Identificarea tuturor componentelor conexe într-un graf
- Identificarea celui mai scurt drum într-un graf
- Optimizări în rețele de transport → algoritmul Ford-Fulkerson
- Serializarea/deserializarea unui arbore binar (vs. serializarea în mod sortat) permite reconstrucția eficientă a arborelui
- Copierea colecțiilor (garbage collection) → algoritmul Cheney





SCnI în structuri arborescente căutare de cost uniform (uniform cost search – UCS)

Aspecte teoretice

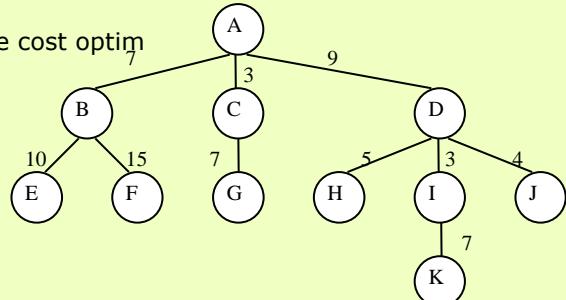
- BFS + procedură specială de expandare a nodurilor (bazată pe costurile asociate legăturilor dintre noduri)
- Toate nodurile de la adâncimea d sunt expandate înaintea nodurilor de la adâncimea $d+1$
- Toate nodurile fii obținute prin expandarea nodului curent se adaugă într-o *listă ORDONATĂ* de tip *FIFO*
 - Se expandează mai întâi nodurile de cost minim
 - Odată obținut un drum până la nodul ţintă, acesta devine candidat la drumul de cost optim
- Algoritmul *Branch and bound*

Exemplu

- Ordinea vizitării nodurilor: A, C, B, D, G, E, F, I, H, J, K

Algoritm

```
bool UCS(elem, list){
    found = false;
    visited = {};
    toVisit = {start}; //FIFO sorted list
    while((toVisit != {}) && (!found)) {
        node = pop(toVisit);
        visited = visited U {node};
        if (node== elem)
            found = true;
        else
            aux = {};
            for all (unvisited) children of node do{
                aux = aux U {child};
            } // for
            toVisit = toVisit U aux;
            TotalCostSort(toVisit);
    } //while
    return found;
}
```



visited	toVisit
{}	A
A	C(3), B(7), D(9)
A, C	B(7), D(9), G(3+7)
A, C, B	D(9), G(10), E(7+10), F(7+15)
A, C, B, D	G(10), I(9+3), J(9+4), H(9+5), E(17), F(22)
A, C, B, D, G	I(12), J(13), H(14), E(17), F(22)
A, C, B, D, G, I	J(13), H(14), E(17), F(22), K(9+3+7)
A, C, B, D, G, I, J	H(14), E(17), F(22), K(19)
A, C, B, D, G, I, J, H	E(17), F(22), K(19)
A, C, B, D, G, I, J, H, E	F(22), K(19)
A, C, B, D, G, I, J, H, E, F	K(19)
A, C, B, D, G, I, J, H, E, F, K	{}



SCnI în structuri arborescente căutare de cost uniform (uniform cost search – UCS)

□ Analiza complexității

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d - lungimea (adâncimea) soluției
 - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Complexitate spațială
 - $S(n) = T(n)$
- Completitudine
 - Da – dacă soluția există, atunci UCS o găsește
- Optimalitate
 - Da

□ Avantaje

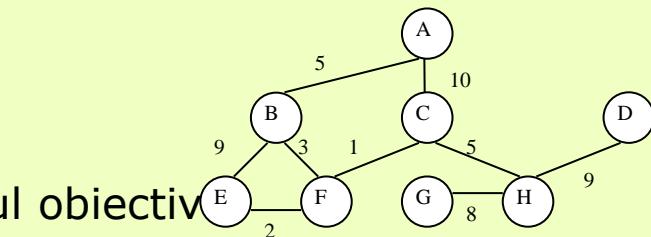
- Găsirea drumului de **cost** minim până la nodul obiectiv

□ Dezavantaje

- Complexitate temporală și spațială exponentială

□ Aplicații

- Cel mai scurt drum → algoritmul Dijkstra



Vizitate deja	De vizitat
Φ	A(0)
A(0)	B(5), C(10)
A(0), B(5)	F(8), C(10), E(14)
A(0), B(5), F(8)	C(9), E(10)
A(0), B(5), F(8), C(9)	E(10), H(14)
A(0), B(5), F(8), C(9), E(10)	H(14)

SCnI în structuri arborescente căutare în adâncime (depth-first search – DFS)



Aspecte teoretice

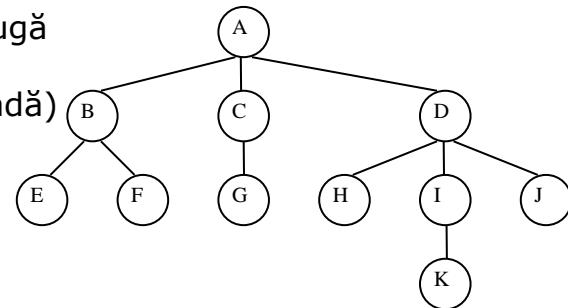
- Expandarea intr-un nod fiu și înaintarea în adâncime până când
 - Este găsit un nod țintă (obiectiv) sau
 - Nodul atins nu mai are fiu
- Cu revenirea în cel mai recent nod care mai poate fi explorat
- Toate nodurile fiu obținute prin expandarea nodului curent se adaugă într-o listă de tip **LIFO (stivă)**
- Similar cu BFS, dar nodurile se plasează într-o stivă (în loc de coadă)

Exemplu

- Ordinea vizitării nodurilor: A, B, E, F, C, G, D, H, I, K, J

Algoritm

```
bool DFS(elem, list){  
    found = false;  
    visited = Φ;  
    toVisit = {start};      //LIFO list  
    while((toVisit != Φ) && (!found)){  
        node = pop(toVisit);  
        visited = visited U {node};  
        if (node== elem)  
            found = true;  
        else{  
            aux = Φ;  
            for all (unvisited) children of node do{  
                aux = aux U {child};  
            }  
            toVisit = aux U toVisit;  
        }  
    } //while  
    return found;
```



Vizitate deja	De vizitat
Φ	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	K, J
A, B, E, F, C, G, D, H, I, K	J
A, B, E, F, C, G, D, H, I, K, J	Φ

SCnI în structuri arborescente căutare în adâncime (depth-first search – DFS)



■ Analiza complexității

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d_{max} - lungimea (adâncimea) maximă a unui arbore explorat
 - $T(n) = 1 + b + b^2 + \dots + b^{d_{max}} \Rightarrow O(b^{d_{max}})$
- Complexitate spațială
 - $S(n) = b * d_{max}$
- Completitudine
 - Nu → algoritmul nu se termină pt drumurile infinite (neexistând suficientă memorie pt reținerea nodurilor deja vizitate)
- Optimalitate
 - Nu → căutarea în adâncime poate găsi un drum soluție mai lung decât drumul optim

■ Avantaje

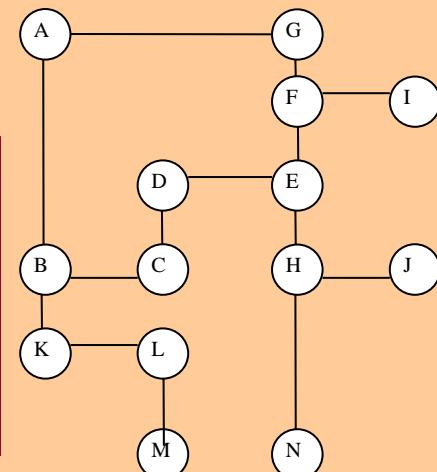
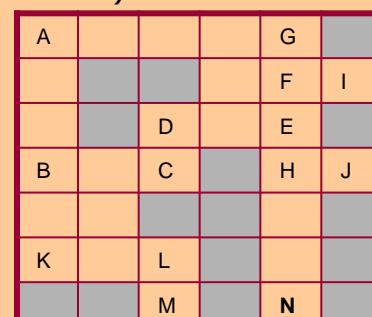
- Găsirea drumului de lungime minimă până la nodul obiectiv cu consum minim de memorie
 - versiunea recursivă

■ Dezavantaje

- Se poate bloca pe anumite drumuri greșite (nenorocoase) fără a putea reveni
 - Ciclu infinit
 - Găsirea unei soluții mai "lungi" decât soluția optimă

■ Aplicații

- Problema labirintului (maze)
- Identificarea componentelor conexe
- Sortare topologică
- Testarea planarității



SCnI în structuri arborescente căutare în adâncime (depth-first search – DFS)



```

bool DFS_edges(elem, list){
    discovered = Φ;
    back = Φ;
    toDiscover = Φ;      //LIFO
    for (all neighbours of start) do
        toDiscover = toDiscover U {(start, neighbour)}
    found = false;
    visited = {start};
    while((toDiscover != Φ) && (!found)){
        edge = pop(toDiscover);
        if (edge.out !∈ visited){
            discovered = discovered U {edge};
            visited = visited U {edge.out}
            if (edge.out == end)
                found = true;
            else{
                aux = Φ;
                for all neighbours of edge.out do{
                    aux = aux U {(edge.out, neighbour)};
                }
                toDiscover = aux U toDiscover;
            }
        }
        else
            back = back U {edge};
    } //while
    return found;
}

```

Muchia	Muchii vizitate deja	Muchii de vizitat	înapoi	Noduri vizitate
	Φ	AB, AF	Φ	A
AB	AB	BC, BK, AF	Φ	A, B
BC	AB, BC	CD, BK, AF	Φ	A, B, C
CD	AB, BC, CD	DE, BK, AF	Φ	A, B, C, D
DE	AB, BC, CD, DE	EF, EH, BK, AF	Φ	A, B, C, D, E
EF	AB, BC, CD, DE, EF	FI, FG, EH, BK, AF	Φ	A, B, C, D, E, F
FI	AB, BC, CD, DE, EF, FI	FG, EH, BK, AF	Φ	A, B, C, D, E, F, I
FG	AB, BC, CD, DE, EF, FI, FG	GA, EH, BK, AF	Φ	A, B, C, D, E, F, I, G
GA	AB, BC, CD, DE, EF, FI, FG	EH, BK, AF	GA	A, B, C, D, E, F, I, G
EH	AB, BC, CD, DE, EF, FI, FG	HJ, HN, BK, AF	GA	A, B, C, D, E, F, I, G, H
HJ	AB, BC, CD, DE, EF, FI, FG, HJ	HN, BK, AF	GA	A, B, C, D, E, F, I, G, H, J
HN	AB, BC, CD, DE, EF, FI, FG, HI, HN	BK, AF	GA	A, B, C, D, E, F, I, G, H, N

SCnI în structuri arborescente

căutare în adâncime limitată (depth-limited search – DLS)



Aspecte teoretice

- DFS + adâncime maximă care limitează căutarea (d_{lim})
- Se soluționează problemele de completitudine ale căutării în adâncime (DFS)

Exemplu

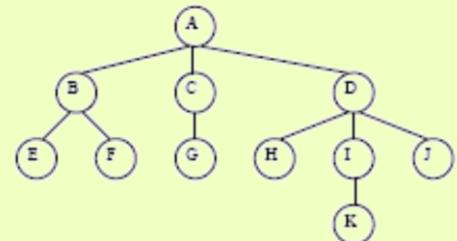
- $d_{lim} = 2$
- Ordinea vizitării nodurilor: A, B, E, F, C, G, D, H, I, J

Algoritm

```

bool DLS(elem, list, dlim) {
    found = false;
    visited = {};
    toVisit = {start}; //LIFO list
    while((toVisit != {}) && (!found)) {
        node = pop(toVisit);
        visited = visited ∪ {node};
        if (node.depth <= dlim) {
            if (node == elem)
                found = true;
            else{
                aux = {};
                for all (unvisited) children of node do{
                    aux = aux ∪ {child};
                }
                toVisit = aux ∪ toVisit;
            } //if found
        } //if dlim
    } //while
    return found;
}

```



Vizitate deja	De vizitat
\emptyset	A
A	B, C, D
A, B	E, F, C, D
A, B, E	F, C, D
A, B, E, F	C, D
A, B, E, F, C	G, D
A, B, E, F, C, G	D
A, B, E, F, C, G, D	H, I, J
A, B, E, F, C, G, D, H	I, J
A, B, E, F, C, G, D, H, I	J
A, B, E, F, C, G, D, H, I, K, J	\emptyset



SCnI în structuri arborescente

căutare în adâncime limitată (depth-limited search – DLS)

□ Analiza complexității

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d^{lim} – limita lungimii (adâncimii) permisă pentru un arbore explorat
 - $T(n) = 1 + b + b^2 + \dots + b^{d_{lim}} \Rightarrow O(b^{d_{lim}})$
- Complexitate spațială
 - $S(n) = b * d_{lim}$
- Completitudine
 - Da, dar $\Leftrightarrow d_{lim} > d$, unde d = lungimea (adâncimea) soluției optime
- Optimalitate
 - Nu \rightarrow căutarea în adâncime poate găsi un drum soluție mai lung decât drumul optim

□ Avantaje

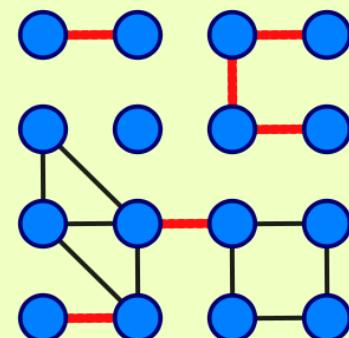
- Se soluționează problemele de completitudine ale căutării în adâncime (DFS)

□ Dezavantaje

- Cum se alege o limită d_{lim} bună?

□ Aplicații

- Determinarea “podurilor” într-un graf



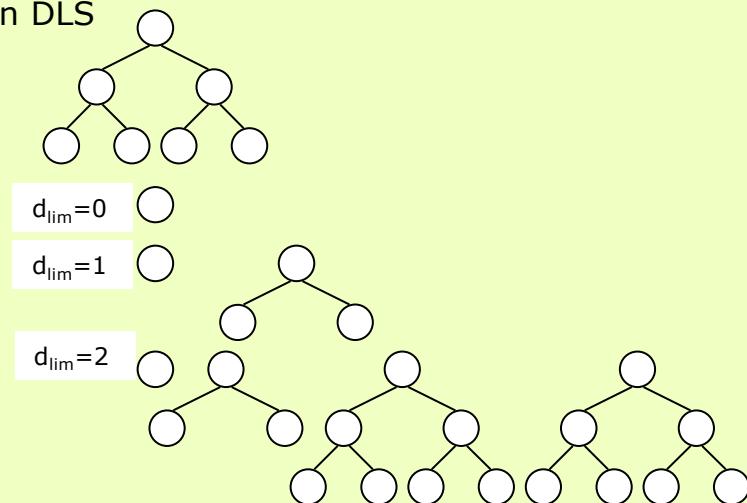
SCnI în structuri arborescente – căutare în adâncime iterativă (iterative deepening depth search – IDDS)



□ Aspecte teoretice

- U DLS(d_{lim}), unde $d_{lim} = 1, 2, 3, \dots, d_{max}$
- Se soluționează problema stabilitării limitei d_{lim} optime din DLS
- De obicei, se aplică acolo unde:
 - spațiul de căutare este mare și
 - se cunoaște lungimea (adâncimea) soluției

□ Exemplu



□ Algoritm

```
bool IDS(elem, list){  
    found = false;  
    dlim = 0;  
    while ((!found) && (dlim < dmax)) {  
        found = DLS(elem, list, dlim);  
        dlim++;  
    }  
    return found;  
}
```



SCnI în structuri arborescente – căutare în adâncime iterativă (iterative deepening depth search – IDDS)

□ Analiza complexității

- Complexitate temporală:
 - Nodurile situate la adâncimea d_{max} (în număr de $b^{d_{max}}$) se expandează o singură dată => $1 * b^{d_{max}}$
 - Nodurile situate la adâncimea $d_{max}-1$ (în număr de $b^{d_{max}-1}$) se expandează de 2 ori => $2 * (b^{d_{max}-1})$
 - ...
 - Nodurile situate la adâncimea 1 (în număr de b) se expandează de d_{max} ori => $d_{max} * b^1$
 - Nodurile situate la adâncimea 0 (în număr de 1 - rădăcina) se expandează de $d_{max}+1$ ori => $(d_{max}+1)*b^0$

$$T(n) = \sum_{i=0}^{d_{max}} (i+1)b^{d_{max}-i} \Rightarrow O(b^{d_{max}})$$

- Complexitate spațială
 - $S(n) = b * d_{max}$
- Completitudine
 - Da
- Optimalitate
 - Da

□ Avantaje

- Necesită memorie liniară
- Asigură atingerea nodului întă urmând un drum de lungime minimă
- Mai rapidă decât BFS și DFS

□ Dezavantaje

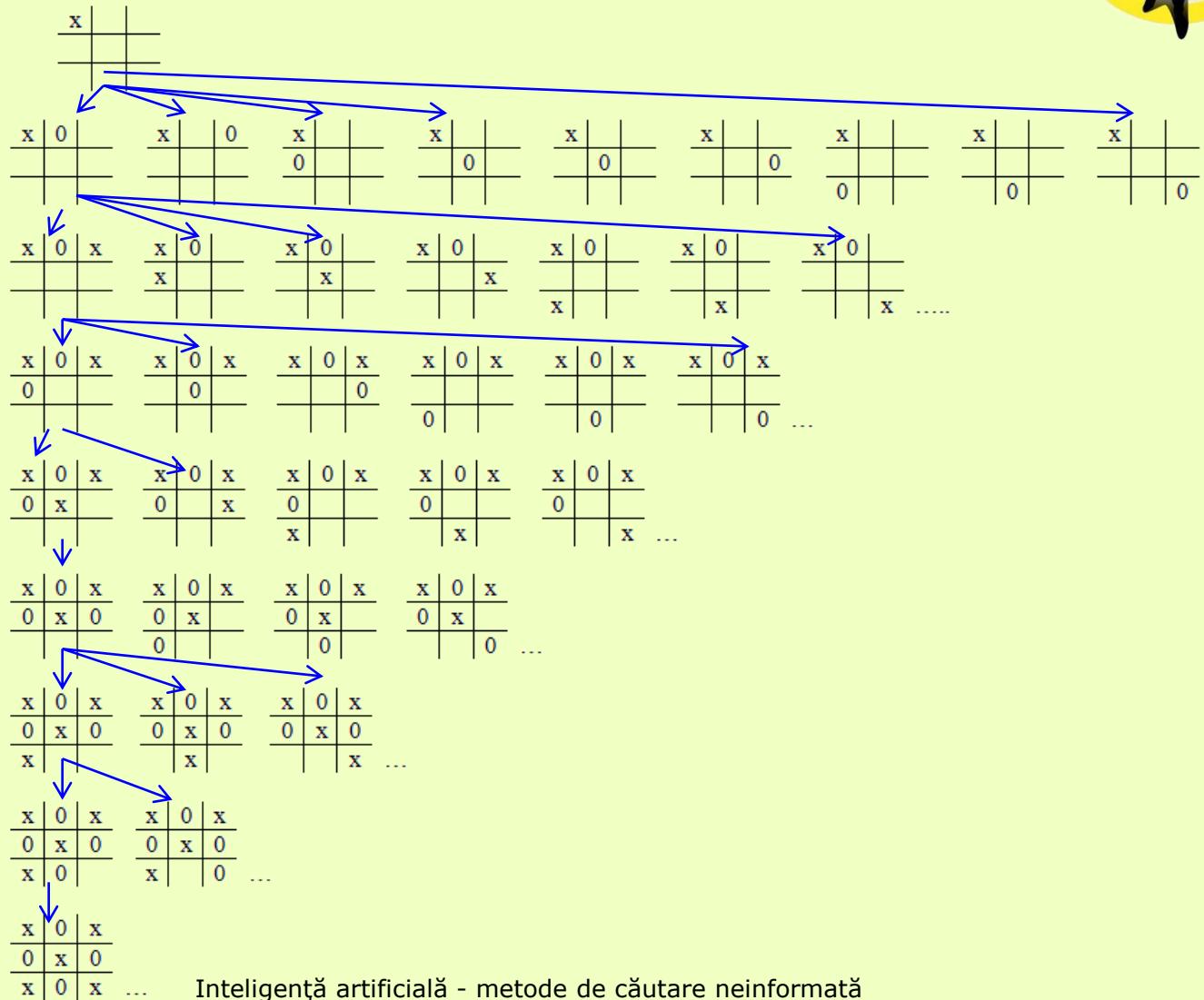
- Necesită cunoașterea "adâncimii" soluției

□ Aplicații

- Jocul Tic tac toe



SCnI în structuri arborescente – căutare în adâncime iterativă (iterative deepening depth search – IDDS)



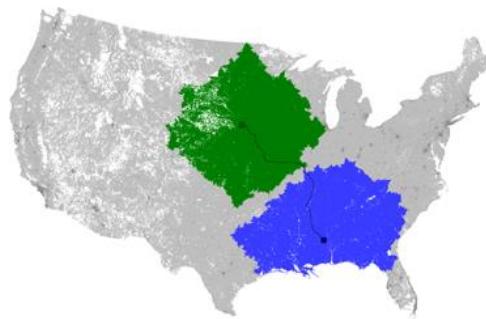


SCnI în structuri arborescente căutare bidirecțională (bi-directional search – BDS)

□ Aspecte teoretice

- 2 căutări simultane
 - Înainte (*forward*): de la rădăcină spre frunze
 - Înapoi (*backward*): de la frunze spre rădăcină
- care se opresc atunci când ajung la un nod comun
- într-o direcție pot fi folosite oricare dintre stategiile de căutare anterioare
- necesită
 - stabilirea succesorilor, respectiv a predecesorilor unui nod
 - stabilirea locului de întâlnire

□ Exemplu



□ Algoritm

- În funcție de strategia de căutare folosită



SCnI în structuri arborescente căutare bidirecțională (bi-directional search – BDS)

□ Analiza complexității

- Complexitate temporală
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d - lungimea (adâncimea) soluției
 - $O(b^{d/2}) + O(b^{d/2}) \Rightarrow O(b^{d/2})$
- Complexitate spațială
 - $S(n) = T(n)$
- Completitudine
 - da
- Optimalitate
 - da

□ Avantaje

- Complexitate spațială și temporală redusă

□ Dezavantaje

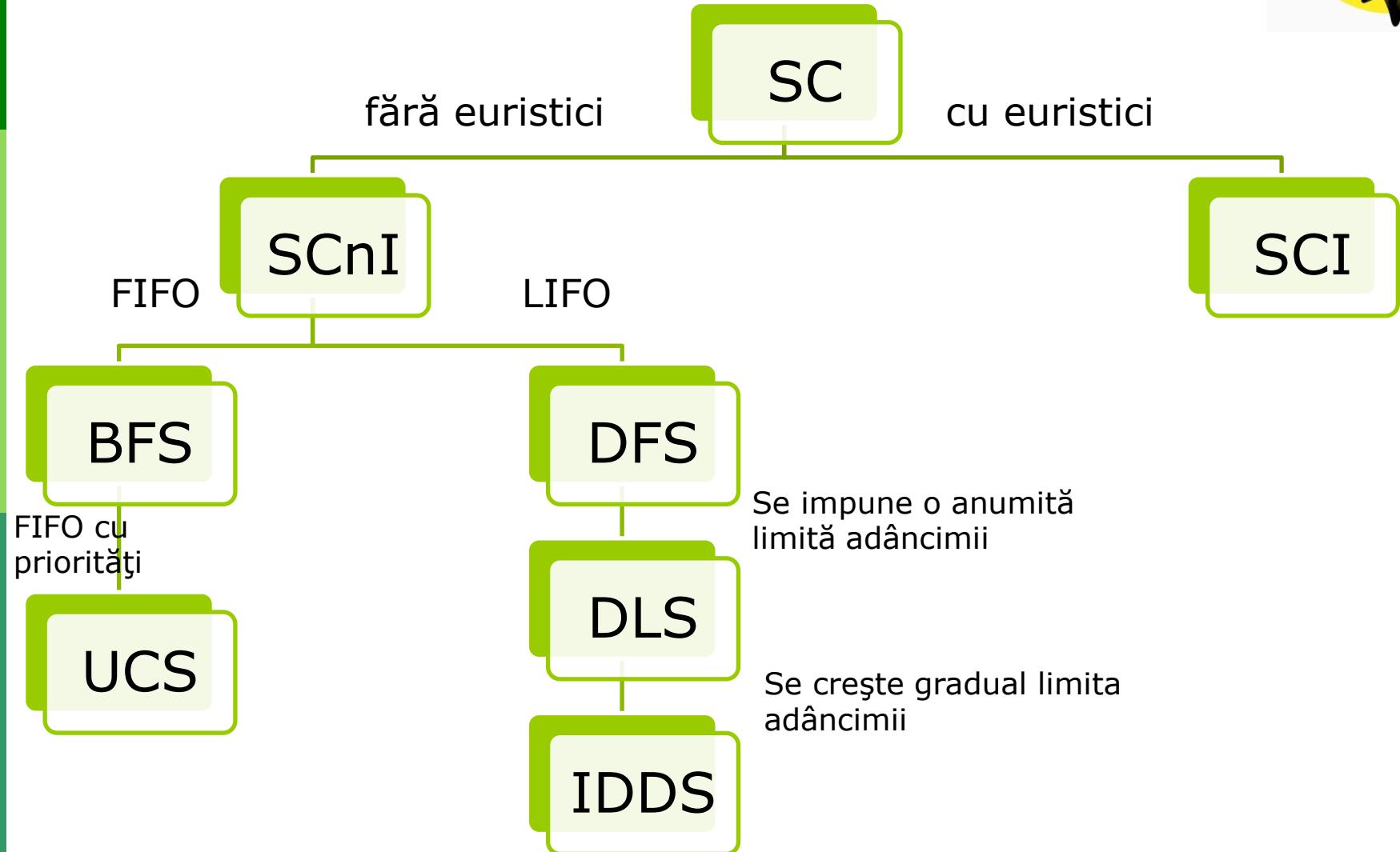
- Dificultăți în formularea problemei astfel încât fiecare stare să poată fi inversată
 - Parcurgere dinspre cap spre coadă
 - Parcurgere dinspre coadă spre cap
- Implementare dificilă
- Trebuie determinați succesorii și predecesorii tuturor stărilor
- Trebuie cunoscută starea finală (obiectiv)

□ Aplicații

- Problema partiționării
- Cel mai scurt drum



SCnI în structuri arborescente





SCnI în structuri arborescente

Compararea performanțelor

Metoda de căutare	Complexitate temporală	Complexitate spațială	Completitudine	Optimalitate
BFS	$O(b^d)$	$O(b^d)$	Da	Da
UCS	$O(b^d)$	$O(b^d)$	Da	Da
DFS	$O(b^{d_{\max}})$	$O(b * d_{\max})$	Nu	Nu
DLS	$O(b^{d_{\text{lim}}})$	$O(b * d_{\text{lim}})$	Da dacă $d_{\text{lim}} > d$	Nu
IDS	$O(b^d)$	$O(b * d)$	Da	Da
BDS	$O(b^{d/2})$	$O(b^{d/2})$	Da	Da



Rezolvarea problemelor prin căutare

❑ Strategii de căutare (SC)

■ Topologie

- ❑ În funcție de informația disponibilă
 - SC ne-informate (oarbe)
 - SC informate (euristice)



Strategii de căutare informate (SCI)

□ Caracteristici

- se bazează pe informații specifice problemei încercând să restrângă căutarea prin alegerea intelligentă a nodurilor care vor fi explorate
- ordonarea nodurilor se face cu ajutorul unei funcții (euristici) de evaluare
- sunt particulare

□ Topologie

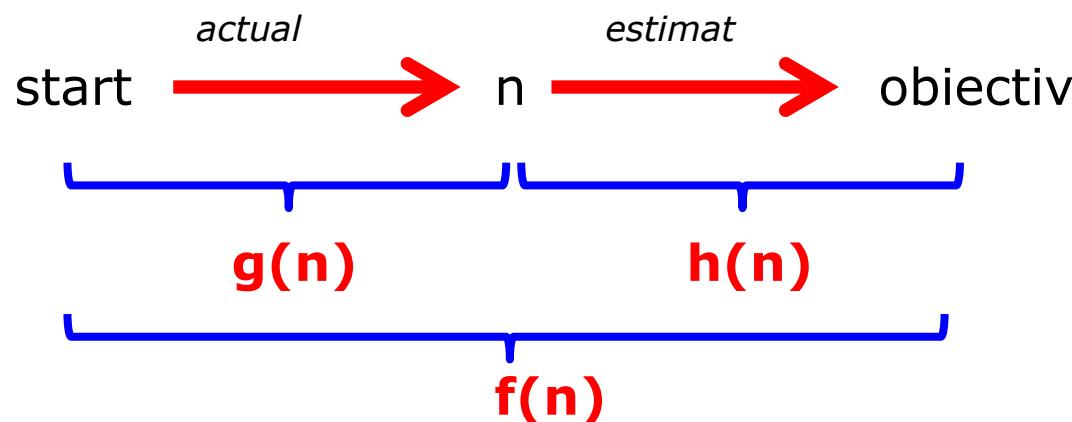
- Strategii globale
 - Best-first search
 - Greedy best-first search
 - A* + versiuni ale A*
- Strategii locale
 - Căutare tabu
 - *Hill climbing*
 - *Simulated annealing*

SC în structuri arborescente



❑ Noțiuni necesare

- $f(n)$ – funcție de evaluare pentru estimarea costului soluției prin nodul (starea) n
- $h(n)$ – funcție euristică pentru estimarea costului drumului de la nodul n la nodul obiectiv
- $g(n)$ – funcție de cost pentru estimarea costului drumului de la nodul de start până la nodul n
- $f(n) = g(n) + h(n)$



SCI – Best first search



□ Aspecte teoretice

- Best first search = mai întâi cel mai bun
- Se determină costul fiecărei stări cu ajutorul funcției de evaluare f
- Nodurile de expandant sunt reținute în structuri (cozi) ordonate
- Pentru expandare se alege starea cu cea mai bună evaluare
 - Stabilirea nodului care urmează să fie expandat
- Exemple de SC care depind de funcția de evaluare
 - Căutare de cost uniform (SCnI)
 - $f = \text{costul drumului}$
 - În SCI se folosesc funcții euristice
- Două categorii de SCI de tip best first search
 - SCI care încearcă să expandeze nodul cel mai apropiat de starea obiectiv
 - SCI care încearcă să expandeze nodul din soluția cu costul cel mic

□ Exemplu

- Detalii în slide-urile următoare



SCI – Best first search

□ Algoritm

```
bool BestFS(elem, list){  
    found = false;  
    visited =  $\emptyset$ ;  
    toVisit = {start}; //FIFO sorted list (priority queue)  
    while((toVisit !=  $\emptyset$ ) && (!found)){  
        if (toVisit ==  $\emptyset$ )  
            return false  
        node = pop(toVisit);  
        visited = visited U {node};  
        if (node == elem)  
            found = true;  
        else  
            aux =  $\emptyset$ ;  
        for all unvisited children of node do{  
            aux = aux U {child};  
        }  
        toVisit = toVisit U aux; //adding a node into the FIFO list based on its  
                           // evaluation (best one in the front of list)  
    } //while  
    return found;  
}
```

SCI – best first search



□ Analiza căutării

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d - lungimea (adâncimea) maximă a soluției
 - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Complexitate spațială
 - $S(n) = T(n)$
- Completitudine
 - Nu- drumuri infinite dacă euristică evaluează fiecare stare din drum ca fiind cea mai bună alegere
- Optimalitate
 - Posibil – depinde de euristică

□ Avantaje

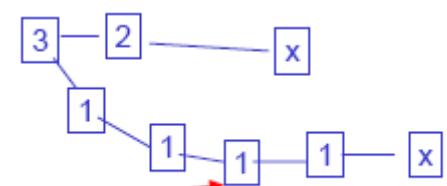
- Informațiile despre domeniul problemei ajută căutarea (SCI)
- Viteză mai mare de a ajunge la starea obiectiv

□ Dezavantaje

- Necesită evaluarea stărilor → efort computațional, dar nu numai
- Anumite path-uri pot “arăta” ca fiind bune conform funcției euristicice

□ Aplicații

- *Web crawler (automatic indexer)*
- Jocuri



SCI – Funcții euristice



- Etimologie: *heuriskein* (gr)
 - *a găsi, a descoperi*
 - *studiu metodelor și regulilor de descoperire și inventie*
- Utilitate
 - Evaluarea potențialului unei stări din spațiul de căutare
 - Estimarea costului drumului (în arborele de căutare) din starea curentă până în starea finală (cât de aproape de țintă a ajuns căutarea)
- Caracteristici
 - Depind de problema care trebuie rezolvată
 - Pentru probleme diferite trebuie proiectate sau învățate diferite euristici
 - Se evaluatează o anumită stare (nu operatorii care transformă o stare în altă stare)
 - Funcții pozitive pentru orice stare n
 - $h(n) \geq 0$ pentru orice stare n
 - $h(n) = 0$ pentru starea finală
 - $h(n) = \infty$ pentru o stare din care începe un drum mort (o stare din care nu se poate ajunge în starea finală)

SCI – Funcții euristică



□ Exemple

- Problema misionarilor și canibalilor
 - $h(n)$ – nr. persoanelor aflate pe malul inițial
- 8-puzzle
 - $h(n)$ – nr pieselor care nu se află la locul lor
 - $h(n)$ – suma distanțelor Manhattan (la care se află fiecare piesă de poziția ei finală)
- Problema comisului voiajor
 - $h(n)$ – cel mai apropiat vecin !!!
- Plata unei sume folosind un număr minim de monezi
 - $h(n)$ – alegerea celei mai valoroase monezi mai mică decât suma (rămasă) de plată

SCI - Greedy



□ Aspecte teoretice

- Funcția de evaluaare $f(n) = h(n)$
 - estimarea costului drumului de la starea curentă la starea finală – $h(n)$
 - minimizarea costului drumului care mai trebuie parcurs

□ Exemplu

- A,D,E,H

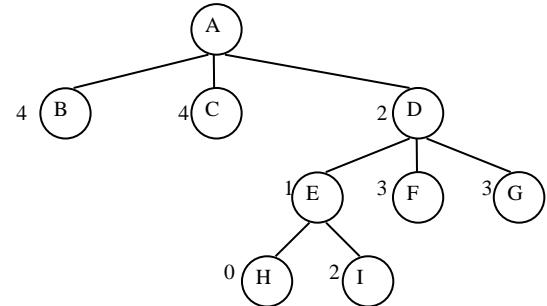
□ Algoritm

```

bool BestFS(elem, list){
    found = false;
    visited = {};
    toVisit = {start};           //FIFO sorted list (priority queue
    while((toVisit != {}) && (!found)){
        if (toVisit == {})
            return false;
        node = pop(toVisit);
        visited = visited U {node};
        if (node == elem)
            found = true;
        else
            aux = {};
        for all unvisited children of node do{
            aux = aux U {child};
        }
        toVisit = toVisit U aux; //adding a node onto the FIFO list based on its evaluation  $h(n)$ 
                                //(best one in the front of list)

    } //while
    return found;
}

```



Vizitate deja	De vizitat
\emptyset	A
A	D, B, C
A, D	E, F, G, B, C
A, D, E	H, I, F, G, B, C
A, D, E, H	\emptyset

SCI - Greedy



□ Analiza căutării:

- Complexitate temporală → DFS
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d^{\max} – lungimea (adâncimea) maximă a unui arbore explorat
 - $T(n) = 1 + b + b^2 + \dots + b^{d^{\max}} \Rightarrow O(b^{d^{\max}})$
- Complexitate spațială → BFS
 - d – lungimea (adâncimea) soluției
 - $S(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Completitudine
 - Nu (există posibilitatea intrării în cicluri infinite)
- Optimalitate
 - posibil

□ Avantaje

- Găsirea rapidă a unei soluții (dar nu neapărat soluția optimă), mai ales pentru probleme mici

□ Dezavantaje

- Suma alegerilor optime de la fiecare pas nu reprezintă alegerea globală optimă
 - Ex. Problema comisului voiajor

□ Aplicații

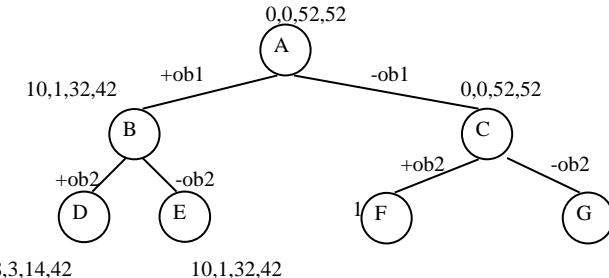
- Probleme de planificare
- Probleme de sume parțiale
 - Plata unei sume folosind diferite tipuri de monezi
 - Problema rucsacului
- Puzzle-uri
- Drumul optim într-un graf

SCI – A*



□ Aspecte teoretice

- Combinarea aspectelor pozitive ale
 - căutării de cost uniform
 - optimalitate și completitudine
 - utilizarea unei cozi ordonate
 - căutării Greedy
 - viteza mare
 - ordonare pe baza unei funcții de evaluare
- Funcția de evaluaire $f(n)$
 - estimarea costului celui mai bun drum care trece prin nodul n
 - $f(n) = g(n) + h(n)$
 - $g(n)$ – funcție folosită pentru stabilirea costului drumului de la starea inițială până la starea curentă n
 - $h(n)$ – funcție euristică folosită pentru estimarea costului drumului de la starea curentă n la starea finală
- Minimizarea costului total al unui drum



□ Exemplu

- Problema rucsacului de capacitate W în care pot fi puse n obiecte (o_1, o_2, \dots, o_n) fiecare având o greutate w_i și aducând un profit $p_i, i=1,2,\dots,n$
 - Soluția: pentru un rucsac cu $W = 5 \rightarrow$ alegerea obiectelor o_1 și o_3
- $g(n) = \sum p_i$, pentru acele obiecte o_i care au fost selectate
- $h(n) = \sum p_j$, pentru acele obiecte care nu au fost selectate și $\sum w_j \leq W - \sum w_i$
- Fiecare nod din graf este un tuplu: (p, w, p^*, f) , unde:
 - p – profitul adus de obiectele deja selectate (funcția $g(n)$)
 - w – greutatea obiectelor selectate
 - p^* - profitul maxim care se poate obține pornind din starea curentă și ținând cont de locul disponibil în rucsac (funcția $h(n)$)

	o_1	o_2	o_3	o_4
p_i	10	18	32	14
w_i	1	2	4	3

SCI – A*



■ Algoritm

```
bool BestFS(elem, list){  
    found = false;  
    visited = Φ;  
    toVisit = {start}; //FIFO sorted list (priority queue  
    while((toVisit != Φ) && (!found)) {  
        if (toVisit == Φ)  
            return false  
        node = pop(toVisit);  
        visited = visited U {node};  
        if (node == elem)  
            found = true;  
        else  
            aux = Φ;  
        for all unvisited children of node do{  
            aux = aux U {child};  
        }  
        toVisit = toVisit U aux; //adding a node onto the FIFO list  
                                // based on its evaluation  $f(n) = g(n) + h(n)$   
                                // (best one in the front of list)  
    } //while  
    return found;  
}
```

SCI – A*



□ **Analiza căutării:**

- Complexitate temporală:
 - b – factor de ramificare (nr de noduri fii ale unui nod)
 - d^{\max} – lungimea (adâncimea) maximă a unui arbore explorat
 - $T(n) = 1 + b + b^2 + \dots + b^{d^{\max}} \Rightarrow O(b^{d^{\max}})$
- Complexitate spațială
 - d – lungimea (adâncimea) soluției
 - $T(n) = 1 + b + b^2 + \dots + b^d \Rightarrow O(b^d)$
- Completitudine
 - Da
- Optimalitate
 - Da

□ **Avantaje**

- Algoritmul care expandează cele mai puține noduri din arborele de căutare

□ **Dezavantaje**

- Utilizarea unei cantități mari de memorie

□ **Aplicații**

- Probleme de planificare
- Probleme de sume partiale
 - Plata unei sume folosind diferite tipuri de monezi
 - Problema rucsacului
- Puzzle-uri
- Drumul optim într-un graf

SCI – A*



□ Variante

- iterative deepening A* (IDA*)
- memory-bounded A* (MA*)
- simplified memory bounded A* (SMA*)
- recursive best-first search (RBFS)
- dynamic A* (DA*)
- real time A*
- hierarchical A*

□ Bibliografie suplimentară

- 02/A_IDA.pdf
- 02/A_IDA_2.pdf
- 02/SMA_RTA.pdf
- 02/Recursive Best-First Search.ppt
- 02/IDS.pdf
- 02/IDA_MA.pdf
- <http://en.wikipedia.org/wiki/IDA%2A>
- <http://en.wikipedia.org/wiki/SMA%2A>



Rezolvarea problemelor prin căutare

□ Tipologia strategiilor de căutare

- În funcție de modul de **generare** a soluției
 - Căutare **constructivă**
 - Construirea progresivă a soluției
 - Ex. TSP
 - Căutare **perturbativă**
 - O soluție candidat este modificată în vederea obținerii unei noi soluții candidat
 - Ex. SAT - Propositional Satisfiability Problem
- În funcție de modul de **traversare** a spațiului de căutare
 - Căutare **sistemantică**
 - Traversarea completă a spațiului
 - Ideintificarea soluției dacă ea există → algoritmi compleți
 - Căutare **locală**
 - Traversarea spațiului de căutare dintr-un punct în alt punct vecin → algoritmi incompleți
 - O stare a spațiului poate fi vizitată de mai multe ori
- În funcție de elementele de **certitudine** ale căutării
 - Căutare **deterministă**
 - Algoritmi de identificare exactă a soluției
 - Căutare **stocastică**
 - Algoritmi de aproximare a soluției
- În funcție de stilul de **explorare** a spațiului de căutare
 - Căutare **secvențială**
 - Căutare **paralelă**



Rezolvarea problemelor prin căutare

Poate consta în:

- ❑ Construirea progresivă a soluției
- ❑ Identificarea soluției potențiale optime
 - Componentele problemei
 - ❑ Condiții (constrângeri) pe care trebuie să le satisfacă (parțial sau total) soluția
 - ❑ Funcție de evaluare a unei soluții potențiale → identificarea optimului
 - Spațiul de căutare
 - ❑ mulțimea tuturor soluțiilor potențiale complete
 - ❑ Stare = o soluție completă
 - ❑ Stare finală (scop) → soluția optimă
 - Exemple
 - ❑ Problema celor 8 regine,
 - Stările posibile: configurații ale tablei de sah cu câte 8 regine
 - Operatori: modificarea coloanei în care a fost plasată una din regine
 - Scopul căutării: configurația în care nu există atacuri între regine
 - Funcția de evaluare: numărul de atacuri
 - ❑ probleme de planificare,
 - ❑ proiectarea circuitelor digitale, etc



www.shutterstock.com - 36774760



Rezolvarea problemelor prin căutare

Poate consta în:

- Construirea progresivă a soluției
- Identificarea soluției potențiale optime
 - Algoritmi

- Algoritmii discutați până acum explorau în mod **sistemantic** spațiul de căutare
 - De ex. A* → 10^{100} stări ≈ 500 variabile binare
- Problemele reale pot avea 10 000 – 100 000 variabile → nevoie unei alte categorii de algoritmi care explorează **local** spațiul de căutare (algoritmi iterativi)
- Ideea de bază:
 - se începe cu o stare care nu respectă anumite constrângeri pentru a fi soluție optimă și
 - se efectuează modificări pentru a elimina aceste violări (se deplasează căutarea într-o soluție vecină cu soluția curentă) astfel încât căutarea să se îndrepte spre soluția optimă
- Iterativi → se memorează o singură stare și se încearcă îmbunătățirea ei
 - versiunea inteligentă a algoritmului de forță brută
- Istoricul căutării nu este reținut

```
bool LS(elem, list){  
    found = false;  
    crtState = initState  
    while ((!found) && timeLimitIsNotExceeded) {  
        toVisit = neighbours(crtState)  
        if (best(toVisit) is better than crtState)  
            crtState = best(toVisit)  
        if (crtState == elem)  
            found = true;  
    } //while  
    return found;  
}
```



www.shutterstock.com - 36774760



Rezolvarea problemelor prin căutare

Poate consta în:

- ❑ Construirea progresivă a soluției
- ❑ Identificarea soluției potențiale optime



www.shutterstock.com - 36774796

- Avantaje

- ❑ Simplu de implementat
- ❑ Necesită puțină memorie
- ❑ Poate găsi soluții rezonabile în spații de căutare (continue) foarte mari pentru care alți algoritmi sistematici nu pot fi aplicați

- E utilă atunci când:

- ❑ Se pot genera soluții complete rezonabile
- ❑ Se poate alege un bun punct de start
- ❑ Există operatori pentru modificarea unei soluții complete
- ❑ Există o măsură pentru a aprecia progresul (avansarea căutării)
- ❑ Există un mod de a evalua soluția completă (în termeni de constrângeri violate)



Strategii de căutare locală

□ Tipologie

- Căutare locală simplă - se reține o singură stare vecină
 - Hill climbing → alege cel mai bun vecin
 - Simulated annealing → alege probabilistic cel mai bun vecin
 - Căutare tabu → reține lista soluțiilor recent vizitate
- Căutare locală în fascicol (*beam local search*) – se rețin mai multe stări (o populație de stări)
 - Algoritmi evolutivi
 - Optimizare bazată pe comportamentul de grup (*Particle swarm optimisation*)
 - Optimizare bazată pe furnici (*Ant colony optimisation*)



Strategii de căutare locală

□ Căutare locală simplă

■ elemente de interes special:

- Reprezentarea soluției
- Funcția de evaluare a unei potențiale soluții
- Vecinătatea unei soluții
 - Cum se definește/generează o soluție vecină
 - Cum are loc căutarea soluțiilor vecine
 - Aleator
 - Sistematic
- Criteriul de acceptare a unei noi soluții
 - Primul vecin mai bun decât soluția curentă
 - Cel mai bun vecin al soluției curente mai bun decât soluția curentă
 - Cel mai bun vecin al soluției curente mai slab decât soluția curentă
 - Un vecin aleator

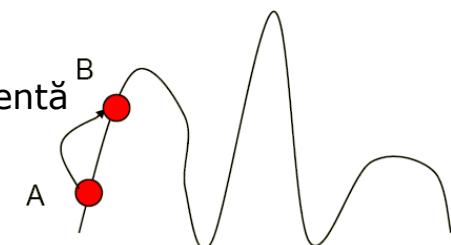
} dependente
de problemă

Strategii de căutare locală – Hill climbing (HC)



□ Aspecte teoretice

- Urcarea unui munte în condiții de ceață și amnezie a excursionistului :D
- Mișcarea continuă spre valori mai bune (mai mari → urcușul pe munte)
- Căutarea avansează în direcția îmbunătățirii valorii stărilor succesor până când se atinge un optim
- Criteriul de acceptare a unei noi soluții
 - cel mai bun vecin al soluției curente mai bun decât soluția curentă
- Îmbunătățire prin
 - Maximizarea calității unei stări → *steepest ascent HC*
 - Minimizarea costului unei stări → *gradient descent HC*
- $HC \neq \text{steepest ascent/gradient descent (SA/GD)}$
 - HC optimizează $f(x)$ cu $x \in R^n$ prin modificarea unui element al lui x
 - SA/GD optimizează $f(x)$ cu $x \in R^n$ prin modificarea tuturor elementelor lui x

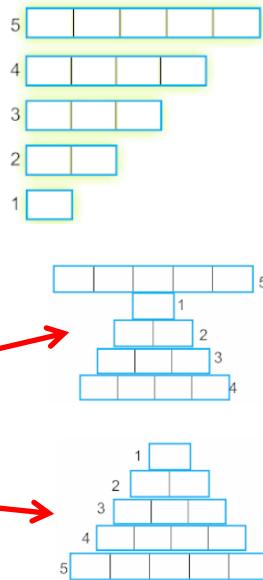


Strategii de căutare locală – Hill climbing (HC)



□ Exemplu

- Construirea unor turnuri din diferite forme geometrice
 - Se dau n piese de formă dreptunghiulară (de aceeași lățime, dar de lungimi diferite) așezate unele peste altele formând un turn (stivă). Să se re-ăseze piesele astfel încât să se formeze un nou turn știind că la o mutare se poate mișca doar o piesă din vârful stivei, piesă care poate fi mutată pe una din cele 2 stive ajutătoare.
 - Reprezentarea soluției
 - Stare x – vector de n perechi de forma (i,j) , unde i reprezintă indexul piesei ($i=1,2,\dots,n$), iar j indexul stivei pe care se află piesa ($j=1,2,3$)
 - Starea initială – vectorul corespunzător turnului inițial
 - Starea finală – vectorul corespunzător turnului final
 - Funcția de evaluare a unei stări
 - f_1 = numărul pieselor corect amplasate \rightarrow maximizare
 - conform turnului final – $f_1 = n$
 - f_2 = numărul pieselor greșit amplasate \rightarrow minimizare
 - conform turnului final – $f_2 = 0$
 - $f = f_1 - f_2 \rightarrow$ maximizare
 - Vecinătate
 - Mutări posibile
 - Mutarea unei piese i din vârful unei stive j_1 pe altă stivă j_2
 - Criteriul de acceptare a unei noi soluții
 - Cel mai bun vecin al soluției curente mai bun decât soluția curentă



Strategii de căutare locală – Hill climbing (HC)

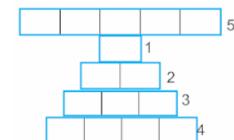


□ Exemplu

■ Iterația 1

□ Starea curentă x = starea inițială:

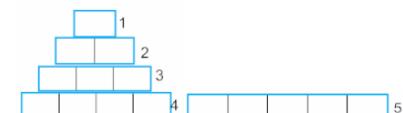
- $x = s_1 = ((5,1), (1,1), (2,1), (3,1), (4,1))$
- Piesele 1, 2 și 3 sunt corect așezate
- Piesele 4 și 5 nu sunt corect așezate
- $f(s_1) = 3 - 2 = 1$



□ $x^* = x$

□ Vecinii stării curente x – un singur vecin \rightarrow piesa 5 se mută pe stiva 2 \rightarrow

- $s_2 = ((1,1), (2,1), (3,1), (4,1), (5,2))$
- $f(s_2) = 4 - 1 = 3 > f(x) \rightarrow x = s_2$



Strategii de căutare locală – Hill climbing (HC)



❑ Exemplu

■ Iterația 2

- ❑ Starea curentă $x = ((1,1), (2,1), (3,1), (4,1), (5,2))$
 - $f(x) = 3$
- ❑ Vecinii stării curente – doi vecini:
 - piesa 1 se mută pe stiva 2 $\rightarrow s_3 = ((2,1), (3,1), (4,1), (1,2), (5,2)) \rightarrow f(s_3) = 3-2=1 < f(x)$



- piesa 1 se mută pe stiva 3 $\rightarrow s_4 = ((2,1), (3,1), (4,1), (5,2), (1,3)) \rightarrow f(s_4) = 3-2=1 < f(x)$



- nu există vecin de-al lui x mai bun ca $x \rightarrow$ stop

■ $x^* = x = ((1,1), (2,1), (3,1), (4,1), (5,2))$

■ Dar x^* este doar optim local, nu global

Strategii de căutare locală – Hill climbing (HC)



□ Exemplu

- Construirea unor turnuri din diferite forme geometrice
 - Funcția de evaluare a unei stări
 - f_1 = suma înălțimilor stivelor pe care sunt amplasate corect piese (conform turnului final – $f_1 = 10$) → maximizare
 - f_2 = suma înălțimilor stivelor pe care sunt amplasate incorect piese (conform turnului final – $f_2 = 0$) → minimizare
 - $f = f_1 - f_2 \rightarrow$ maximizare
 - Vecinătate
 - Mutări posibile
 - Mutarea unei piese i din vârful unei stive j_1 pe altă stivă j_2

Strategii de căutare locală – Hill climbing (HC)



□ Exemplu

■ Iterația 1

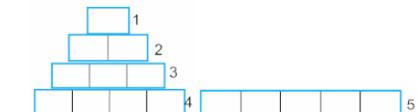
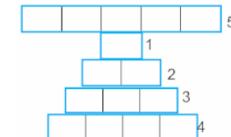
- Starea curentă x = starea inițială $s_1 = ((5,1), (1,1), (2,1), (3,1), (4,1))$

- Toate piesele nu sunt corect așezate $\rightarrow f_1 = 0, f_2 = 3+2 + 1 + 0 + 4 = 10$
 - $f(s_1) = 0 - 10 = -10$

- $x^* = x$

- Vecinii stării curente x – un singur vecin \rightarrow piesa 5 se mută pe stiva 2 $\rightarrow s_2 = ((1,1), (2,1), (3,1), (4,1), (5,2))$

- $f(s_2) = 0 - (3+2+1+0) = -6 > f(x) \rightarrow x = s_2$



Strategii de căutare locală – Hill climbing (HC)



□ Exemplu

■ Iterația 2

- Starea curentă $x = ((1,1), (2,1), (3,1), (4,1), (5,2))$
 - $f(x) = -6$
- Vecinii stării curente – doi vecini:
 - piesa 1 se mută pe stiva 2 $\rightarrow s_3 = ((2,1), (3,1), (4,1), (1,2), (5,2)) \rightarrow f(s_3) = 0 - (0+2+3+0) = -5 > f(x)$ 
 - piesa 1 se mută pe stiva 3 $\rightarrow s_4 = ((2,1), (3,1), (4,1), (5,2), (1,3)) \rightarrow f(s_4) = 0 - (1+2+1) = -4 > f(x)$ 
- cel mai bun vecin al lui x este $s_4 \rightarrow x = s_4$

■ Iterația 3

- ...

■ Se evită astfel blocarea în optimele locale

Strategii de căutare locală – Hill climbing (HC)



□ Algoritm

```
Bool HC(S) {  
    x = s1      //starea inițială  
    x*=x        // cea mai bună soluție găsită  
    (până la un moment dat)  
    k = 0        // numarul de iterații  
    while (not termiantion criteria) {  
        k = k + 1  
        generate all neighbours of x (N)  
        Choose the best solution s from N  
        if f(s) is better than f(x) then x = s  
        else stop  
    } //while  
    x* = x  
    return x*;  
}
```

Strategii de căutare locală – Hill climbing (HC)



□ **Analiza căutării**

- Convergența spre optimul local

□ **Avantaje**

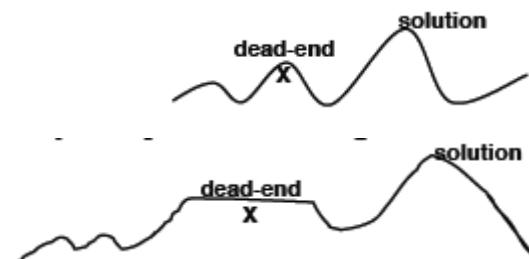
- Simplu de implementat → se poate folosi ușor pentru a aproxima soluția unei probleme când soluția exactă este dificil sau imposibil de găsit
 - Ex. TSP cu forate multe orașe
- Nu necesită memorie (nu se revine în starea precedentă)

□ **Dezavantaje**

- Funcția de evaluare (euristică) poate fi dificil de estimat
- Dacă se execută foarte multe mutări algoritmul devine inefficient
- Dacă se execută prea puține mutări algoritmul se poate bloca
 - Într-un optim local (nu mai poate "cobiști" din vârf)
 - Pe un platou – zonă din spațiul de căutare în care funcția de evaluare este constantă
 - Pe o creastă – saltul cu mai mulți pași ar putea ajuta căutarea

□ **Aplicații**

- Problema canibalilor
- 8-puzzle, **15-puzzle**
- TSP
- Problema damelor



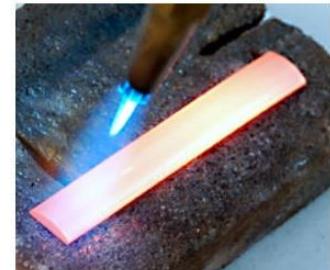
Strategii de căutare locală – Hill climbing (HC)



□ Variante

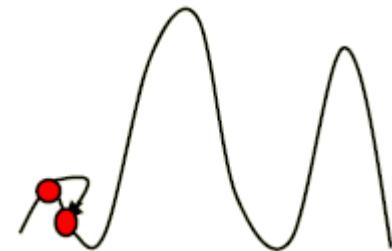
- HC stocastic
 - Alegerea aleatoare a unui succesor
- HC cu prima opțiune
 - Generarea aleatoare a successorilor până la întâlnirea unei mutări neefectuate
- HC cu repornire aleatoare → *beam local search*
 - Repornirea căutării de la o stare inițială aleatoare atunci când căutarea nu progresează

Strategii de căutare locală – Simulated Annealing



□ Aspecte teoretice

- Inspirată de modelarea proceselor fizice
 - Metropolis et al. 1953, Kirkpatrick et al. 1982;
- Succesorii stării curente sunt aleşi și în mod aleator
 - Dacă un succesor este mai bun decât starea curentă
 - atunci el devine noua stare curentă
 - altfel succesorul este reținut doar cu o anumită probabilitate
- Se permit efectuarea unor mutări “slabe” cu o anumită probabilitate p
 - → evadarea din optimele locale
- Probabilitatea $p = e^{\Delta E/T}$
 - Proportională cu valoarea diferenței (energia) ΔE
 - Modelată de un parametru de temperatură T
- Frecvența acestor mutări “slabe” și mărimea lor se reduce gradual prin scăderea temperaturii
 - $T = 0 \rightarrow$ hill climbing
 - $T \rightarrow \infty \rightarrow$ mutările “slabe” sunt tot mai mult executate
- Soluția optimă se identifică dacă temperatura se scade treptat (“slowly”)
- Criteriul de acceptare a unei noi soluții
 - Un vecin aleator mai bun sau mai slab (cu probabilitatea p) decât soluția curentă



Strategii de căutare locală – Simulated Annealing



□ Exemplu – Problema celor 8 regine

- Enunț
 - Să se amplaseze pe o tablă de săh 8 regine astfel încât ele să nu se atace reciproc
- Reprezentarea soluției
 - Stare x – permutare de n elemente $x = (x_1, x_2, \dots, x_n)$, unde x_i – linia pe care este plasată regina de pe coloana j
 - Nu există atacuri pe verticală sau pe orizontală
 - Pot exista atacuri pe diagonală
 - Starea inițială – o permutare oarecare
 - Starea finală – o permutare fără atacuri de nici un fel
- Funcția de evaluare a unei stări
 - f – suma reginelor atacate de fiecare regină → minimizare
- Vecinătate
 - Mutări posibile
 - Mutarea unei regine de pe o linie pe alta (interschimbarea a 2 elemente din permutare)
- Criteriul de acceptare a unei noi soluții
 - Un vecin oarecare al soluției curente
 - mai bun decât soluția curentă
 - mai slab decât soluția curentă – cu o anumită probabilitate $P(\Delta E) = e^{-\frac{\Delta E}{T}}$ unde:
 - ΔE – diferența de energie (evaluare) între cele 2 stări (vecină și curentă)
 - T – temperatura, $T(k) = 100/k$, unde k este nr iterăției

Strategii de căutare locală – Simulated Annealing



❑ Exemplu – Problema celor 8 regine

■ Iterația 1 ($k = 1$)

- ❑ Starea curentă x = starea inițială

$$s_1 = (8, 5, 3, 1, 6, 7, 2, 4)$$

- ❑ $f(s_1) = 1+1 = 2$

- ❑ $x^* = x$

- ❑ $T = 100/1 = 100$

- ❑ Un vecin al stării curente $x \rightarrow$ regina de pe linia 5 se interschimbă cu regina de pe linia 7

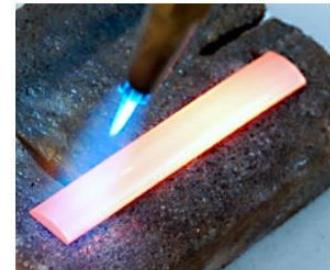
$$\rightarrow s_2 = (8, 7, 3, 1, 6, 5, 2, 4)$$

- ❑ $f(s_2) = 1+1+1=3 > f(x)$
- ❑ $\Delta E = f(s_2) - f(s_1) = 1$
- ❑ $P(\Delta E) = e^{-1/100}$
- ❑ $r = \text{random}(0,1)$
- ❑ Dacă $r < P(\Delta E) \rightarrow x = s_2$

	a	b	c	d	e	f	g	h	
1	Q								1
2					Q				2
3						Q			3
4									4
5			Q						5
6							Q		6
7								Q	7
8	Q								8
	a	b	c	d	e	f	g	h	

	a	b	c	d	e	f	g	h	
1	Q								1
2					Q				2
3				Q					3
4								Q	4
5								Q	5
6						Q			6
7		Q							7
8									8
	a	b	c	d	e	f	g	h	

Strategii de căutare locală – Simulated Annealing



□ Algoritm

```
bool SA(S){  
    x = s1           //starea inițială  
    x*=x // cea mai bună soluție găsită (până la un moment dat)  
    k = 0 // numarul de iterații  
    while (not termiantion criteria){  
        k = k + 1  
        generate a neighbour s of x  
        if f(s) is better than f(x) then x = s  
        else  
            pick a random number r (in (0,1) range)  
            if r < P( $\Delta E$ ) then x = s  
    } //while  
    x* = x  
    return x*;  
}
```

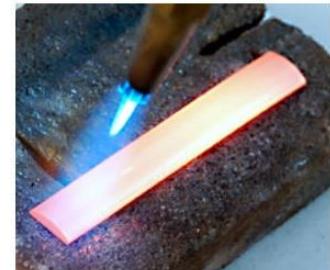
□ Criterii de oprire

- S-a ajuns la soluție
- S-a parcurs un anumit număr de iterații
- S-a ajuns la temepratura 0 (îngheț)

□ Cum se alege o probabilitate mică?

- $p = 0.1$
- p scade de-a lungul iterațiilor
- p scade de-a lungul iterațiilor și pe măsură ce "răul" $|f(s) - f(x)|$ crește
 - $p = \exp(-|f(s) - f(x)|/T)$
 - Unde T – temepratura (care crește)
 - Pentru o T mare se admite aproape orice vecin v
 - Pentru o T mică se admite doar un vecin mai bun decât s
 - Dacă "răul" e mare, atunci probabilitatea e mică

Strategii de căutare locală – Simulated Annealing



□ Analiza căutării

- Convergența (complet, optimal) lentă spre optimul global

□ Avantaje

- Algoritm fundamentat statistic → garantează găsirea soluției optime, dar necesită multe iterații
- Ușor de implementat
- În general găsește o soluție relativ bună (optim global)
- Poate rezolva probleme complexe (cu zgomot și multe constrângeri)

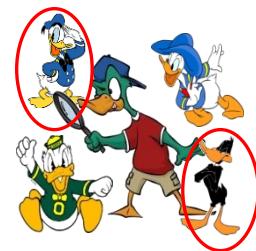
□ Dezavantaje

- Algoritm încet – convergența la soluție durează foarte mult timp
 - Compromis între calitatea soluției și timpul necesar calculării ei
- Depinde de anumiți parametri (temperatura) care trebuie reglați
- Nu se știe dacă soluția oferită este optimă (local sau global)
- Calitatea soluției găsite depinde de precizia variabilelor implicate în algoritm

□ Aplicații

- Probleme de optimizare combinatorială → problema rucsacului
- Probleme de proiectare → Proiectarea circuitelor digitale
- Probleme de planificare → Planificarea producției, planificarea meciurilor în turnurile de tenis US Open

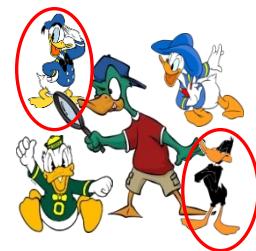
Strategii de căutare locală – Căutare tabu



□ Aspecte teoretice

- "Tabu" → interdicție socială severă cu privire la activitățile umane sacre și interzise
- Propusă în anii 1970 de către F. Glover
- Ideea de bază
 - se începe cu o stare care nu respectă anumite constrângeri pentru a fi soluție optimă și
 - se efectuează modificări pentru a elimina aceste violări (se deplasează căutarea în **cea mai bună soluție vecină** cu soluția curentă) astfel încât căutarea să se îndrepte spre soluția optimă
 - se memorează
 - **Starea curentă**
 - **Stările vizitate** până la momentul curent al căutării și **mutările efectuate** pentru a ajunge în fiecare din acele stări de-a lungul căutării (se memorează o **listă limitată de stări care nu mai trebuie revizitate**)
- Criteriul de acceptare a unei noi soluții
 - Cel mai bun vecin al soluției curente mai bun decât soluția curentă și nevizitat încă
- 2 elemente importante
 - Mutări tabu (T) – determinate de un proces non-Markov care se folosește de informațiile obținute în timpul căutării de-a lungul ultimelor generații
 - Condiții tabu – pot fi inegalități liniare sau legături logice exprimate în funcție de soluția curentă
 - Au rol în alegerea mutărilor tabu

Strategii de căutare locală – Căutare tabu



□ Exemplu

■ Enunț

- Plata sumei S folosind cât mai multe dintre cele n monezi de valori v_i (din fiecare monedă există b_i bucăți)

■ Reprezentarea soluției

- Stare x – vector de n întregi $x = (x_1, x_2, \dots, x_n)$ cu $x_i \in \{0, 1, 2, \dots, b_i\}$
- Starea inițială – aleator

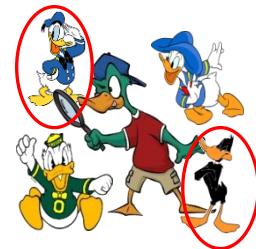
■ Funcția de evaluare a unei stări

- f_1 = Diferența între S și valoarea monezilor alese → minimă
 - Dacă valoarea monezilor depășește $S \rightarrow$ penalizare de 500 unități
- f_2 = Numărul monezilor selectate → maxim
- $f = f_1 - f_2 \rightarrow$ minimizare

■ Vecinătate

- Mutări posibile
 - Includerea în sumă a monezii i în j exemplare ($\text{plus}_{i,j}$)
 - Excluderea din sumă a monezii i în j exemplare ($\text{minus}_{i,j}$)
- Lista tabu reține mutările efectuate într-o iterație
 - Mutare – moneda adăugată/eliminată din sumă

Strategii de căutare locală – Căutare tabu



Exemplu

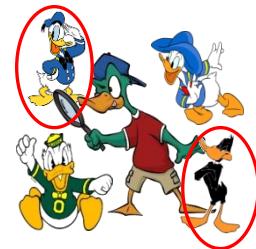
- $S = 500$, penalizare = 500, $n = 7$

$S=500$	m_1	m_2	m_3	m_4	m_5	m_6	m_7
v_i	10	50	15	20	100	35	5
b_i	5	2	6	5	5	3	10

Stare curentă	Val. f	Listă tabu	Stări vecine	Mutări	Val. f
2 0 1 0 0 2 1	384	\emptyset	2 0 1 3 0 2 1	plus _{4,3}	321
			2 0 1 0 0 3 1	plus _{6,1}	348
			0 0 1 0 0 2 1	minus _{1,2}	406
2 0 1 3 0 2 1	321	plus _{4,3}	2 0 1 3 5 2 1	plus _{5,5}	316
			2 0 1 1 0 2 1	minus _{4,2}	363
			2 1 1 3 0 2 1	plus _{2,1}	270
2 1 1 3 0 2 1	270	plus _{4,3} plus _{2,1}	...		

- Soluția finală: 4 1 5 4 1 3 10 ($f = -28$)

Strategii de căutare locală – Căutare tabu



□ Exemplu

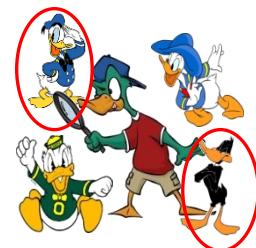
- $S = 500$, penalizare = 500, $n = 7$

$S=50$	m_1	m_2	m_3	m_4	m_5	m_6	m_7
v_i	10	50	15	20	100	35	5
b_i	5	2	6	5	4	3	10

Stare curentă	Val. f	Listă tabu	Stări vecine	Mutări	Val. f
2 0 1 0 0 2 1	384	\emptyset	1 0 1 4 0 2 1	minus _{1,1} , plus _{4,4}	311
			2 0 4 0 1 2 1	plus _{3,3} , minus _{5,1}	235
			2 0 1 0 4 2 6	plus _{5,4} , plus _{7,5}	450
2 0 4 0 1 2 1	235	plus _{3,3} , minus _{5,1}	2 0 5 0 5 2 1	plus _{3,1} , plus _{5,4}	315
			5 0 4 0 4 2 1	plus _{1,3} , plus _{5,3}	399
			2 2 4 0 5 2 1	plus _{2,2} , plus _{5,4}	739
2 0 4 0 1 2 1	235	plus _{3,3} , minus _{5,1}	...		

- Soluția finală: 4 1 5 4 1 3 10 ($f = -28$)

Strategii de căutare locală – Căutare tabu



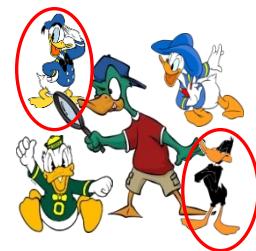
□ Algoritm

```
bool TS(S){  
    Select x∈S //S - spațiul de căutare  
    x*=x        //cea mai bună soluție (până la un mom. dat)  
    k = 0         // numarul de iterații  
    T = Ø         // listă de mutări tabu  
    while (not termiantion criteria){  
        k = k + 1  
        generate a subset of solutions in the neighborhood N-T of x  
        choose the best solution s from N-T and set x=s.  
        if f(x)<f(x*) then x*=x  
        update T with moves of generating x  
    } //while  
    return x*;  
}
```

■ Criterii de terminare

- Număr fix de iterații
- Număr de iterații fără îmbunătățiri
- Apropierea suficientă de soluție (dacă aceasta este cunoscută)
- Epuizarea elementelor nevizitate dintr-o vecinătate

Strategii de căutare locală – Căutare tabu



□ **Analiza căutării**

- Convergența rapidă spre optimul global

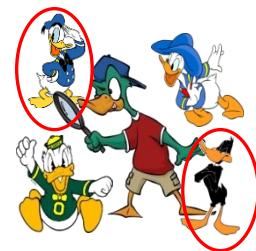
□ **Avantaje**

- Algoritm general și simplu de implementat
- Algoritm rapid (poate oferi soluția optimă globală în scurt timp)

□ **Dezavantaje**

- Stabilirea stărilor vecine în spații continue
- Număr mare de iterații
- Nu se garantează atingerea optimului global

Strategii de căutare locală – Căutare tabu



□ Aplicații

- Determinarea structurii tridimensionale a proteinelor în sevențe de aminoacizi (optimizarea unei funcții de potențial energetic cu multiple optime locale)
- Optimizarea traficului în rețele de telecomunicații
- Planificare în sisteme de producție
- Proiectarea rețelelor de telecomunicații optice
- Ghidaj automat pentru vehicule
- Probleme în grafuri (partiționări)
- Planificări în sistemele de audit
- Planificări ale task-urilor paralele efectuate de procesor (multiprocesor)
- Optimizarea structurii electomagnetice (imagistica rezonanței magnetice medicale)
- Probleme de asignare quadratică (proiectare VLSI)
- Probleme de combinatorică (ricsac, plata sumei)
- Problema tăierii unei bucăți în mai multe părți
- Controlul structurilor spațiale (NASA)
- Optimizarea proceselor cu decizii multi-stagiu
- Probleme de transport
- Management de portofoliu
- Chunking



Recapitulare

□ SCI best first search

- Nodurile mai bine evaluate (de cost mai mic) au prioritate la expandare

■ SCI de tip greedy

- minimizarea costului de la starea curentă la starea obiectiv – $h(n)$
- Timp de căutare < SCnI
- Ne-completă
- Ne-optimală

■ SCI de tip A*

- minimizarea costului de la starea inițială la starea curentă – $g(n)$ – și a costului de la starea curentă la starea obiectiv – $h(n)$
- Evitarea repetării stărilor
- Fără supraestimarea lui $h(n)$
- Timp și spațiu de căutare mare → în funcție de euristica folosită
- Complet
- Optimal



Recapitulare

❑ SC locale

■ Algoritmi iterativi

- ❑ Lucrează cu o soluție potențială → soluția optimă
- ❑ Se pot bloca în optime locale

	Alegerea stării următoare	Criteriul de acceptare	Convergența
HC	Cel mai bun vecin	Vecinul este mai bun decât strarea curentă	Optim local sau global
SA	Un vecin oarecare	Vecinul este mai bun sau mai slab (acceptat cu probabilitatea p) decât strarea curentă	Optim global (lentă)
TS	Cel mai bun vecin nevizitat încă	Vecinul este mai bun decât strarea curentă	Optim global (rapidă)

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

- Sisteme care învață singure

- Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi

- Sisteme bazate pe reguli

- Sisteme hibride

C. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare

- Strategii de căutare

- Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, **Algoritmi evolutivi, PSO, ACO**)
 - Strategii de căutare adversială

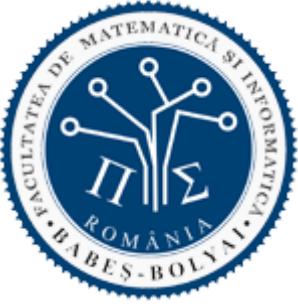
Cursul următor – Materiale de citit și legături utile

- capitolul 14 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- capitolul 7.6 din *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

-
- Informațiile prezentate au fost colectate din diferite surse bibliografice, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare

Strategii de căutare informată locală

Algoritmi Evolutivi

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

- Sisteme care învață singure

- Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi

- Sisteme bazate pe reguli

- Sisteme hibride

C. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare

- Strategii de căutare

- Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, **Algoritmi evolutivi, PSO, ACO**)
 - Strategii de căutare adversială

Sumar

- Rezolvarea problemelor prin căutare
 - Strategii de căutare informate (euristice) – SCI
 - Strategii locale
 - Algoritmi evolutivi

Materiale de citit și legături utile

- capitolul 14 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- capitolul 7.6 din *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

Căutare locală

□ Tipologie

- Căutare locală simplă - se reține o singură stare vecină
 - Hill climbing → alege cel mai bun vecin
 - Simulated annealing → alege probabilistic cel mai bun vecin
 - Căutare tabu → reține lista soluțiilor recent vizitate
- Căutare locală în fascicol (beam local search) – se rețin mai multe stări (o populație de stări)
 - Algoritmi evolutivi
 - Optimizare bazată pe comportamentul de grup (Particle swarm optimisation)
 - Optimizare bazată pe furnici (Ant colony optimisation)

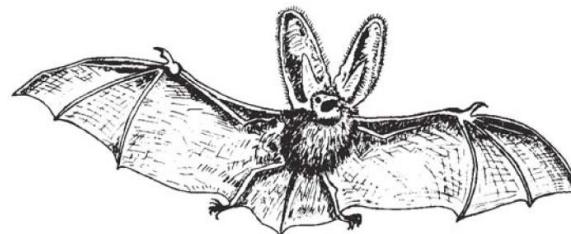
Algoritmi inspirați de natură

- Care este cea mai bună metodă de rezolvare a unei probleme?
 - Creierul uman
 - a creat roata, mașina, orașul, etc
 - Mecanismul evoluției
 - a creat creierul (mintea) umană
- Simularea naturii
 - Cu ajutorul mașinilor → rețelele neuronale artificiale simulează mintea umană
 - mașini de zbor, computere bazate pe ADN, computere cu membrane
 - Cu ajutorul algoritmilor
 - algoritmii evolutivi simulează evoluția naturii
 - algoritmii inspirați de comportamentul de grup simulează adaptarea colectivă și procesele sociale dintr-un colectiv (*Particle Swarm Optimisation*)
 - algoritmii inspirați de furnici (*Ant Colony Optimisation*)

Algoritmi evolutivi – aspecte teoretice

□ Simularea naturii

- Zborul liliencilor

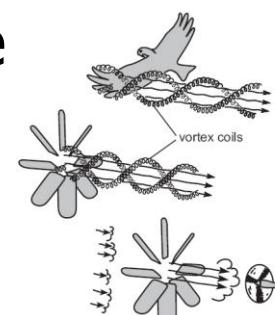


- Leonardo da Vinci – schema unei mașini de zbor



- Zborul păsărilor și al avioanelor

- Zborul păsărilor și turbinele eoliene



Algoritmi evolutivi – aspecte teoretice

- Care sunt caracteristicile de bază ale AE?
 - Implică procese iterative și paralele
 - Folosesc populații de potențiale soluții
 - Se bazează pe o căutare aleatoare
 - Sunt inspirați de biologie – implică mecanisme precum:
 - selecția naturală
 - reproducerea
 - recombinarea
 - mutația

Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

❑ Jean Baptise de Lamark (1744-1829)

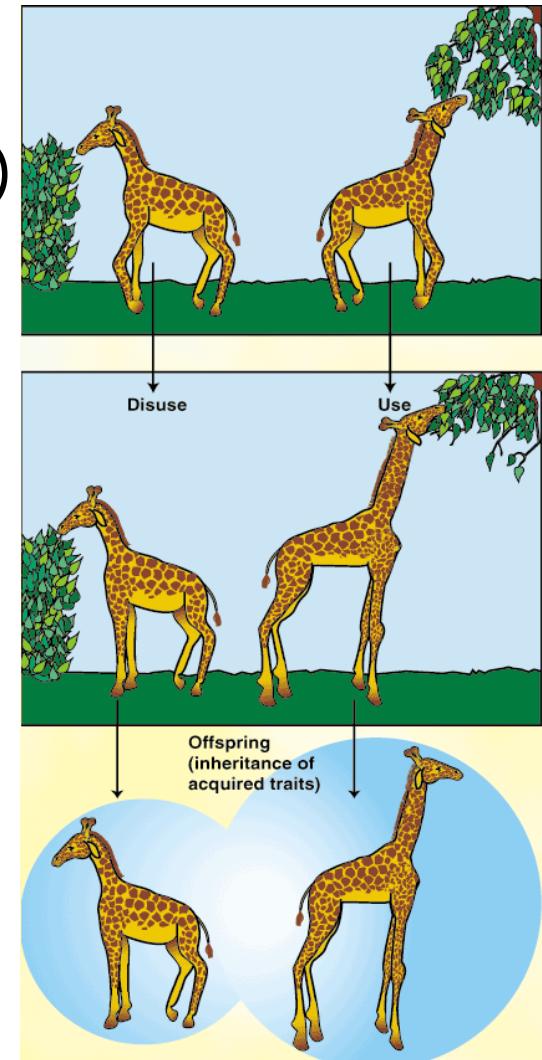
- A propus în 1809 o explicație pentru originea speciilor în cartea

Zoological Philosophy:

- ❑ Nevoile unui organism determină caracteristicile care evoluează
- ❑ Caracteristicile utile dobândite în cursul vieții unui organism se pot transfera urmașilor acestuia

■ Legea utilizării și neutilizării

- ❑ *use and disuse*



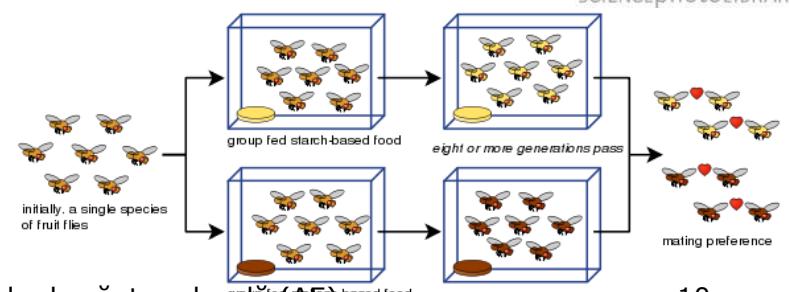
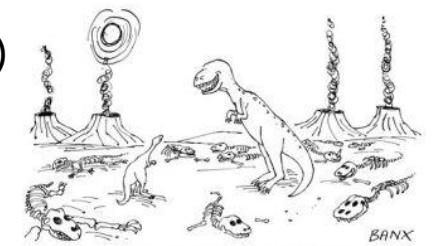
Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

□ Charles Darwin (1807-1882)

- În cartea *Origin of Species* demostrează că toate organismele au evoluat din alte organisme pe baza:

- variației
 - supraproducția de descendenți
- selecției naturale
 - competiția (generații constante ca dimensiune)
 - supraviețuirea pe baza calității/adaptării la mediul de viață (fitness)
 - reproducerea
 - apariția de specii noi



Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

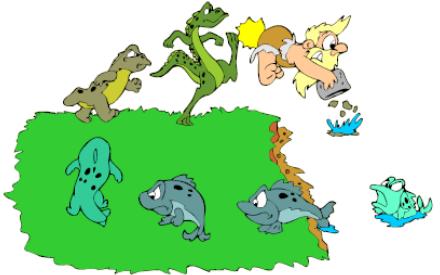
- ❑ Teoria evolutivă modernă
 - Îmbogățește teoria Darwiniană cu mecanismul moștenirii genetice
 - Variația genetică se produce prin:
 - mutație – spontană – și
 - reproducere sexuală
 - L. Fogel 1962 (San Diego, CA) → programare evolutivă – PE – (*Evolutionary Programming*)
 - J. Holland 1962 (Ann Arbor, MI) → algoritmi genetici – AG – (*Genetic Algorithms*)
 - I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany) → strategii evolutive – SE – (*Evolution Strategies*)
 - J. Koza 1989 (Palo Alto, CA) → programare genetică – PG – (*Genetic Programming*)

Algoritmi evolutivi – aspecte teoretice

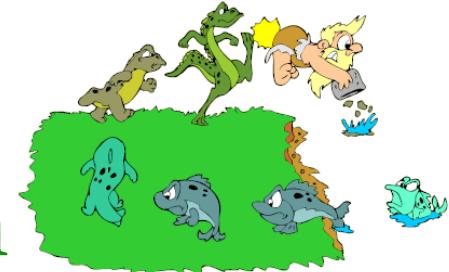
□ Metafora evolutivă

Evoluția naturală		Rezolvarea problemelor
Individ	↔	Soluție potențială
Populație	↔	Mulțime de soluții potențiale
Cromozom	↔	Codarea unei soluții potențiale
Genă	↔	Parte a codării
Fitness	↔	Calitate
Încrucișare și mutație	↔	Operatori de căutare
Mediu	↔	Problemă

Algoritmi evolutivi - algoritm



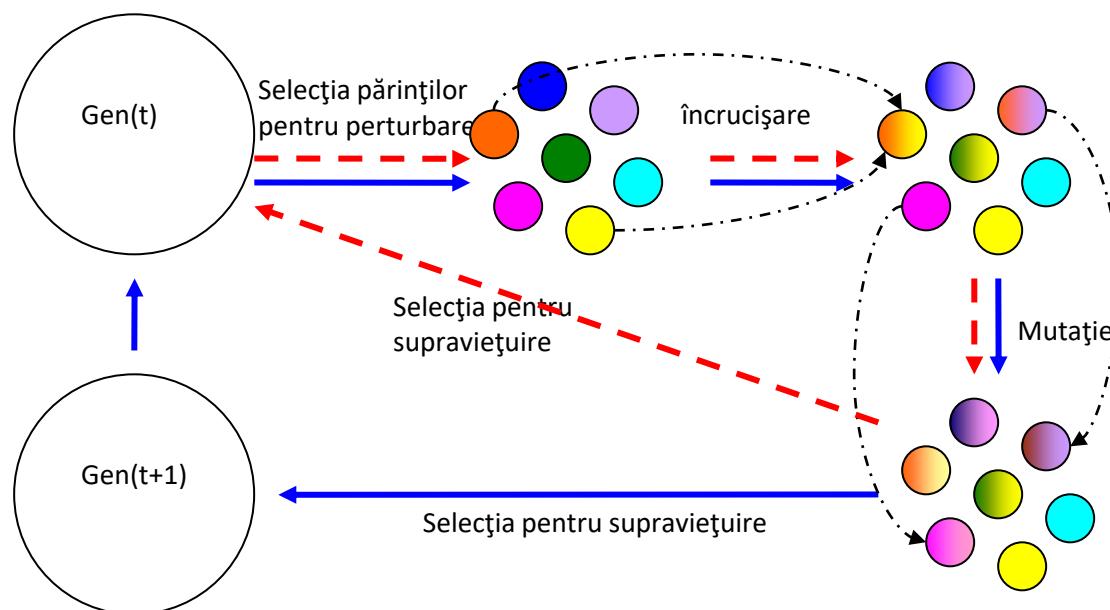
- Schema generală
- Proiectare



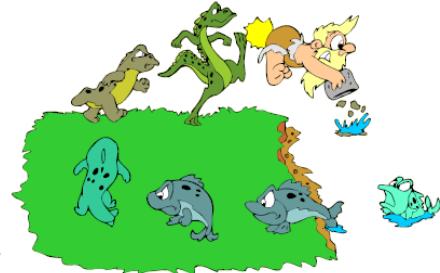
Algoritmi evolutivi – algoritm

□ Schema generală a unui AE

- Generațional →
- Steady-state →



Algoritmi evolutivi – algoritm



□ Proiectare

- Alegerea unei reprezentări a cromozomilor
- Alegerea unui model de populație
- Stabilirea unei funcții de evaluare
- Stabilirea operatorilor genetici
 - Selecție
 - Mutăție
 - Recombinare
- Stabilirea unui criteriu de stop

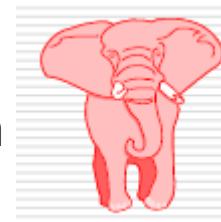
Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ 2 nivele de existență pentru o soluție candidat

■ Nivel exterior → fenotip

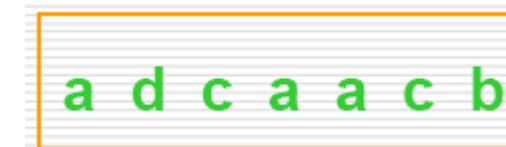
- Individ - obiectul original în contextul dat de problemă
- Aici are loc evaluarea unei potențiale soluții
- Furnică, rucsac, elefant, orașe, ...



■ Nivel interior → genotip

- Cromozom – codul asociat unui obiect
 - format din gene, poziționate în locuri (fixe) – loci – și având anumite valori – alele
- Aici are loc căutarea unei noi potențiale soluții
- Vector unidimensional (numeric, boolean, string), matrice,

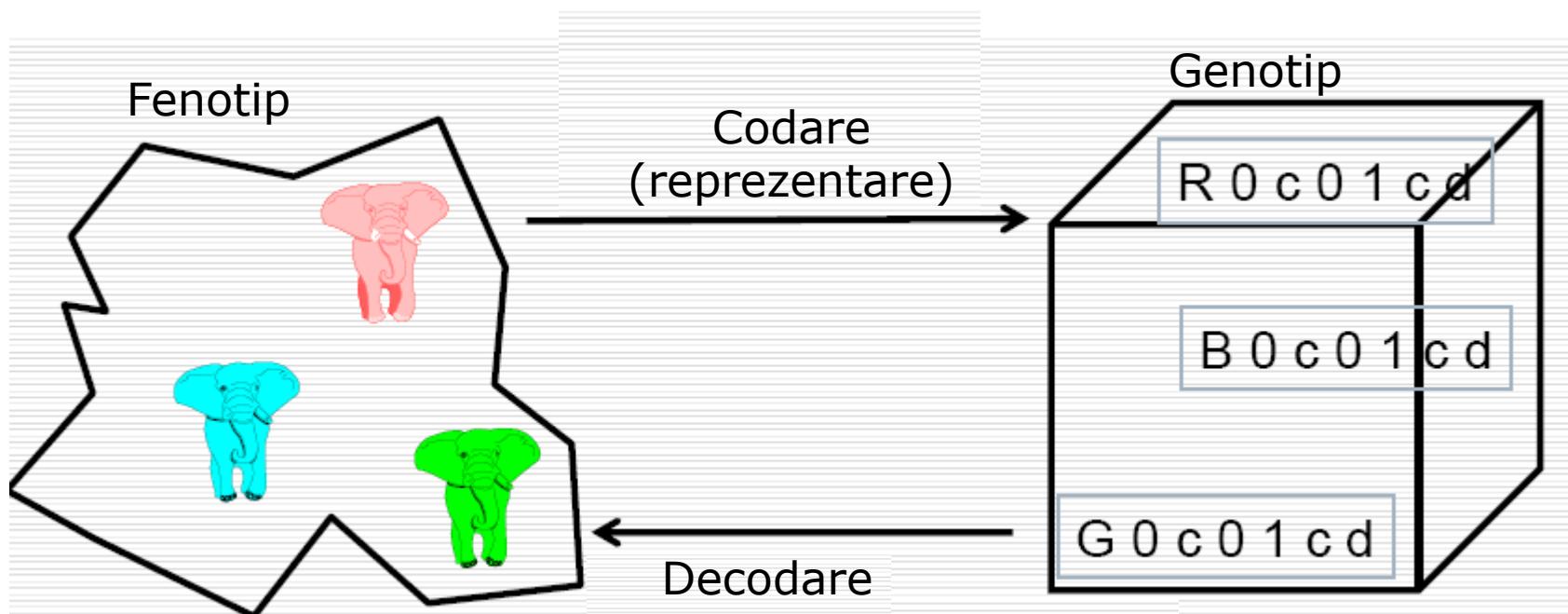
...



Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

- Reprezentarea trebuie să fie relevantă pentru:
 - problemă,
 - funcția de evaluare și
 - operatorii genetici



Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

Tipologia reprezentării cromozomilor

- Liniară
 - Discretă
 - Binară → problema rucsacului
 - Ne-binară
 - Întreagă
 - Oarecare → procesarea imaginilor
 - Permutări → problema comisului voiajor
 - Categorială → problema colorării hărților
 - Continuă (reală) → optimizări de funcții
 - Arborescentă → probleme de regresie

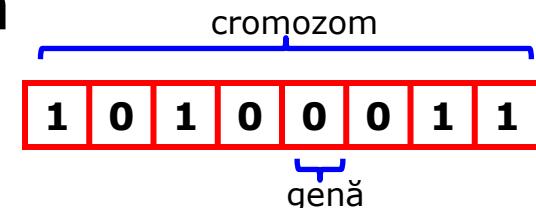
Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

- Genotip

- sir de biți



Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

- Genotip
 - sir de biți
- Fenotip
 - Elemente de tip Boolean
 - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului

Genotip
1 0 1 0 0 0 1 1

Fenotip
 $ob_1 + ob_3 + ob_7 + ob_8$

Au fost alese obiectele 1, 3, 7 și 8

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

- Genotip
 - şir de biţi
 - Fenotip
 - Elemente de tip Boolean
 - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului

- ## □ Numere întregi

1	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

Fenotip
= 163

$$1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = \\ 128 + 32 + 2 + 1 = 163$$

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

- Genotip
 - sir de biți
- Fenotip
 - Elemente de tip Boolean
 - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului

- Numere întregi

- Numere reale într-un anumit Interval (ex. [2.5, 20.5])

Genotip	Fenotip
1 0 1 0 0 0 1 1	= 13.9609

$x = 2.5 + \frac{163}{256}(20.5 - 2.5) = 13.9609$

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

Transformarea valorilor reale reprezentate pe biți

- ▣ Fie $z \in [x,y] \subseteq \mathcal{R}$ reprezentat ca $\{a_1, \dots, a_L\} \in \{0,1\}^L$
- ▣ Funcția $[x,y] \rightarrow \{0,1\}^L$ trebuie să fie inversabilă (un fenotip corespunde unui genotip)
- ▣ Funcția $\Gamma: \{0,1\}^L \rightarrow [x,y]$ definește reprezentarea
$$\Gamma(a_1, \dots, a_L) = x + \frac{y-x}{2^L - 1} \cdot \left(\sum_{j=0}^{L-1} a_{L-j} \cdot 2^j \right) \in [x, y]$$
- ▣ Observații
 - Se pot reprezenta doar 2^L valori
 - L indică precizia maximă a soluției
 - Pentru o precizie cât mai bună → cromozomi lungi → evoluție încetinită

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară întreagă oarecare
 - Genotip
 - sir de numere întregi dintr-un anumit interval
 - Fenotip
 - Utilitatea numerelor în problemă
 - Ex. Problema plății unei sume folosind diferite monezi
 - Genotip → sir de nr întregi de lungime egală cu numărul de monezi diferite, fiecare număr din intervalul $[0, \text{suma}/\text{valoarea monezii curente}]$
 - Fenotip → câte monezi din fiecare tip trebuie considerate

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară întreagă de tip permutare
 - Genotip
 - Permutare de n numere (n – numărul de gene)
 - Fenotip
 - Utilitatea permutării în problemă
 - Ex. Problema comisului voiajor
 - Genotip → permutare de n elemente
 - Fenotip → ordinea de vizitare a orașelor, știind că fiecărui oraș îi corespunde un număr din mulțimea $\{1, 2, \dots, n\}$

Algoritmi evolutivi – algoritm Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară categorială
 - Similară cu cea întreagă, dar în loc de numere se folosesc etichete
 - Genotip
 - sir de etichete dintr-o anumită mulțime
 - Fenotip
 - Interpretarea etichetelor
 - Ex. Problema colorării hărților
 - Genotip → sir de etichete (culori) de lungime egală cu numărul de țări, fiecare etichetă aparținând unei mulțimi de culori date
 - Fenotip → cu ce culoare trebuie hașurată fiecare hartă a unei țări

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ Reprezentare liniară continuă (reală)

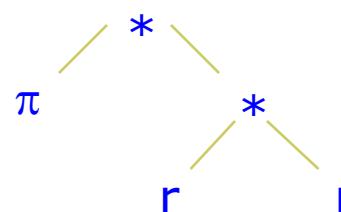
- Genotip
 - Sir de numere reale
- Fenotip
 - Utilitatea numerelor în problemă
- Ex. Problema optimizării funcțiilor $f: R^n \rightarrow R$
 - Genotip \rightarrow tuplu de numere reale $X=[x_1, x_2, \dots, x_n]$, $x_i \in R$
 - Fenotip \rightarrow valorile asociate argumentelor funcției f

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

- Reprezentare arborescentă
 - Genotip
 - Arbori care codează S-Expresii
 - Nodurile interne ale arborelui → funcții (F)
 - Matematice
 - Operatori aritmetici
 - Operatori de tip Boolean
 - Instrucțiuni
 - Într-un limbaj de programare
 - Alt tip de instrucțiuni
 - Frunzele arborelui → terminale (T)
 - Valori reale sau Booleene, constante sau variabile
 - Subprograme
 - Fenotip
 - Interpretarea S-expresiilor
 - Ex. Calculul ariei unui cerc

$$\pi * r^2$$



Algoritmi evolutivi – algoritm Proiectare – formarea unei populații

□ Populație – concept

■ Scop

- reține o colecție de soluții candidat
 - se permit repetiții
- este folosită în întregime în procesul de selecție pentru reproducere

■ Proprietăți

- dimensiune (de obicei) fixă μ
- diversitate
 - Nr de fitness-uri/fenotipuri/genotipuri diferite

■ Observații

- Reprezintă unitatea de bază care evoluează
 - populația întreagă evoluează, nu indivizi!!!!

Algoritmi evolutivi – algoritm Proiectare – formarea unei populații

- Populație – inițializare
 - Uniformă (dacă e posibil) în spațiul de căutare
 - Stringuri binare
 - generarea de 0 și 1 cu probabilitatea 0.5
 - Siruri de numere reale generate uniform (într-un anumit interval)
 - Permutări
 - generarea permutării identice și efectuarea unor schimbări

Algoritmi evolutivi – algoritm Proiectare – formarea unei populații

□ Populație – inițializare

- Uniformă (dacă e posibil) în spațiul de căutare
 - Arbori
 - Metoda *Full* – arbori compleți
 - Nodurile de la adâncimea $d < D_{\max}$ se inițializează aleator cu o funcție din setul de funcții F
 - Nodurile de la adâncimea $d = D_{\max}$ se inițializează aleator cu un terminal din setul de terminale T
 - Metoda *Grow* – arbori incompleți
 - Nodurile de la adâncimea $d < D_{\max}$ se inițializează aleator cu un element din $F \cup T$
 - Nodurile de la adâncimea $d = D_{\max}$ se inițializează aleator cu un terminal din setul de terminale T
 - Metoda *Ramped half and half*
 - $\frac{1}{2}$ din populație se creează cu metoda *Full*
 - $\frac{1}{2}$ din populație se creează cu metoda *Grow*
 - Folosind diferite adâncimi

Algoritmi evolutivi – algoritm Proiectare – formarea unei populații

- Modele de populații – algoritm evolutiv:
 - Generațional
 - În fiecare generație se crează μ descendenți
 - Fiecare individ supraviețuiește o singură generație
 - Multimea părinților este înlocuită în întregime cu multimea descendenților
 - Steady-state
 - În fiecare generație se obține un singur descendant
 - Un singur părinte (cel mai slab) este înlocuit cu descendantul obținut
- Discrepanța între generații (*Generation Gap*)
 - Proportia populației înlocuite
 - $1 = \mu/\mu$, pentru modelul generational
 - $1/\mu$, pentru modelul steady-state

Algoritmi evolutivi – algoritm Proiectare – funcția de evaluare

- Scop
 - Reflectă condițiile la care trebuie să se adapteze populația
 - Funcție de calitate sau funcție obiectiv
 - Asociază o valoare fiecărei soluții candidat
 - Consecințe asupra selecției → cu cât sunt mai multe valori diferite, cu atât e mai bine
- Proprietăți
 - Etapa cea mai costisitoare
 - Nu se re-evaluează indivizii nemodificați
- Tipologie:
 - După nr de obiective urmărite:
 - Uni-obiectiv
 - Multi-obiectiv → fronturi Pareto
 - După direcția optimizării
 - De maximizat
 - De minimizat
 - După gradul de exactitate
 - Exactă
 - Euristică

Algoritmi evolutivi – algoritm Proiectare – funcția de evaluare

- Exemple
 - Problema rucsacului
 - reprezentare → liniară discretă binară
 - fitness → $\text{abs}(\text{greutatea rucsacului} - \text{greutatea obiectelor alese}) \rightarrow \text{minimizare}$
 - Problema plății unei sume folosind diferite monezi
 - reprezentare → liniară discretă întreagă
 - fitness → $\text{abs}(\text{suma de plată} - \text{suma monezilor selectate}) \rightarrow \text{minimizare}$
 - Problema comisului voiajor
 - reprezentare → liniară discretă întreagă sub formă de permutare
 - fitness → costul drumului parcurs → minimizare
 - Problema optimizării funcțiilor
 - Reprezentare → liniară continuă reală
 - fitness → valoarea funcției → minimizare/maximizare
 - Calculul ariei unui cerc
 - reprezentare → arborescentă
 - fitness → suma pătratelor erorilor (diferențelor între valoarea reală și cea calculată pe un set de exemple) → minimizare

Algoritmi evolutivi – algoritm Proiectare – selectia



- Scop:
 - acordă şanse de reproducere/supravieţuire mai mari indivizilor mai buni
 - şi indivizii mai slabii trebuie să aibă şansa să se reproducă/supravieţuiască pentru că pot conţine material genetic util
 - direcţionează populaţia spre îmbunătăţirea calităţii

- Proprietăţi
 - lucrează la nivel de populaţie
 - se bazează doar pe fitnessul indivizilor (este independentă de reprezentare)
 - ajută la evadarea din optimele locale datorită naturii sale stocastice

Algoritmi evolutivi – algoritm Proiectare – selecția



□ Tipologie

- în funcție de scop:
 - ▣ Selecția părinților (din generația curentă) pentru reproducere
 - ▣ Selecția supraviețuitorilor (din părinți și descendenți) pentru generația următoare
- în funcție de modul de decizie al câștigătorului
 - ▣ Deterministă – cel mai bun câștigă
 - ▣ Stocastică – cel mai bun are cele mari şanse să câștige
- în funcție de mecanism
 - ▣ Selecția pentru reproducere
 - Selecție proporțională (bazată pe fitness)
 - Selecție bazată pe ranguri
 - Selecție prin turnir ----> Bazată pe o parte din populație
 - ▣ Selecția pentru supraviețuire
 - Bazată pe vârstă
 - Bazată pe calitate (fitness)

Algoritmi evolutivi – algoritm



Proiectare – selectia pt. reproducere

□ Selectie proportională (bazată pe fitness) – SP

□ Ideea de bază

- Algoritmul ruletei la nivelul întregii populații
- Estimarea numărului de copii ale unui individ

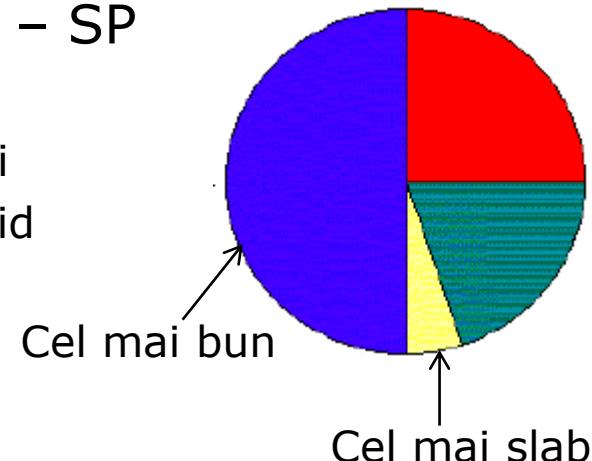
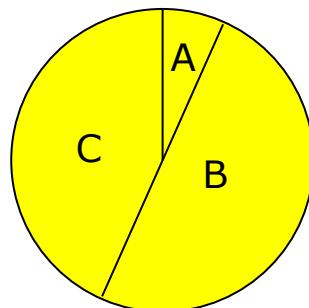
$$E(n_i) = \mu \frac{f(i)}{\langle f \rangle}, \text{ unde:}$$

- μ = dimensiunea populației,
- $f(i)$ = fitnessul individului i ,
- $\langle f \rangle$ = fitnessul mediu al populației

□ Indivizii mai buni

- au alocat mai mult spațiu în ruletă
- au sanse mai mari să fie selectați

□ Ex. O populație cu $\mu = 3$ indivizi



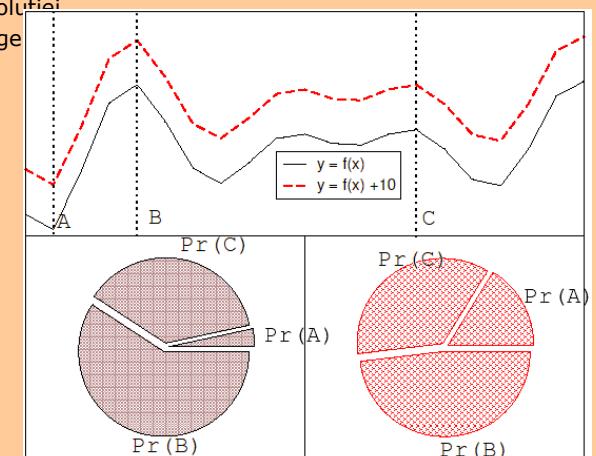
	$f(i)$	$P_{selSP}(i)$
A	1	$1/10=0.1$
B	5	$5/10=0.5$
C	4	$4/10=0.4$
Suma	10	1

Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



Selectie proporțională (bazată pe fitness)

- Avantaje
 - Algoritm simplu
- Dezavantaje
 - Convergența prematură
 - cromozomii foarte buni tind să domine populația
 - Presiune de selecție foarte mică atunci când fintessurile indivizilor sunt foarte apropiate (la sfârșitul rulării)
 - Susceptibilă de traspoziția funcției
 - Rezultatele reale ale unei astfel de selecții diferă de distribuția probabilistică teoretică
 - Lucrează cu întreaga populație
- Soluții
 - scalarea fitnessului
 - Windowing
 - $f'(i) = f(i) - \beta^t$, unde β este un parametru care depinde de istoria recentă a evoluției
 - ex. β este fitnessul celui mai slab individ din populația curentă (a t -a generație)
 - Scalare de tip sigma (de tip Goldberg)
 - $f'(i) = \max\{f(i) - (\langle f \rangle - c * \sigma_f), 0.0\}$, unde:
 - c este o constantă (de obicei 2)
 - $\langle f \rangle$ - fitnessul mediu al populației
 - σ_f - deviația standard a fitnessului populației
 - Scalare prin normalizare
 - Se începe cu fitnessurile absolute (inițiale)
 - Se standardizează astfel încât se ajustează fitnessurile a.î.:
 - ele să aparțină $[0,1]$
 - cel mai bun fitness să fie cel mai mic (egal cu 0)
 - suma lor să fie 1
 - alt mecanism de selecție



Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



□ Selectia bazata pe ranguri – SR

■ Ideea de bază

- Se ordonează întreaga populație pe baza fitnessului
 - Crește puțin complexitatea algoritmului, dar se poate neglija această creștere comparativ cu timpul necesar evaluării unui individ
- Se acordă ranguri fiecărui individ
- Se calculează probabilitățile de selecție pe baza rangurilor
 - Cel mai slab individ are rangul 1
 - Cel mai bun individ are rangul μ
- Încearcă să rezolve problemele selecției proporționale prin folosirea fitnessurilor relative (în locul celor absolute)

Algoritmi evolutivi – algoritm



Proiectare – selecția pt. reproducere

□ Selecția bazată pe ranguri – SR

■ Modalități de acordare a rangurilor

□ Liniară (RL) $P_{lin_rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$

- s – presiunea de selecție
 - măsoară avantajele celui mai bun individ
 - $1.0 < s \leq 2.0$
 - în algoritmul genetic generațional s este numărul de copii ai unui individ
- Ex. pentru o populație cu $\mu = 3$ indivizi

	f(i)	P _{selSP} (i)	Rang	P _{selRL} (i) pt. s=2	P _{selRL} (i) pt. s=1
A	1	1/10=0.1	1	0.33	0.33
B	5	5/10=0.5	3	1.00	0.33
C	4	4/10=0.4	2	0.67	0.33
Suma	10	1			

□ Exponențială (RE) $P_{exp_rank}(i) = \frac{1-e^{-i}}{c}$

- Cel mai bun individ poate avea mai mult de 2 copii
- c – factor de normalizare
 - depinde de dimensiunea populației (μ)
 - trebuie ales a.î. suma probabilităților de selecție să fie 1

Algoritmi evolutivi – algoritm



Proiectare – selecția pt. reproducere

□ Selecția bazată pe ranguri – SR

- Avantaje

- Păstrează presiunea de selecție constantă

- Dezavantaje

- Lucrează cu întreaga populație

- Soluții

- Alt mecanism de selecție

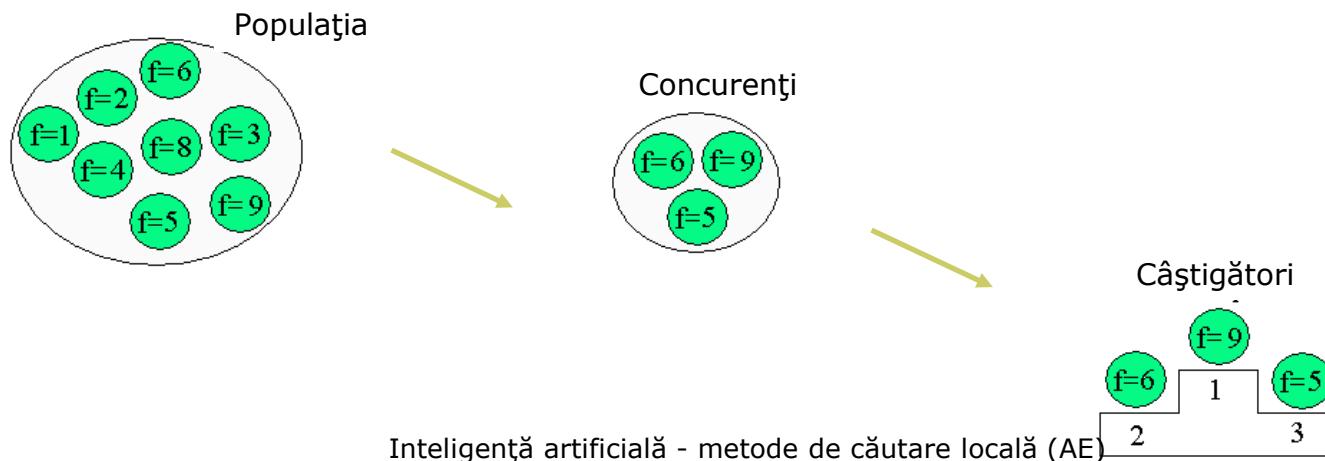
Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



□ Selecția prin turnir

■ Ideea de bază

- Se aleg aleator k indivizi \rightarrow eșantion de k indivizi (k – mărimea turnirului)
- Se selectează cel mai bun individ dintre cei aleși anterior
- Probabilitatea alegерii unui individ în eșantion depinde de
 - Rangul individului
 - Dimensiunea eșantionului (k)
 - Cu cât k este mai mare, cu atât crește și presiunea de selecție
 - Modul în care se face alegerea – dacă se realizează cu înlocuire (model steady-state) sau nu
 - Alegerea fără înlocuire crește presiunea de selecție
 - Pt $k = 2$ timpul necesar ca cel mai bun individ să domine populația este același cu cel de la selecția pe bază de ranguri liniare cu $s = 2 * p$, p – probabilitatea alegерii celui mai bun individ din populație



Algoritmi evolutivi – algoritm Proiectare – selecția pt. reproducere



□ Selectă prin turnir

■ Avantaje

- ▣ Nu implică lucrul cu întregă populație
- ▣ Ușor de implementat
- ▣ Ușor de controlat presiunea de selecție prin intermediul parametrului k

■ Dezavantaje

- ▣ Rezultatele reale ale unei astfel de selecții diferă de distribuția probabilistică teoretică (similar selecției prin mecanismul ruletei)

Algoritmi evolutivi

Proiectare – selecția



- Selecția pentru supraviețuire (înlocuire)
 - Pe baza vârstei
 - ▣ eliminarea celor mai “bătrâni” indivizi
 - Pe baza calității (fitness-ului)
 - ▣ selecției proporțională
 - ▣ selecție bazată pe ranguri
 - ▣ selecție prin turnir
 - ▣ elitism
 - Păstrarea celor mai buni indivizi de la o generație la alta (dacă descendenții sunt mai slabii ca părinții se păstrează părinții)
 - ▣ GENITOR (înlocuirea celui mai slab individ)
 - Eliminarea celor mai slabii λ indivizi

Algoritmi evolutivi - algoritm Proiectare – operatori de variație



- Scop:
 - Generarea unor soluții potențiale noi
- Proprietăți
 - lucrează la nivel de individ
 - se bazează doar pe reprezentarea indivizilor (independent de fitness)
 - Aiută la explorarea și exploatarea spațiului de căutare
 - Trebuie să producă indivizi valizi
- Tipologie
 - În funcție de aritate
 - Aritate 1 → operatori de mutație
 - Aritate > 1 → operatori de recombinare/încrucișare

Algoritmi evolutivi – algoritm Proiectare – mutația

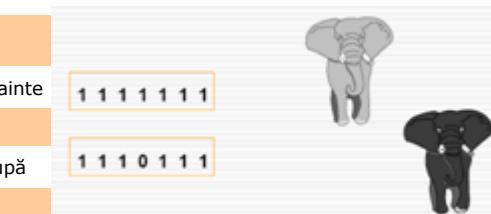


□ Scop

- Reintroducerea în populație a materialului genetic pierdut
- Operator unar de căutare (spațiul continuu)
- Introducerea diversității în populație (în spațiul discret – binar)

□ Proprietăți

- Acționează la nivel de genotip
- Bazată pe elemente aleatoare
- Responsabilă cu explorarea unor noi regiuni promițătoare ale spațiului de căutare
- Este responsabilă de evadarea din optimele locale
- Trebuie să producă mici schimbări stocastice ale individului
- Mărimea mutației trebuie să fie controlabilă
- Se produce cu o anumită probabilitate (p_m) la nivelul fiecărei gene a unui cromozom



Algoritmi evolutivi – algoritm Proiectare – mutația



□ Tipologie

- Reprezentare binară
 - Mutație tare - bit-flipping
 - Mutație slabă
- Reprezentare întreagă
 - Random resetting
 - Creep mutation
- Reprezentare permutare
 - Mutație prin inserție
 - Mutație prin interchimbare
 - Mutație prin inversare
 - Mutație prin amestec
 - Mutație k-opt
- Reprezentare reală
 - Mutație uniformă
 - Mutație neuniformă
 - Mutație Gaussiană
 - Mutație Cauchy
 - Mutație Laplace
- Reprezentare arborescentă → într-un curs viitor
 - Mutație grow
 - mutație shrink
 - Mutație switch
 - Mutație cycle
 - Mutație tip Koza
 - Mutație pentru terminalele numerice

Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. binară)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{0,1\}$, pt. $i=1,2,\dots,L$
- Mutație tare – *bit flipping*
 - Ideea de bază
 - Schimbarea cu probabilitatea p_m (rată de mutație) a unor gene în complementul lor
 - $1 \rightarrow 0$
 - $0 \rightarrow 1$
 - Ex. Un cromozom cu $L = 8$ gene, $p_m = 0.1$



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. binară)



- ❑ Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{0,1\}$, pt. $i=1,2,\dots,L$
- ❑ Mutație slabă
 - Ideea de bază
 - ❑ Schimbarea cu probabilitatea p_m (rată de mutație) a unor gene în 0 sau 1
 - $1 \rightarrow 0/1$
 - $0 \rightarrow 1/0$
 - ❑ Ex. Un cromozom cu $L = 8$ gene, $p_m = 0.1$



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. întreagă)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{val_1, val_2, \dots, val_k\}$, pt. $i=1,2,\dots,L$
- Mutație *random resetting*
 - Ideea de bază
 - Valoarea unei gene este schimbată (cu probabilitatea p_m) într-o altă valoare (din setul de valori posibile)



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. întreagă)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_k\}$, pt. $i=1,2,\dots,L$
- Mutație *creep*
 - Ideea de bază
 - Valoarea unei gene este schimbată (cu probabilitatea p_m) prin adăugarea unei valori (pozitivă sau negativă)
 - valoarea → face parte dintr-o distribuție simetrică față de zero
 - modificarea produsă este fină (mică)



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. permutare)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ cu $g_i \neq g_i'$ pentru orice $i \neq i'$ devine $c' = (g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i = 1, 2, \dots, L$ a.î. $g_i' \neq g_i$ pentru orice $i \neq i'$.
- Mutație prin interschimbare (*swap mutation*)
 - Ideea de bază
 - Se aleg aleator 2 gene și se interschimbă valorile lor



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. permutare)



- ❑ Un cromozom $c = (g_1, g_2, \dots, g_L)$ cu $g_i \neq g_j$ pentru orice $i \neq j$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i=1,2,\dots,L$ a.î. $g'_i \neq g'_j$ pentru orice $i \neq j$.
- ❑ Mutație prin inserție
 - Ideea de bază
 - ❑ Se aleg 2 gene oarecare g_i și g_j cu $j > i$
 - ❑ Se inserează g_j după g_i a.î. $g'_i = g_i, g'_{i+1} = g_j, g'_{k+2} = g_{k+1},$ pentru $k=i, i+1, i+2, \dots$



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. permutare)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ cu $g_i \neq g_i'$ pentru orice $i \neq i'$ devine $c' = (g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i = 1, 2, \dots, L$ a.î. $g_i \neq g_i'$ pentru orice $i \neq i'$.
- Mutație prin inversare
 - Ideea de bază
 - Se aleg aleator 2 gene și se inversează ordinea genelor situate între ele (substringul dintre gene)



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. permutare)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ cu $g_i \neq g_i'$ pentru orice $i \neq i'$ devine $c' = (g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i = 1, 2, \dots, L$ a.î. $g_i' \neq g_i$ pentru orice $i \neq i'$.
- Mutație prin amestec (*scramble mutation*)
 - Ideea de bază
 - Se alege aleator un subșir (continuu sau discontinuu) de gene și se rearanjează acele gene



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. permutare)

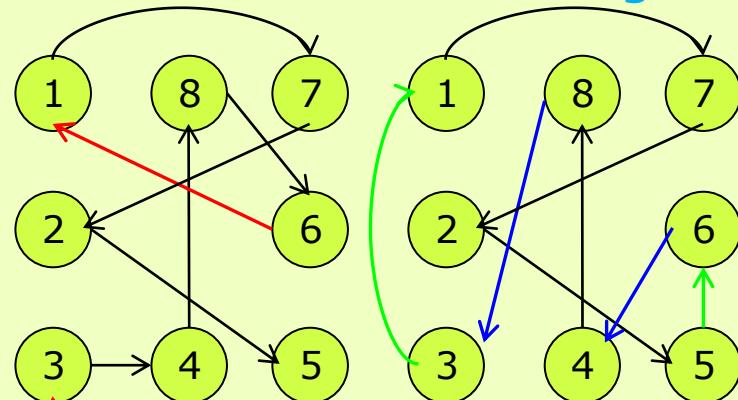
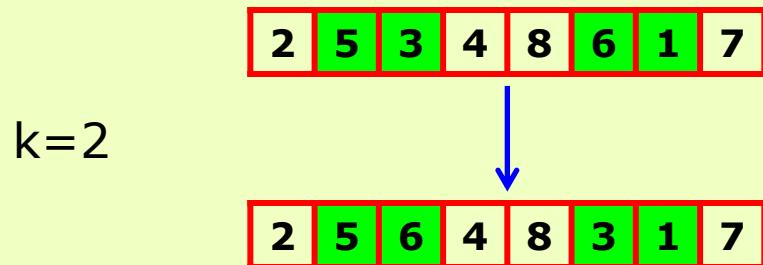


- Un cromozom $c = (g_1, g_2, \dots, g_L)$ cu $g_i \neq g_i'$ pentru orice $i \neq i'$ devine $c' = (g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{\text{val}_1, \text{val}_2, \dots, \text{val}_L\}$, pt. $i=1,2,\dots,L$ a.î. $g_i' \neq g_i$ pentru orice $i \neq i'$.

□ Mutăție k-opt

■ Ideea de bază

- Se aleg 2 substringuri disjuncte și de lungime k
- Se interchimbă 2 elemente ale acestor substringuri de gene



Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. reală)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Mutație uniformă
 - Ideea de bază
 - g'_i este schimbată cu probabilitatea p_m la o valoare aleasă aleator uniform din $[LI_i, LS_i]$

Algoritmi evolutivi – algoritm Proiectare – mutația (reprez. reală)



- Un cromozom $c = (g_1, g_2, \dots, g_L)$ devine $c' = (g'_1, g'_2, \dots, g'_L)$, unde $g_i, g'_i \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Mutație neuniformă
 - Ideea de bază
 - Valoarea unei gene este schimbată (cu probabilitatea p_m) prin adăugarea unei valori (pozitivă sau negativă)
 - valoarea → face parte dintr-o distribuție
 - $N(\mu, \sigma)$ (Gaussiană) cu $\mu = 0$
 - Cauchy (x_0, γ)
 - Laplace (μ, b)
 - și readusă la $[LI_i, LS_i]$ (dacă este necesar) – *clamping*

Algoritmi evolutivi – algoritm Proiectare - recombinarea

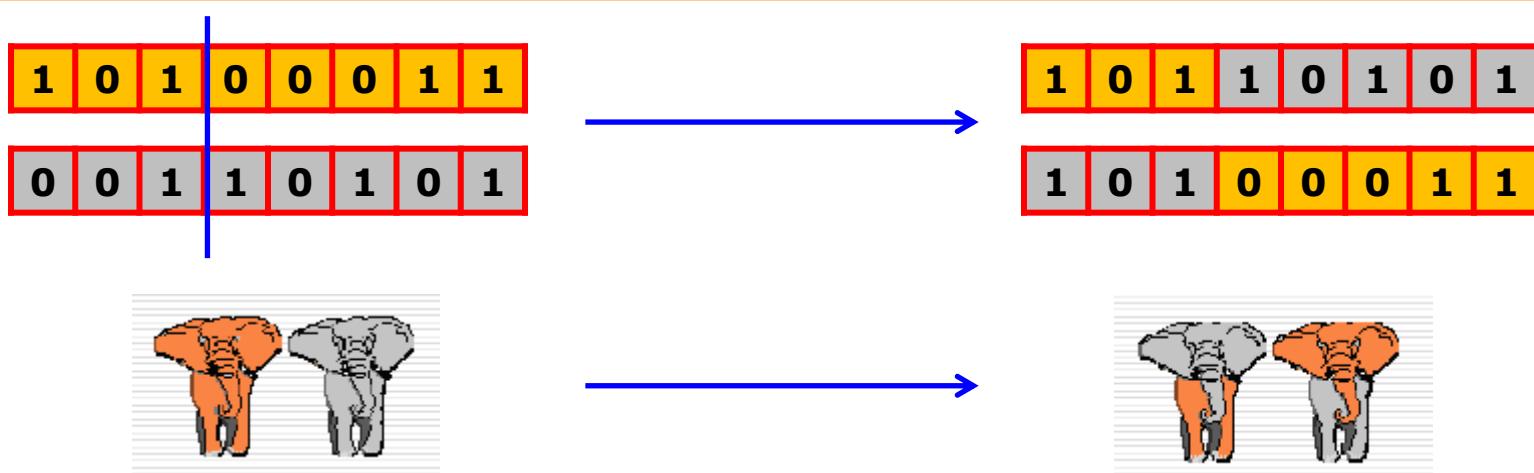


□ Scop

- Amestecarea informațiilor preluate din părinți

□ Proprietăți

- Descendentul trebuie să moștenească ceva de la fiecare dintre părinți
 - Alegerea informațiilor care se amestecă este aleatoare
- Operator de exploatare probabilistică (p_c) a spațiilor deja descoperite
- Descendenții pot să fie mai buni, la fel de buni sau mai slabi decât părinții lor
- Efectele sale se reduc pe măsură ce căutarea converge



Algoritmi evolutivi – algoritm Proiectare - recombinarea



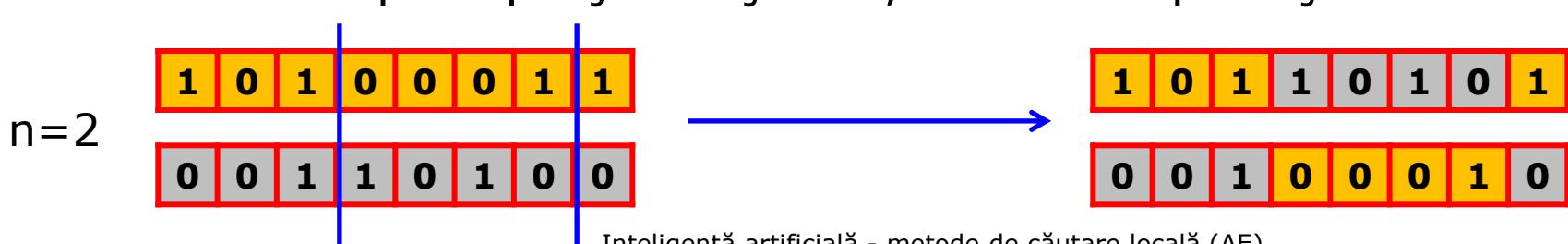
- Tipologie – în funcție de reprezentarea indivizilor
 - Reprezentare binară și întreagă
 - Cu puncte de tăietură
 - Uniformă
 - Reprezentare cu permutări
 - Încrucișare prin ordonare (versiunea 1 și versiunea 2)
 - Încrucișare transformată parțial (Partially Mapped Crossover)
 - Încrucișare ciclică
 - Încrucișare bazată pe legături (muchii)
 - Reprezentare reală
 - Discretă
 - Intermediară (aritmetică)
 - Aritmetică singulară
 - Aritmetică simplă
 - Aritmetică completă
 - Geometrică
 - Încrucișare amestecată
 - Încrucișare binară simulată
 - Reprezentare cu arbori
 - Încrucișare de sub-arbori → într-un curs viitor



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. binară și întreagă)

- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in \{0,1\} / \{\text{val}_1, \text{val}_2, \dots, \text{val}_k\}$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare cu n puncte de tăietură
 - Ideea de bază
 - ❑ Se aleg n puncte de tăietură ($n < L$)
 - ❑ Se taie cromozomii părinți prin aceste puncte
 - ❑ Se lipesc părțile obținute, alternând părinții





Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. binară și întreagă)

□ Încrucișare cu n puncte de tăietură

■ Proprietăți

- Media valorilor codate de părinți = media valorilor codate de descendenți
 - Ex. Reprezentarea binară pe 4 biți a numerelor întregi – XO cu $n = 1$ după bitul 2
 - $p_1 = (1,0,1,0)$, $p_2 = (1,1,0,1)$
 - $d_1 = (1,0, 0,1)$, $d_2 = (1,1,1,0)$
 - $\text{val}(p_1) = 10$, $\text{val}(p_2) = 13 \rightarrow (\text{val}(p_1) + \text{val}(p_2))/2 = 23/2=11.5$
 - $\text{val}(d_1) = 9$, $\text{val}(d_2) = 14 \rightarrow (\text{val}(d_1) + \text{val}(d_2))/2 = 23/2=11.5$
 - Ex. Reprezentare binară pe 4 biți pentru problema rucsacului de capacitate $K = 10$ cu 4 obiecte de greutate și valoare $((2,7), (1,8), (3,1), (2,3))$
 - $p_1 = (1,0,1,0)$, $p_2 = (1,1,0,1)$
 - $d_1 = (1,0, 0,1)$, $d_2 = (1,1,1,0)$
 - $\text{val}(p_1) = 8$, $\text{val}(p_2) = 18 \rightarrow (\text{val}(p_1) + \text{val}(p_2))/2 = 26/2=13$
 - $\text{val}(d_1) = 10$, $\text{val}(d_2) = 16 \rightarrow (\text{val}(d_1) + \text{val}(d_2))/2 = 26/2=13$
 - Probabilitatea apariției unui factor de răspândire $\beta \approx 1$ este mai mare decât probabilitatea oricărui alt factor

$$\beta = \left| \frac{\text{val}(d_1) - \text{val}(d_2)}{\text{val}(p_1) - \text{val}(p_2)} \right|$$

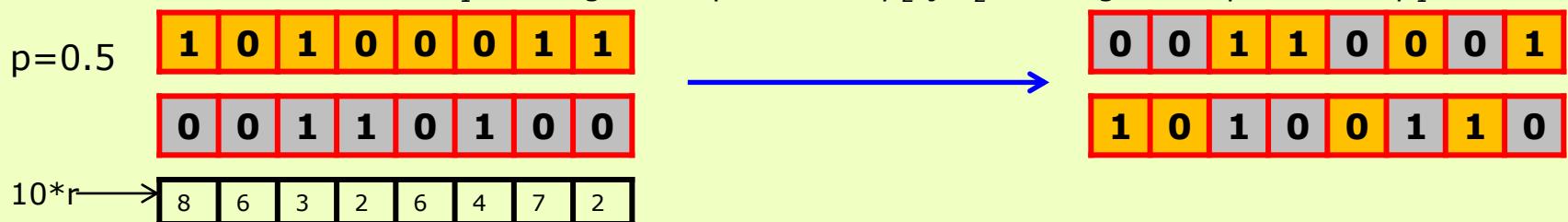
- Încrucișare prin contracție $\beta < 1$
 - Valorile descendenților se află între valorile părinților
- Încrucișare prin extensie $\beta > 1$
 - Valorile părinților se află între valorile descendenților
- Încrucișare staționară $\beta = 1$
 - Valorile descendenților coincid cu valorile părinților



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. binară și întreagă)

- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in \{0,1\} / \{\text{val}_1, \text{val}_2, \dots, \text{val}_k\}$, pt. $i=1,2,\dots,L$
- Încrucișare uniformă
 - Ideea de bază
 - Fiecare genă a unui descendent provine dintr-un părinte ales aleator și uniform:
 - Pentru fiecare genă în parte se generează un număr aleator r care respectă legea uniformă
 - Dacă numărul generat $r <$ probabilitatea p (de obicei $p=0.5$), c_1 va lua gena respectivă din p_1 și c_2 va lua gena respectivă din p_2 ,
 - Altfel c_1 va lua gena respectivă din p_2 și c_2 va lua gena respectivă din p_1



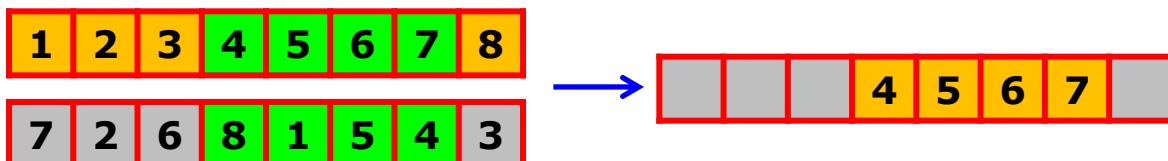
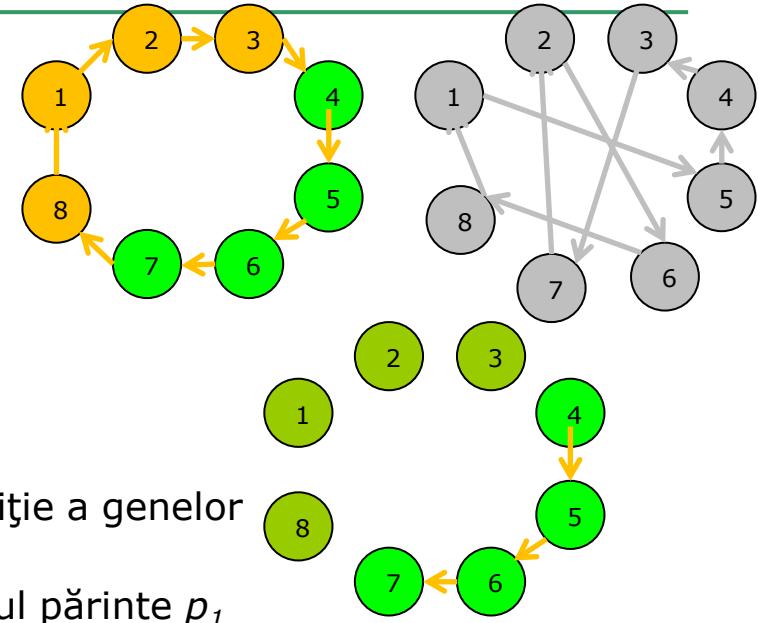


Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$

□ Încrucișare ordonată

- Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)



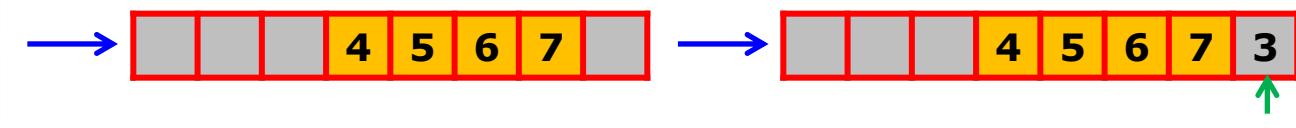
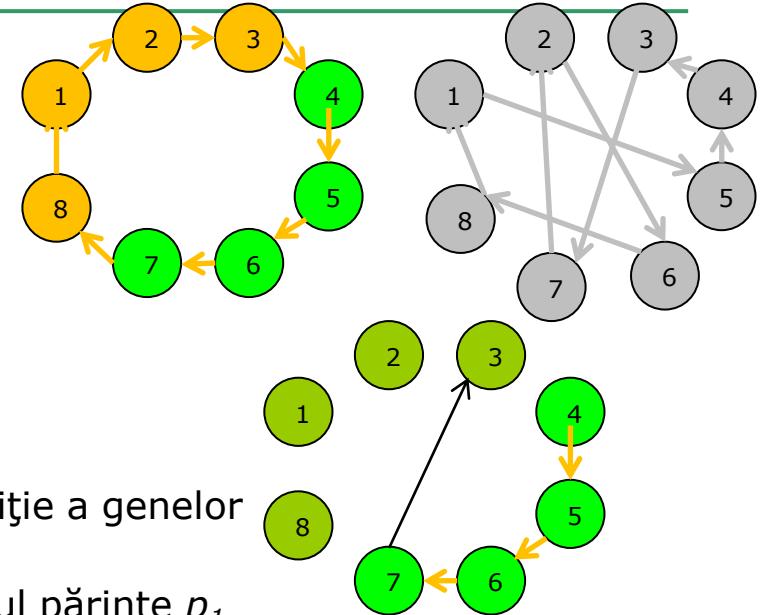


Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$

□ Încrucișare ordonată

- Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



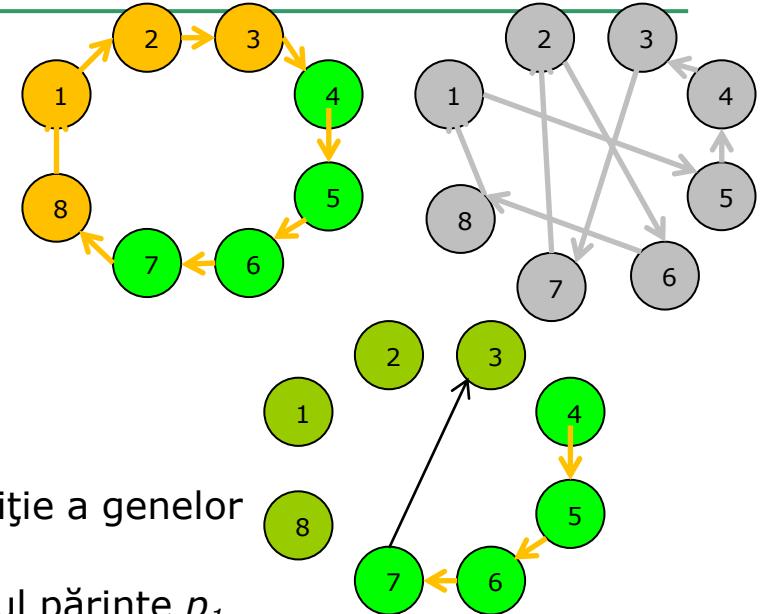


Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$

□ Încrușire ordonată

- Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



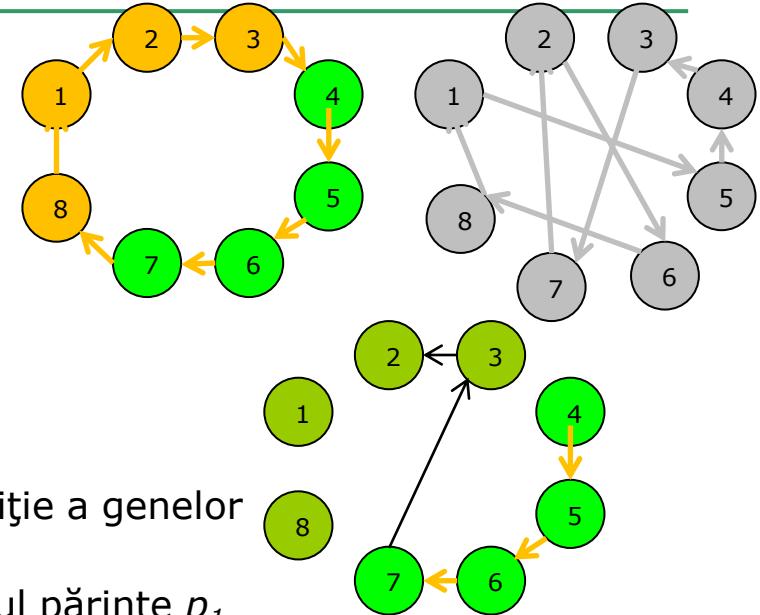


Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$

□ Încrucișare ordonată

- Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



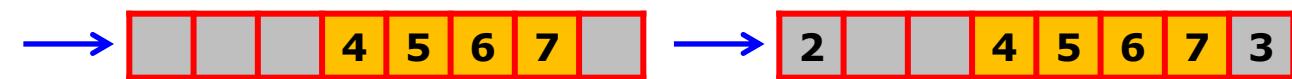
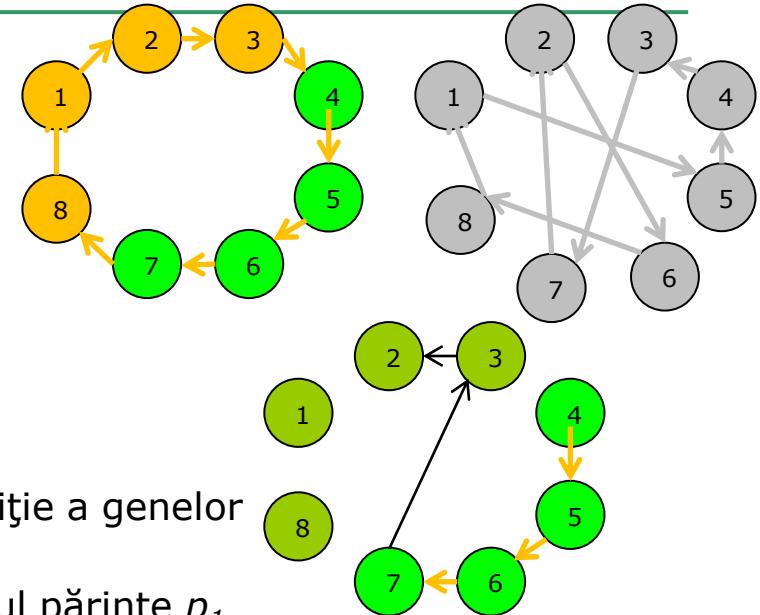


Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$

□ Încrucișare ordonată

- Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



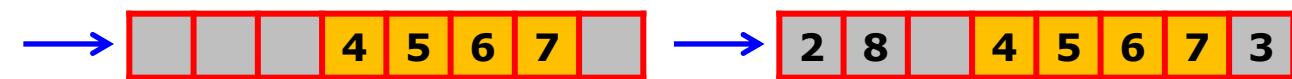
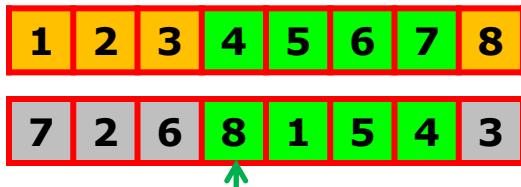
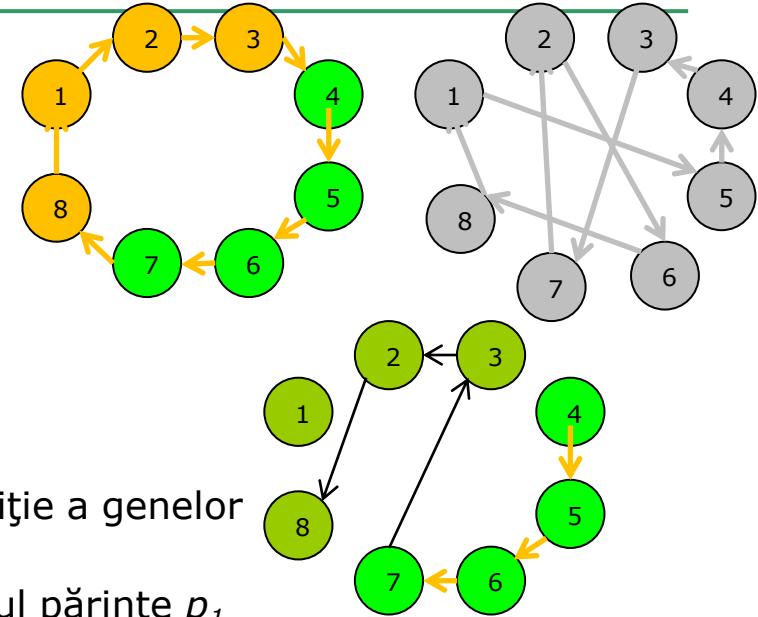


Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$

□ Încrucișare ordonată

- Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



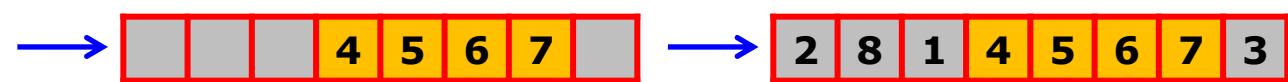
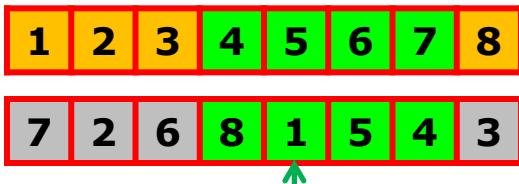
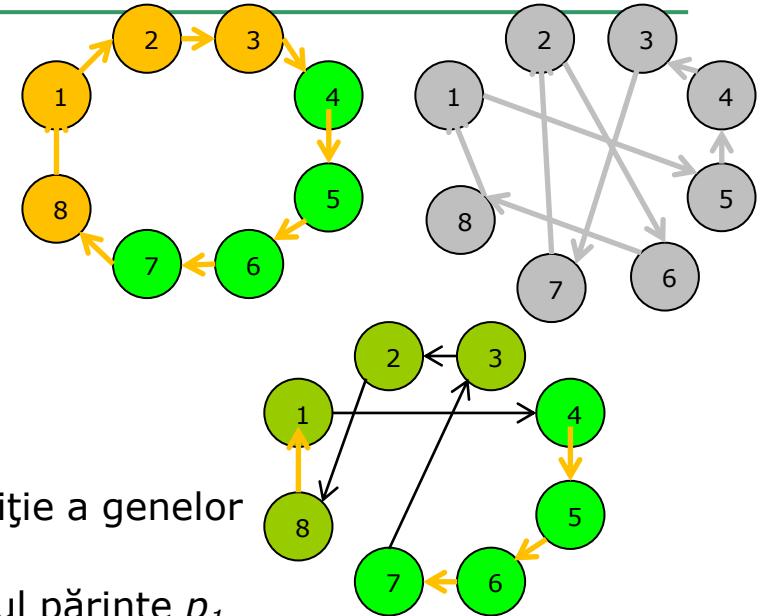


Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$

□ Încrucișare ordonată

- Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



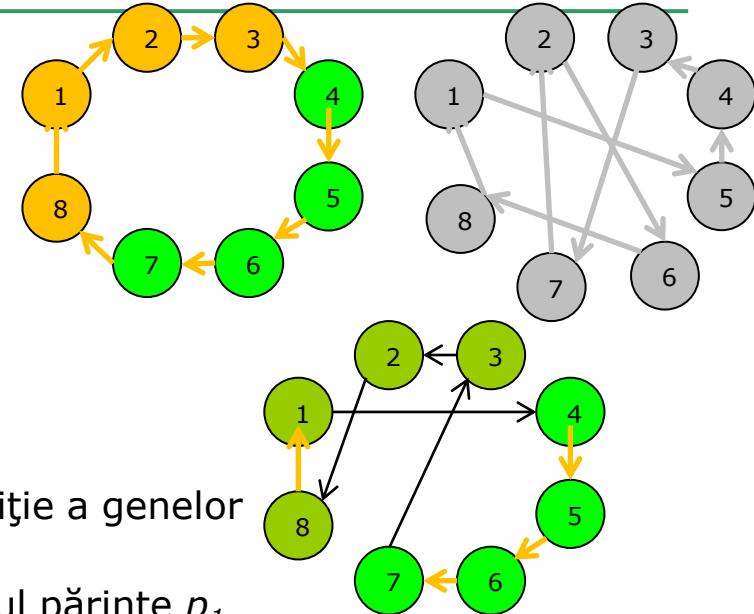


Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$

□ Încrușare ordonată

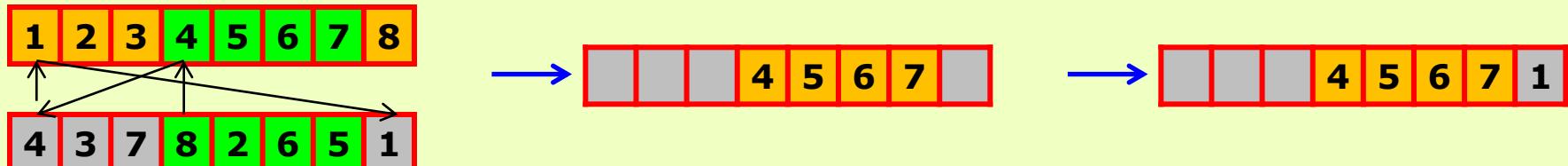
- Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)
 - Se reia procedeul pentru al doilea descendent d_2 .





Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

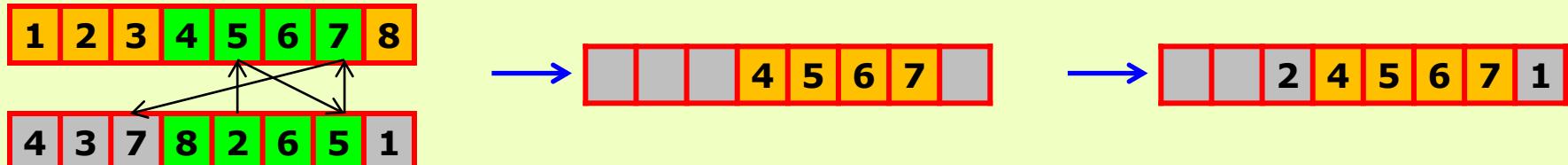
- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare parțial transformată (*partially mapped XO*)
 - Ideea de bază
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 și se determină care element j a fost copiat în locul lui din p_1
 - Se plasează i în d_1 în poziția ocupată de j în p_2 (dacă locul este liber)
 - Dacă locul ocupat de j în p_2 a fost deja completat în d_1 cu elementul k , i se pune în locul ocupat de k în p_2
 - Restul elementelor se copiază din p_2 în d_1
 - Pentru descendentul d_2 se procedează similar, dar inversând părinții





Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare parțial transformată (*partially mapped XO*)
 - Ideea de bază
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 și se determină care element j a fost copiat în locul lui din p_1
 - Se plasează i în d_1 în poziția ocupată de j în p_2 (dacă locul este liber)
 - Dacă locul ocupat de j în p_2 a fost deja completat în d_1 cu elementul k , i se pune în locul ocupat de k în p_2
 - Restul elementelor se copiază din p_2 în d_1
 - Pentru descendentul d_2 se procedează similar, dar inversând părinții





Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

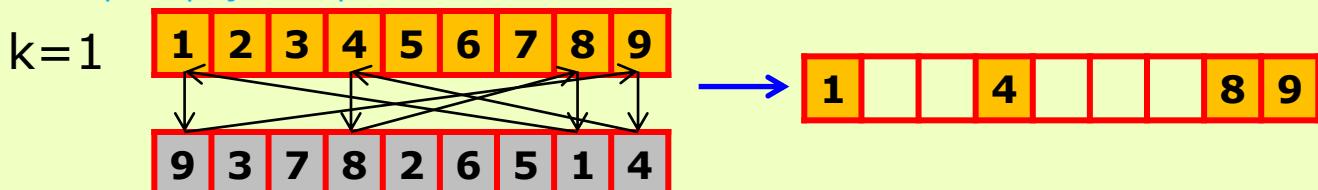
- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare parțială transformată
 - Ideea de bază
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 și se determină care element j a fost copiat în locul lui i din p_1
 - Se plasează i în d_1 în poziția ocupată de j în p_2 (dacă locul este liber)
 - Dacă locul ocupat de j în p_2 a fost deja completat în d_1 cu elementul k , i se pune în locul ocupat de k în p_2
 - Restul elementelor se copiază din p_2 în d_1
 - Pentru descendentul d_2 se procedează similar, dar inversând părintii





Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

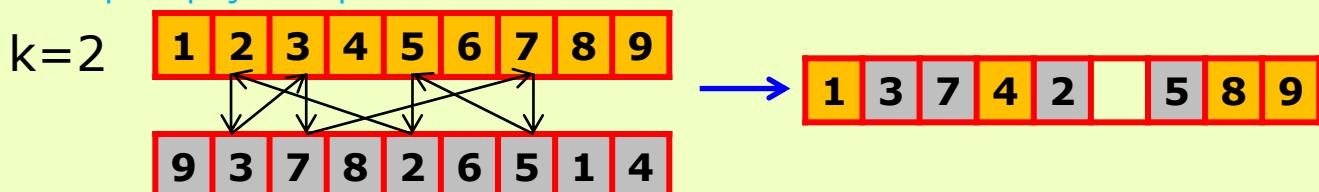
- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare ciclică
 - Ideea de bază
 1. inițial $k = 1$
 2. Se formează un ciclu
 - Se adaugă în ciclu gena de pe poziția k din p_1 (g_k^1)
 - Se consideră gena de pe poziția k din p_2 (g_k^2)
 - Se alege gena din p_1 cu valoarea egală cu g_k^2 (g_r^1) și se include în ciclu
 - Se consideră gena de pe poziția r din p_2 (g_r^2)
 - Se repetă pașii anteriori până când se ajunge la gena de pe poziția k din p_1
 3. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_1)
 4. Se incrementează k și se formează un nou ciclu dar cu genele din p_2
 5. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_2)
 6. Se repetă pașii 2-5 până când $k = L$





Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

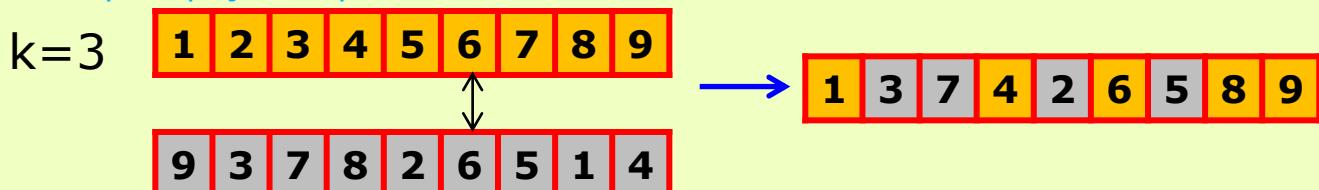
- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare ciclică
 - Ideea de bază
 1. inițial $k = 1$
 2. Se formează un ciclu
 - Se adaugă în ciclu gena de pe poziția k din p_1 (g_k^1)
 - Se consideră gena de pe poziția k din p_2 (g_k^2)
 - Se alege gena din p_1 cu valoarea egală cu g_k^2 (g_r^1) și se include în ciclu
 - Se consideră gena de pe poziția r din p_2 (g_r^2)
 - Se repetă pașii anteriori până când se ajunge la gena de pe poziția k din p_1
 3. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_1)
 4. Se incrementează k și se formează un nou ciclu dar cu genele din p_2
 5. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_2)
 6. Se repetă pașii 2-5 până când $k = L$





Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare ciclică
 - Ideea de bază
 1. inițial $k = 1$
 2. Se formează un ciclu
 - Se adaugă în ciclu gena de pe poziția k din p_1 (g_k^1)
 - Se consideră gena de pe poziția k din p_2 (g_k^2)
 - Se alege gena din p_1 cu valoarea egală cu g_k^2 (g_r^1) și se include în ciclu
 - Se consideră gena de pe poziția r din p_2 (g_r^2)
 - Se repetă pașii anteriori până când se ajunge la gena de pe poziția k din p_1
 3. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_1)
 4. Se incrementează k și se formează un nou ciclu dar cu genele din p_2
 5. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_2)
 6. Se repetă pașii 2-5 până când $k = L$





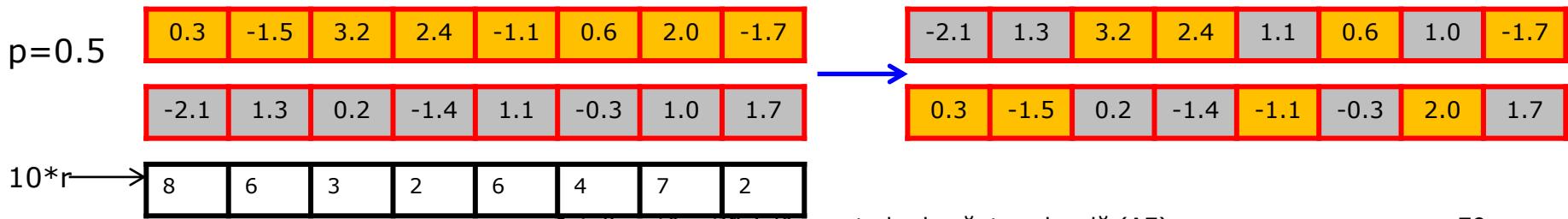
Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare bazată pe muchii
 - A se consulta: *Whitley, Darrell, Timothy Starkweather, D'Ann Fuquay (1989). "Scheduling problems and traveling salesman: The genetic edge recombination operator". International Conference on Genetic Algorithms. pp. 133–140* [link](#)



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare discretă
 - Ideea de bază
 - Fiecare genă a unui descendente este luată (cu aceeași probabilitate, $p = 0.5$) dintr-unul din părinti
 - Similar încrucișării uniforme de la reprezentarea binară/întreagă
 - Nu se modifică valorile efective ale genelor (nu se creează informație nouă)





Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)

- ❑ Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare intermediară (aritmetică)
 - Ideea de bază
 - Se creează copii aflați (ca valoare) între părinți → încrucișare aritmetică
 - $z_i = \alpha x_i + (1 - \alpha) y_i$ unde $\alpha : 0 \leq \alpha \leq 1$.
 - Parametrul α poate fi:
 - Constant → încrucișare aritmetică uniformă
 - Variabil → ex. dependent de vârstă populației
 - Aleator pt fiecare încrucișare produsă
 - Apar noi valori ale genelor

- Tipologie
 - ❑ Încrucișare aritmetică singulară
 - ❑ Încrucișare aritmetică simplă
 - ❑ Încrucișare aritmetică completă

Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



- ❑ Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- ❑ se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- ❑ Încrucișare intermediară (aritmetică) singulară
 - Se alege câte o genă (de același index k) din cei doi părinti și se combină
 - $g_k' = \alpha g_k^1 + (1-\alpha)g_k^2$
 - $g_k'' = (1-\alpha)g_k^1 + \alpha g_k^2$
 - Restul genelor rămân neschimbate
 - $g_i' = g_i^1$
 - $g_i'' = g_i^2$, pentru $i = 1,2,\dots,L$ și $i \neq k$

$$[LI, LS] = [-2.5, +3]$$

$$k=3$$

$$\alpha = 0.6$$

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7



0.3	-1.5	2.0	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	1.4	-1.4	1.1	-0.3	1.0	1.7

$$0.6 * 3.2 + (1-0.6) * 0.2 = 2.0$$

$$(1-0.6) * 3.2 + 0.6 * 0.2 = 1.4$$



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare intermediară (aritmetică) simplă
 - Se alege o poziție k și se combină toate genele de după acea poziție
 - $g_i' = \alpha g_i^1 + (1-\alpha)g_i^2$
 - $g_i'' = (1-\alpha)g_i^1 + \alpha g_i^2$, pentru $i=k, k+1, \dots, L$
 - Genele de pe pozitii $< k$ rămân neschimbrate
 - $g_i' = g_i^1$
 - $g_i'' = g_i^2$, pentru $i = 1,2,\dots,k-1$

$$[LI, LS] = [-2.5, +3]$$

$$k=6$$

$$\alpha = 0.6$$

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

0.3	-1.5	3.2	2.4	-1.1	0.24	1.6	-0.34
-2.1	1.3	0.2	-1.4	1.1	0.06	1.4	0.34

$$0.6*0.6+(1-0.6)*(-0.3)=0.24$$

$$(1-0.6)*0.6+0.6*(-0.3)=0.06$$

Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)



- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare intermediară (aritmetică) completă
 - Toate genele (de pe poziții corespunzătoare) se combină
 - $g_i' = \alpha g_i^1 + (1-\alpha)g_i^2$
 - $g_i'' = (1-\alpha)g_i^1 + \alpha g_i^2$, pentru $i=1,2,\dots,L$

$$0.6*0.3+(1-0.6)*(-2.1)=-0.66$$

$$(1-0.6)*0.3+0.6*(-2.1)=-1.14$$

$[LI, LS] = [-2.5, +3]$

$\alpha = 0.6$

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7

-0.66	4.3	2.0	0.48	-0.22	0.24	1.6	-0.34
-1.14	0.18	1.4	0.12	0.22	0.06	1.4	0.34



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare geometrică
 - Ideea de bază
 - Fiecare genă a unui descendente reprezintă produsul genelor părintilor, fiecare cu un anumit exponent ω , respectiv $1-\omega$ (unde ω număr real pozitiv subunitar)
 - $g_i' = (g_i^1)^\omega (g_i^2)^{1-\omega}$
 - $g_i'' = (g_i^1)^{1-\omega} (g_i^2)^\omega$

$$[LI, LS] = [-2.5, +3]$$

$$\omega = 0.7$$

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7

$$0.3^{0.7} + 2.1^{1-0.7} = 1.68$$

$$0.3^{1-0.7} + 2.1^{0.7} = 2.38$$

1.68	2.41	2.87	2.95	2.10	1.40	2.62	2.62
2.38	2.33	1.74	2.57	2.10	1.29	2.23	2.62

Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. reală)



- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtine 1 descendant
 - $c_1 = (g_1', g_2', \dots, g_L')$,
 - unde $g_i^1, g_i^2, g_i' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare amestecată (*blend crossover – BLX*)
 - Ideea de bază
 - Se generează un singur descendant
 - Genele g_i' ale descendantului sunt alese aleator în intervalul $[Min_i - I*a, Max_i + I*a]$, unde:
 - $Min_i = \min\{g_i^1, g_i^2\}$, $Max_i = \max\{g_i^1, g_i^2\}$
 - $I = Max - Min$, a – parametru din $[0,1]$

$$[LI, LS] = [-2.5, +3]$$

$$a = 0.7$$

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7



1.25	1.45	-1.11	2.37	1.10	0.11	0.70	1.70
------	------	-------	------	------	------	------	------

Min	0.3	1.3	0.2	1.4	1.1	0.3	1.0	1.7
Max	2.1	1.5	3.2	2.4	1.1	0.6	2.0	1.7
I	0.8	0.2	3.0	1.0	0	0.3	1.0	0.0

Min-Ia	-0.26	1.16	-1.90	0.70	1.10	0.09	0.30	1.70
Max+Ia	2.66	1.50	3.20	2.40	1.10	0.60	2.00	1.70



Algoritmi evolutivi – algoritm Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinti
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obtin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare binară simulată
 - Ideea de bază
 - Fiecare genă a unui descendente reprezintă o combinație a genelor părintilor
$$d_1 = \frac{p_1 + p_2}{2} - \beta \frac{p_2 - p_1}{2}, \quad d_2 = \frac{p_1 + p_2}{2} + \beta \frac{p_2 - p_1}{2}$$
 - a.i. să se respecte cele 2 proprietăți de la încrucișarea cu n puncte de tăietură (pt. reprezentarea binară)
 - media valorilor codate în părinti = media valorilor codate în descendenți
 - probabilitatea apariției unui factor de răspândire $\beta \approx 1$ este mai mare decât a oricărui alt factor

Algoritmi evolutivi – algoritm Proiectare – recombinarea



□ Recombinarea multiplă

- Bazată pe frecvența valorilor din părinți (încrucișare uniformă generală)
- Bazată pe segmentare și recombinare (încrucișare generală cu puncte de tăietură diagonală)
- Bazată pe operații numerice specifice valorilor reale (încrucișare bazată pe centrul de masă, încrucișare generală aritmetică)

Algoritmi evolutivi – algoritm Proiectare – mutație sau recombinare?

- Dezbateri aprinse
 - Întrebări:
 - care operator este mai bun?
 - care operator este necesar?,
 - care operator este mai important?
 - Răspunsuri:
 - Depinde de problemă, dar
 - În general, este bine să fie folosiți ambii operatori
 - Fiecare având alt rol
 - Sunt posibili AE doar cu mutație, dar nu sunt posibili AE doar cu încrucișare
- Aspecte ale căutării:
 - Explorare → descoperirea regiunilor promițătoare ale spațiului de căutare (acumulând informație utilă despre problemă)
 - Exploatare → optimizarea într-o regiune promițătoare (folosind informația existentă)
 - Trebuie să existe cooperare și competiție între aceste 2 aspecte
- Încrucișarea
 - Operator exploataativ, realizând un mare salt într-o regiune undeva între regiunile asociate părintilor
 - Efectele exploataitive se reduc pe măsură ce AE converge
 - Operator binar (n-ar) care poate combina informația din 2 (sau mai mulți) părinți
 - Operator care nu schimbă frecvența valorilor din cromozomi la nivelul întregii populații
- Mutatia
 - Operator explorativ, realizând mici diversiuni aleatoare, rămânând în regiunea apropiată părintelui
 - Evadarea din optimele locale
 - Operator care poate introduce informație genetică nouă
 - Operator care schimbă frecvența valorilor din cromozomi la nivelul întregii populații

Algoritmi evolutivi – algoritm Proiectare – criteriu de oprire



- Stabilirea unui criteriu de stop
 - S-a identificat soluția optimă
 - S-au epuizat resursele fizice
 - S-a efectuat un anumit număr de evaluaări ale funcăiei de fitness
 - S-au epuizat resursele utilizatorului (timp, răbdare)
 - S-au “născut” câteva generaăii fără îmbunătăăiri



Algoritmi evolutivi - algoritm

□ Evaluarea performanțelor unui AE

- După mai multe rulări se calculează:

- Măsuri statistice

- media soluțiilor,
 - mediana soluțiilor,
 - cea mai bună soluție,
 - cea mai slabă soluție,
 - deviația standard – pentru comparabilitate

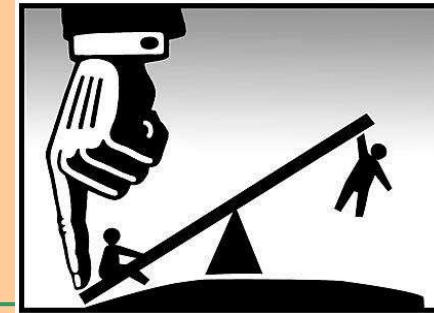
- Calculate pentru un număr suficient de mare de rulări independente

Algoritmi evolutivi



- Analiza complexității
 - Partea cea mai costisitoare → calculul fitnessului

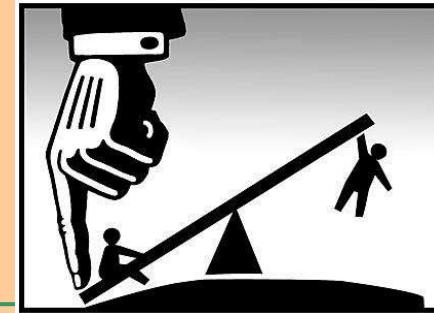
Algoritmi evolutivi



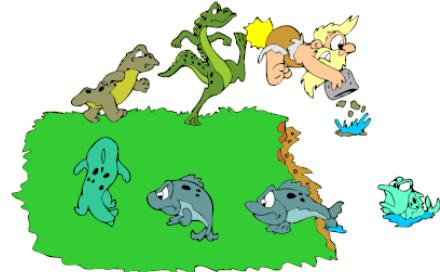
□ Avantaje

- Schema AE universală pentru toate problemele
 - se modifică doar
 - reprezentarea
 - funcția de fitness
- AE sunt capabili să producă rezultate mai bune decât metodele convenționale de optimizare pentru că:
 - nu necesită liniarizare
 - nu implică anumite presupuneri (continuitate, derivabilitate, etc. a funcției obiectiv)
 - nu ignoră anumite potențiale soluții
- AE sunt capabili să exploreze mai multe potențiale soluții decât poate explora omul

Algoritmi evolutivi



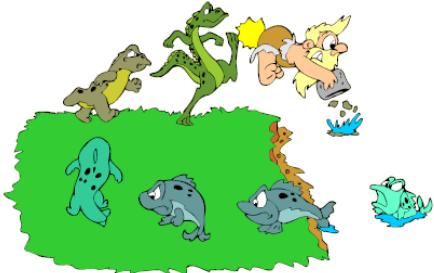
- Dezavantaje
 - Timp de rulare îndelungat



Algoritmi evolutivi

□ Aplicații

- Proiectări vehicule
 - Componența materialelor
 - Forma vehiculelor
- Proiectări inginerești
 - Optimizarea structurală și organizatorică a construcțiilor (clădiri, roboți, sateliți, turbine)
- Robotică
 - Optimizarea proiectării, funcționării componentelor
- Evoluare de hardware
 - Optimizarea de circuite digitale
- Optimizarea telecomunicațiilor
- Generarea de glume și jocuri de cuvinte
- Invenții biomimetice (inspirate de arhitecturi naturale)
- Rutări pentru trafic și transporturi
- Jocuri de calculator
- Criptări
- Profilul expresiv al genelor
- Analiza chimică a cinceticii
- Strategii financiare și marketing



Algoritmi evolutivi

- Tipuri de algoritmi evolutivi
 - Strategii evolutive
 - Programare evolutivă
 - Algoritmi genetici
 - Programare genetică

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

- Sisteme care învață singure

- Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi

- Sisteme bazate pe reguli

- Sisteme hibride

C. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare

- Strategii de căutare

- Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

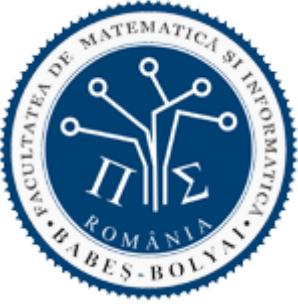
Cursul următor – Materiale de citit și legături utile

- capitolul 16 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *James Kennedy, Russel Eberhart, Particle Swarm Optimisation, Proceedings of IEEE International Conference on Neural Networks. IV.* pp. 1942–1948, 1995
(04_ACO_PSO/PSO_00.pdf)
- *Marco Dorigo, Christian Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243 – 27* (04_ACO_PSO/Dorigo05_ACO.pdf)

-
- ❑ Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare

Strategii de căutare informată
algoritmi inspirați de natură

Laura Dioșan

Sumar

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

- Sisteme care învață singure

- Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi

- Sisteme bazate pe reguli

- Sisteme hibride

C. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare

- Strategii de căutare

- Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

Materiale de citit și legături utile

- capitolul 16 din *C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- *James Kennedy, Russel Eberhart, Particle Swarm Optimisation, Proceedings of IEEE International Conference on Neural Networks. IV.* pp. 1942–1948, 1995
(05_ACO_PSO/PSO_00.pdf)
- *Marco Dorigo, Christian Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243 – 27* (05_ACO_PSO/Dorigo05_ACO.pdf)

Căutare locală

□ Tipologie

- Căutare locală simplă - se reține o singură stare vecină
 - Căutare tabu → reține lista soluțiilor recent vizitate
 - Hill climbing → alege cel mai bun vecin
 - Simulated annealing → alege probabilistic cel mai bun vecin
- Căutare locală în fascicol (beam local search) – se rețin mai multe stări (o populație de stări)
 - Algoritmi evolutivi
 - Optimizare bazată pe comportamentul de grup (Particle swarm optimisation)
 - Optimizare bazată pe furnici (Ant colony optimisation)

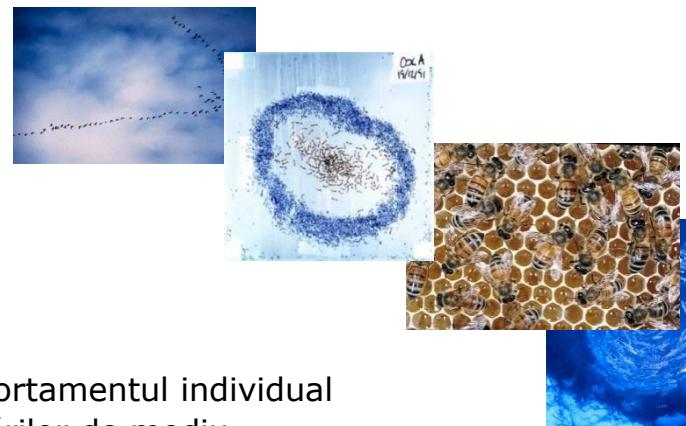
Algoritmi inspirați de natură

- Care este cea mai bună metodă de rezolvare a unei probleme?
 - Creierul uman
 - A creat roata, mașina, orașul, etc
 - Mecanismul evoluției
 - A creată creierul (mintea) umană
- Simularea naturii
 - Cu ajutorul mașinilor → rețele neuronale artificiale simulează mintea umană
 - Mașini de zbor, computere bazate pe ADN, computere cu membrane
 - Cu ajutorul algoritmilor
 - algoritmii evolutivi simulează evoluția naturii
 - algoritmii inspirați de comportamentul de grup simulează adaptarea colectivă și procesele sociale dintr-un colectiv
 - Particle Swarm Optimisation (PSO)
 - <http://www.youtube.com/watch?feature=endscreen&v=JhZKc1Mgub8&NR=1>
 - <http://www.youtube.com/watch?v=ulucJnxT7B4&feature=related>
 - <https://www.youtube.com/watch?v=TWqx57CR69c>
 - Ant Colony Optimisation (ACO)
 - http://www.youtube.com/watch?v=jrW_TTxP1ow

Algoritmi inspirați de natură

❑ Inteligența de grup (colectivă)

- O populație de indivizi care interacționează în scopul atingerii unor obiective prin adaptarea colectivă la un mediu global sau local
- Metaforă computațională inspirată de:
 - zborul păsărilor în formă de V
 - furnicile aflate în căutarea hranei
 - roiurile de albine care își construiesc cuibul
 - bancurile de pești
- deoarece
 - controlul este distribuit între mai mulți indivizi
 - comunicarea între indivizi se realizează local
 - comportamentul sistemului超越e din comportamentul individual
 - sistemul este robust și se poate adapta schimbărilor de mediu



■ Insecte sociale (2% din totalul insectelor):

- Furnici
 - 50% din insectele sociale
 - 1 furnică are aprox. 1 mg → Greutatea totală a furnicilor ≈ greutatea totală a oamenilor
 - Trăiesc de peste 100 milioane de ani (oamenii trăiesc de aprox. 50 000 de ani)
- Termite
- Albine

Algoritmi inspirați de natură

□ Grup (roi - Swarm)

- O colecție aparent dezorganizată de indivizi care se mișcă tinzând să se grupeze, dar fiecare individ pare să se miște într-o direcție oarecare
- În interiorul colecției apar anumite procese sociale
- Colecția este capabilă să efectueze sarcini complexe
 - fără nici o ghidare sau control extern
 - fără nici o coordonare centrală
- Colecția poate atinge performanțe care nu pot fi atinse de indivizi în izolare

□ Adaptare colectivă → auto-organizare

- Multimea mecanismelor dinamice care generează un comportament global ca rezultat al interacțiunii componentelor individuale
- Regulile care specifică interacțiunea sunt executate doar pe baza unor informații locale, fără referințe globale
- Comportamentul global este o proprietate emergentă a sistemului (și nu una impusă din exterior)

PSO

- ❑ Aspecte teoretice
- ❑ Algoritm
- ❑ Exemplu
- ❑ Proprietăți
- ❑ Aplicații

PSO – aspecte teoretice

- Propusă
 - de Kennedy și Eberhart în 1995 <http://www.particleswarm.info/>
 - Inspirată de comportamentul social al stolurilor de păsări și al bancurilor de pești
- Căutare
 - **Cooperativă**, ghidată de calitatea **relativă** a indivizilor
- Operatori de căutare
 - Un fel de mutație

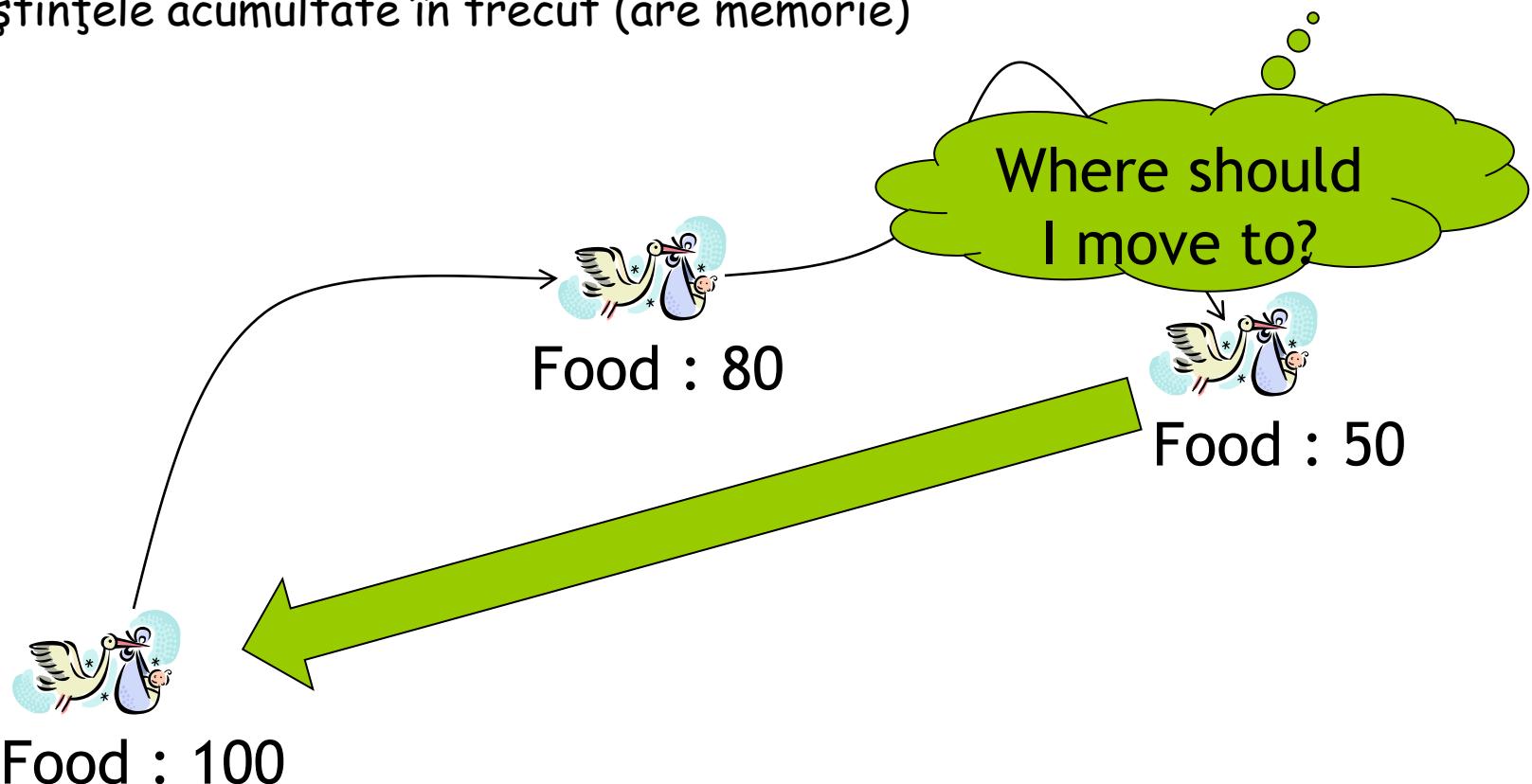
PSO – aspecte teoretice

□ Elemente speciale

- Metodă de optimizare bazată pe:
 - ▣ populații (\approx AG) de particule (\approx cromozomi) care caută soluția optimă
 - ▣ cooperare (în loc de competiție ca în cazul AG)
- Fiecare particulă:
 - ▣ Se mișcă (deplasează în spațiul de căutare) și are o viteză (viteză \approx mutare pt că timpul este discret)
 - ▣ Reține locul (poziția) unde a obținut cele mai bune rezultate
 - ▣ Are asociată o vecinătate de particule
- Particulele cooperează
 - ▣ Schimbă informații (legate de descoperirile făcute în locurile deja vizitate) între ele
 - ▣ Fiecare particulă știe fitnessul vecinilor ei a.î. poate folosi poziția celui mai bun vecin pentru a-și ajusta propria viteză

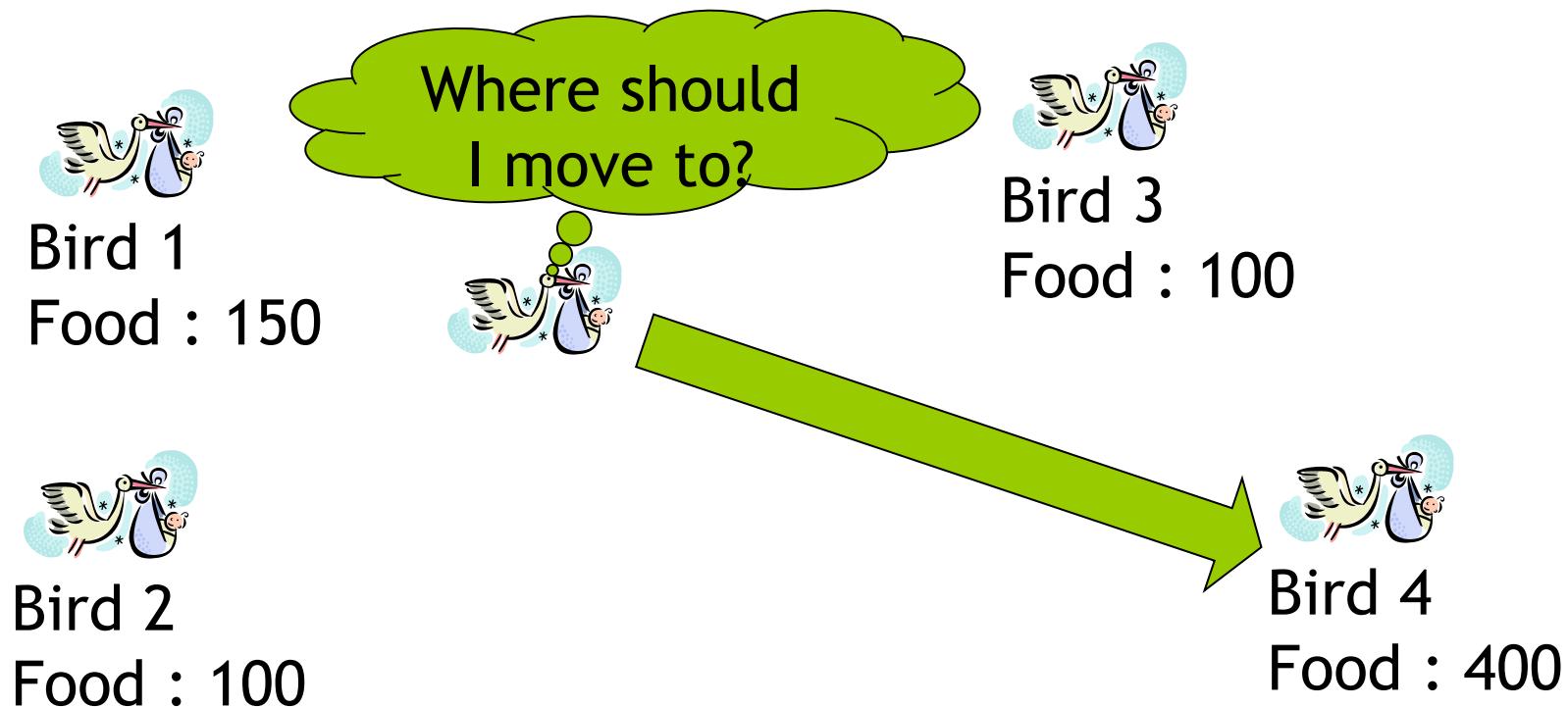
PSO – aspecte teoretice

Ideea de bază: comportament cognitiv → un individ își amintește cunoștințele acumulate în trecut (are memorie)



PSO – aspecte teoretice

Ideea de bază: comportament social → un individ se bazează și pe cunoștințele celorlalți membri ai grupului



PSO – algoritm

□ Schema generală

1. Crearea populației inițiale de particule
 - Poziții aleatoare
 - Viteze nule/aleatoare
2. Evaluarea particulelor
3. Pentru fiecare particulă
 - Actualizarea memoriei
 - Stabilirea celei mai bune particule din swarm (g_{Best}) / dintre particulele vecine (l_{Best})
 - Stabilirea celei mai bune poziții (cu cel mai bun fitness) în care a ajuns până atunci – p_{Best}
 - Modificarea vitezei
 - Modificarea poziției
4. Dacă nu se îndeplinesc condițiile de oprire, se revine la pasul 2, altfel STOP

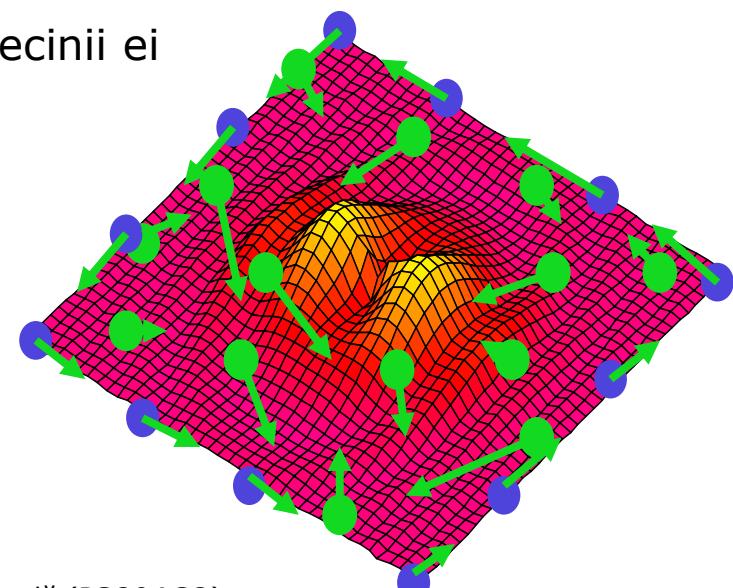
PSO – algoritm

1. Crearea populației inițiale de particule

- Fiecare particulă are asociată
 - o poziție – potențială soluție a problemei
 - o viteză – modifică o poziție în altă poziție
 - o funcție de calitate (fitness)

- Fiecare particulă trebuie să poată:
 - interacționa (schimba informații) cu vecinii ei
 - memora o poziție precedentă
 - utilizează informațiile pentru a lua decizii

- Inițializarea particulelor
 - poziții aleatoare
 - viteze nule/aleatoare



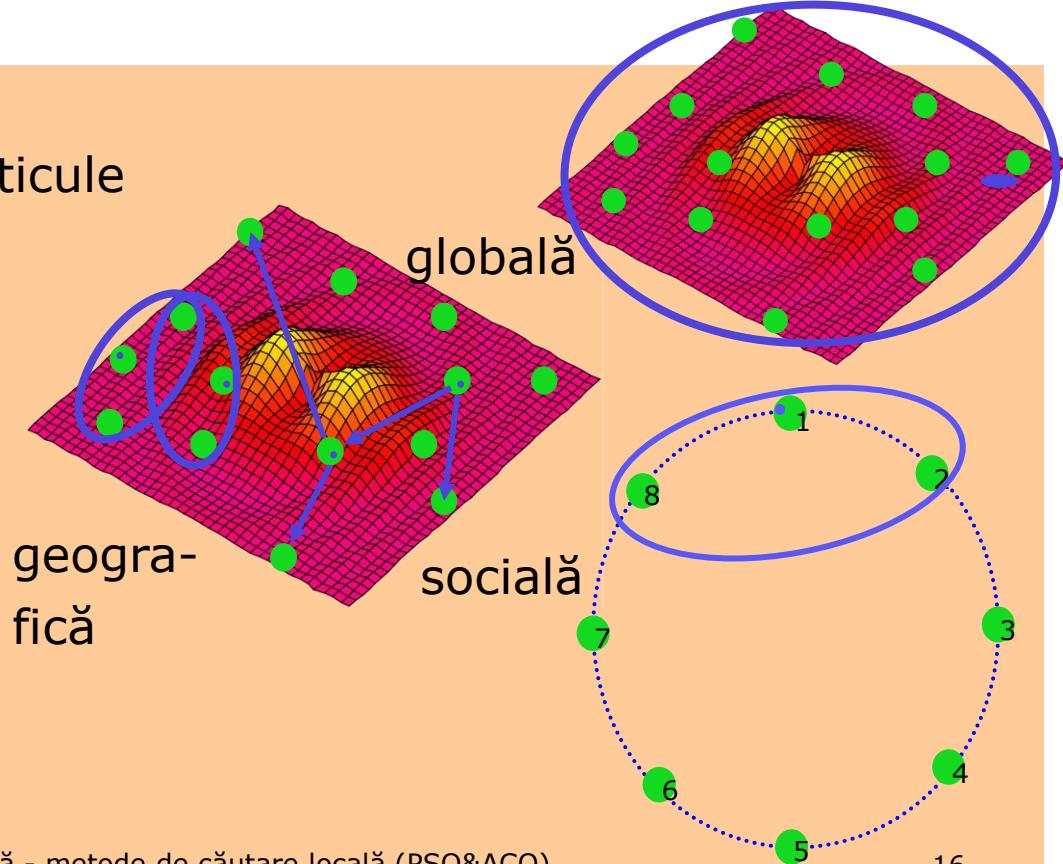
PSO – algoritm

2. Evaluarea particulelor
 - dependentă de problemă

PSO – algoritm

3. Pentru fiecare particulă x

- Actualizarea memoriei
 - Stabilirea celei mai bune particule din swarm (g_{Best}) / dintre particulele vecine (l_{Best})
- Vecinătate a unei particule
 - Întinderea vecinătății
 - Globală
 - Locală
 - Tipul vecinătății
 - Geografică
 - Socială
 - Circulară

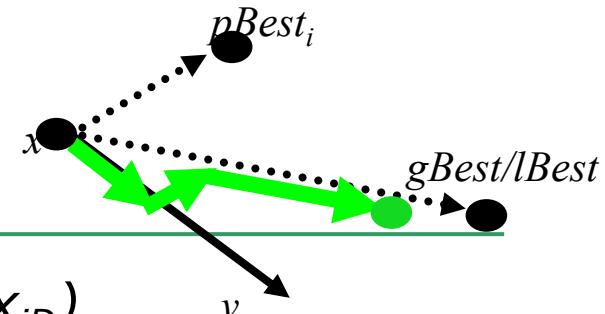


PSO – algoritm

3. Pentru fiecare particulă x

- Actualizarea memoriei
 - Stabilirea celei mai bune particule din swarm (g_{Best}) / dintre particulele vecine (l_{Best})
 - Stabilirea celei mai bune poziții (cu cel mai bun fitness) în care a ajuns până atunci – p_{Best}

PSO – algoritm



3. Pentru fiecare particulă $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iD})$

■ Modificarea vitezei \mathbf{v} și a poziției \mathbf{x} (pe fiecare dimensiune)

- $v_{id} = w * v_{id} + c_1 * rand() * (p_{Best_d} - x_{id}) + c_2 * rand() * (g_{Best_d} - x_{id})$
- $x_{id} = x_{id} + v_{id}$
- unde:
 - $i=1, N$ (N – nr total de particule); $d = 1, D$
 - w – factor de inertie (Shi, Eberhart)
 - $w * v_{id}$ – termen inertial → forțează particula să se deplaseze în aceeași direcție ca și până acum (tendință curajoasă – *audacious*)
 - balansează căutarea între explorare globală (w mare) și locală (w mic).
 - poate fi constantă sau descrescătoare (pe măsura „îmbătrânirii” grupului)
 - c_1 - factor de învățare cognitiv
 - $c_1 * rand() * (p_{Best_d} - x_{id})$ – termen cognitiv → forțează particula să se deplaseze spre cea mai bună poziție atinsă până atunci (tendință de conservare)
 - c_2 - factor de învățare social
 - $c_2 * rand() * (g_{Best_d} - x_{id})$ – termen social → forțează particula să se deplaseze spre cea mai bună poziție a vecinilor; spirit de turmă, de urmăritor
 - Cei doi factori c_1 și c_2 pot fi egali sau diferiți ($c_1 > c_2$ și $c_1 + c_2 < 4$ – Carlisle, 2001)
- Fiecare componentă a vectorului vitezelor este restricționată la un interval: $[-v_{max}, v_{max}]$ pentru a asigura păstrarea particulelor în spațiul de căutare.

PSO – proprietăți

- Principii în PSO:
 - proximitate – grupul trebuie să efectueze calcule în spațiu și timp
 - calitate – grupul trebuie să fie capabil să răspundă la factorii calitativi ai mediului
 - stabilitate – grupul nu trebuie să își schimbe comportamentul la fiecare sesizare a mediului
 - adaptabilitate – grupul trebuie să fie capabil să își schimbe comportamentul atunci când costul schimbării nu este prohibit.
- Diferențe față de EC:
 - nu există un operator de recombinare directă – schimbul de informație are loc în funcție de experiența particulei și în funcție de cea a celui mai bun vecin și nu în funcție de părinții selectați pe baza fitness-ului.
 - Update poziție ~ similar cu mutația
 - Nu se folosește selecția – supraviețuirea nu este legată de fitness.
- Versiuni ale algoritmului de tip PSO
 - PSO binar discret
 - PSO cu mai mulți termeni de învățare socială
 - PSO cu particule eterogene
 - PSO ierarhic

PSO – proprietăți

□ PSO discret (binar)

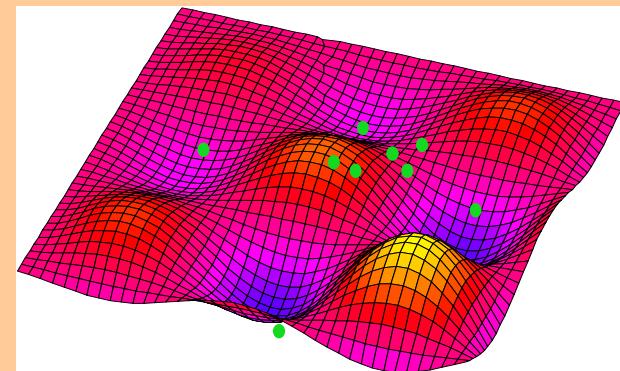
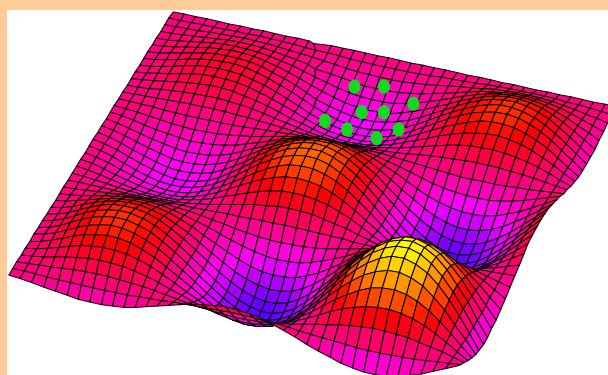
- Versiune a PSO pentru spațiu de căutare discret
- Poziția unei particule
 - Potențială soluție a problemei → string binar
 - Se modifică în funcție de viteza particulei
- Viteza unei particule
 - element din spațiu continuu
 - se modifică conform principiilor de la PSO standard
 - se interpretează ca probabilitatea de modificare a bit-ului corespunzător din poziția particulei

$$x_{ij} = \begin{cases} 1, & \text{dacă } \tau < s(v_{ij}) \\ 0, & \text{altfel} \end{cases}, \text{ unde } s(v_{ij}) = \frac{1}{1 + e^{-v_{ij}}}$$

PSO – proprietăți

❑ Pericole

- Particulele tind să se grupeze în același loc
 - ❑ Converg prea repede și nu reușesc să evadzeze dintr-un optim local
 - ❑ Soluția:
 - Reinițializarea unor particule



- Deplasarea particulelor spre regiuni nefezabile

PSO – proprietăți

□ Analiza algoritmilor de tip PSO

- Comportamentul dinamic al grupului poate fi analizat cu ajutorul a 2 indici

- Indicele de dispersie

- Măsoară gradul de împărtiere a particulelor în jurul celei mai bune particule din grup
 - Media distanțelor absolute (pe fiecare dimensiune) între fiecare particulă și particula cea mai bună
 - Explică gradul de acoperire (întins sau restrâns) a spațiului de căutare

- Indicele vitezei

- Măsoară viteza de mișcare a grupului într-o iteratăie
 - Media vitezelor absolute
 - Explică cum (agresiv sau lent) se mișcă grupul

PSO – aplicații

- ❑ Controlul și proiectarea antenelor
- ❑ Aplicații biologice, medicale, farmaceutice
 - Analiza tremurului în boala Parkinson
 - Clasificare cancerului
 - Predicția structurii proteinelor
- ❑ Comunicare în rețele
- ❑ Optimizare combinatorială
- ❑ Optimizări financiare
- ❑ Analiza imaginilor și analiza video
- ❑ Robotică
- ❑ Planificare
- ❑ Securitatea rețelelor, detecția intrușilor, criptografie, criptanaliză
- ❑ Procesarea semnalelor

ACO

- Aspecte teoretice
- Algoritm
- Exemplu
- Proprietăți
- Aplicații

ACO – aspecte teoretice

□ Propusă

- de Colorni și Dorigo în 1991 inițial pentru rezolvarea problemelor de optimizare discretă – gen TSP – (ca o contrapartidă pentru AG) –
<http://iridia.ulb.ac.be/~mdorigo/ACO/about.html>
- inspirată de comportamentul social al furnicilor în căutarea unui drum între cuib și o sursă de hrana
- De ce furnici?
 - ▣ Munca în colonie (de la câteva furnici până la milioane de furnici)
 - ▣ Diviziunea muncii
 - ▣ Au comportament social complex

□ Căutare

- **Cooperativă**, ghidată de calitatea **relativă** a indivizilor

□ Operatori de căutare

- Constructivi, adăugând elemente în soluție

ACO – aspecte teoretice

□ Elemente speciale

- Problema de optimizare trebuie transformată într-o problemă de identificare a drumului optim într-un graf orientat
- Furnicile construiesc soluția plimbându-se prin graf și depunând pe muchii feromoni
- Metodă de optimizare bazată pe:
 - Colonii (\approx AG) de furnici (în loc de cromozomi) care caută soluția optimă
 - cooperare (în loc de competiție ca în cazul AG)
- Fiecare furnică:
 - Se mișcă (deplasează în spațiul de căutare) și depune o cantitate de feromon pe drumul parcurs
 - Reține drumul parcurs
 - Alege drumul pe care să-l urmeze în funcție de
 - Feromonul existent pe drum
 - Informația euristică asociată acelui drum
 - Cooperă cu celelalte furnici prin urma de feromon corespunzătoare unui drum care
 - depinde de calitatea soluției și
 - se evaporă cu trecerea timpului

ACO – aspecte teoretice

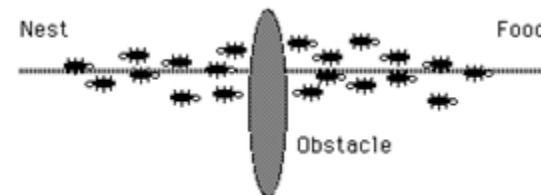
- Furnici naturale
 - O colonie de furnici pleacă în căutarea hranei



ACO – aspecte teoretice

□ Furnici naturale

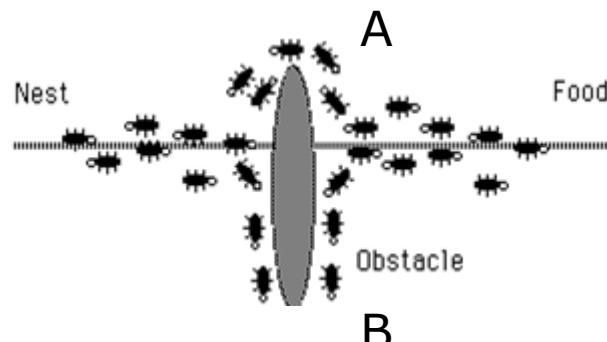
- O colonie de furnici pleacă în căutarea hranei
- La un moment dat, în drumul lor apare un obstacol



ACO – aspecte teoretice

❑ Furnici naturale

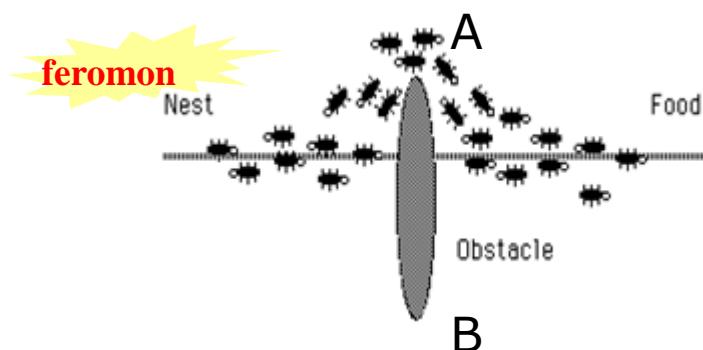
- O colonie de furnici pleacă în căutarea hranei
- La un moment dat, în drumul lor apare un obstacol
- Furnicile vor ocoli obstacolul fie pe ruta A, fie pe ruta B



ACO – aspecte teoretice

❑ Furnici naturale

- O colonie de furnici pleacă în căutarea hranei
- La un moment dat, în drumul lor apare un obstacol
- Furnicile vor ocoli obstacolul fie pe ruta A, fie pe ruta B
- Pentru că ruta A este mai scurtă, furnicile de pe acest drum vor face mai multe ture, deci vor lăsa mai mult feromon
- Concentrația de feromon va crește mai accelerat pe ruta A decât pe ruta B a.î. furniciile de pe ruta B vor alege (pe bază de miros) ruta A
- Pentru că pe ruta B nu vor mai merge furnici și pentru că feromonii sunt volatili, urma furnicilor de pe ruta B va dispărea
- Deci, furnicile se vor plimba doar pe cel mai scurt drum (ruta A)



ACO – aspecte teoretice

- Furnicile artificiale seamănă cu furnicile reale
 - navighează de la cuib spre sursa de hrana
 - descoperă drumul mai scurt pe baza urmei de feromon
 - fiecare frunică execută mișcări aleatoare
 - fiecare furnică depozitează feromon pe drumul parcurs
 - fiecare furnică detectează drumul urmat de "furnica șefă", înclinând să-l urmeze
 - creșterea cantității de feromon de pe un drum îi crește acestuia probabilitatea de a fi urmat de tot mai multe furnici
- dar au anumite îmbunătățiri:
 - au memorie
 - pentru a reține acțiunile efectuate → au stare proprie (cu istoricul acțiunilor efectuate)
 - se pot întoarce la cuib (și pe baza urmei de feromon)
 - nu sunt complet oarbe – pot aprecia calitatea spațiului vecin
 - execută mișcări într-un timp discret
 - depun feromoni și în funcție de calitatea soluției identificate

ACO – aspecte teoretice

- Urma de feromon are rolul
 - unei memorii colective dinamice distribuită (în colonie)
 - unui depozit cu cele mai recente experiențe de căutare a hranei ale furnicilor din colonie

- Furnicile pot comunica indirect și se pot influența reciproc
 - prin modificarea și mirosirea acestui depozit chimic
 - în vederea identificării celui mai scurt drum de la cuib până la hrană

ACO – algoritm

- Cât timp nu s-a ajuns la nr maxim de iterații
 1. Inițializare
 2. Cât timp nu s-a parcurs numărul necesar de pași pentru identificarea soluției
 - Pentru fiecare furnică din colonie
 - Se mărește soluția parțială cu un element (furnica execută o mutare)
 - Se modifică local urma de feromon corespunzător ultimului element adăugat în soluție
 3. Se modifică urma de feromon de pe drumurile parcuse de
 - Toate furnicile/cea mai bună furnică
 4. Se returnează soluția găsită de cea mai bună furnică

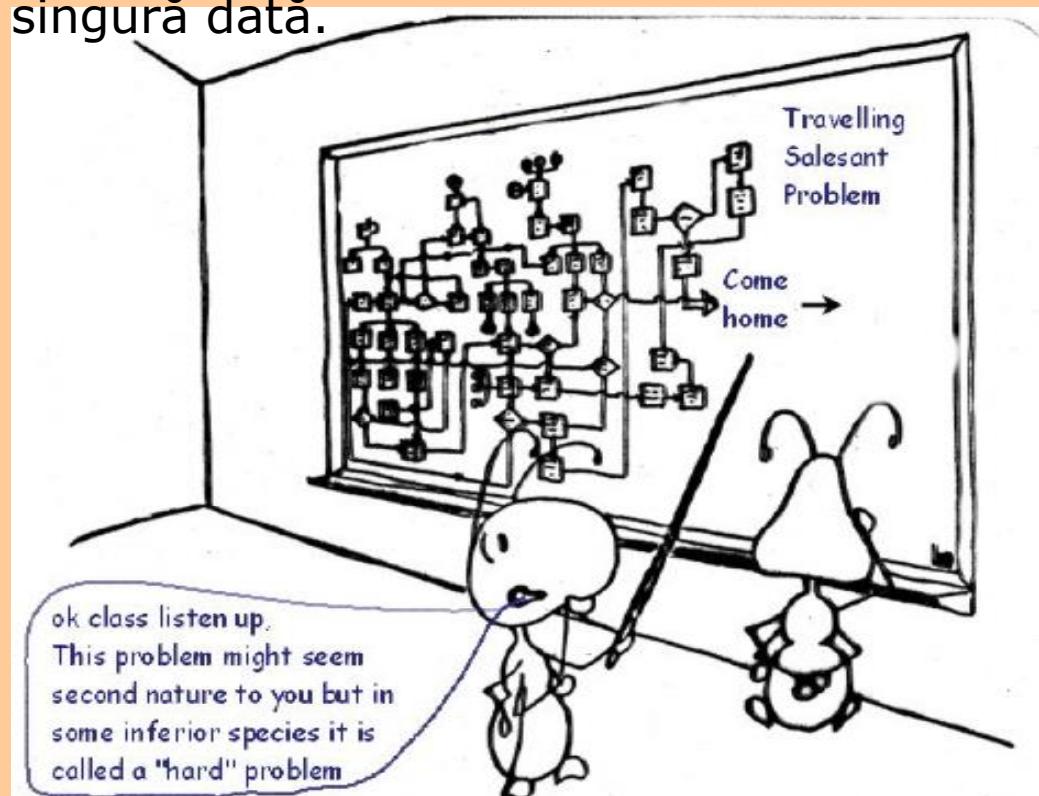
ACO – algoritm

- 3 versiuni principale în funcție de:
 - Regulile de tranziție de la o stare la alta (regulile de deplasare a furnicilor)
 - Momentul la care furnicile depun feromon:
 - pe parcursul construcției soluției
 - la sfârșitul creării unei soluții
 - Furnica deponentă de feromon
 - Toate furnicile
 - Doar cea mai bună furnică
- Versiuni:
 - Ant system (AS)
 - **Toate** furnicile depun feromon **după** construirea unei soluții **complete** (modificare globală colectivă)
 - MaxMin Ant System (MMAS) ≈ AS, dar
 - doar **cea mai bună** frunică depune feromon **după** construirea unei soluții **complete** (modificare globală a leader-ului)
 - feromonul depus este **limitat** la un interval dat
 - Ant Colony System (ACO) ≈ AS, dar
 - **toate** furnicile depun feromon **la fiecare pas** în construcția soluției (modificare locală colectivă)
 - doar **cea mai bună** furnică depune feromon după construirea unei soluții complete (modificare globală a leader-ului)

ACO – exemplu

□ Problema comisului voiajor

- *Travelling salesman problem - TSP*
- să se găsească un drum care să treacă prin n orașe (inclusiv între primul și ultimul) astfel încât costul să fie minim și fiecare oraș să fie vizitat o singură dată.



ACO – exemplu

1. Inițializare:

- $t := 0$ (timpul)
- pentru fiecare muchie (i,j) se inițializează
 - $\tau_{ij}^{(t)} = c$ (intensitatea urmei de feromon pe muchia (i,j) la momentul t)
 - $\Delta\tau_{ij} = 0$ (cantitatea de feromon lăsată pe muchia (i,j) de către toate furnicile)
- se plasează aleator m furnici în cele n noduri-oraș ($m \leq n$)
- fiecare furnică își modifică memoria (lista cu orașele vizitate)
 - adaugă în listă orașul din care pleacă în căutare

ACO – exemplu pentru TSP

2. Cât timp nu s-a parcurs numărul necesar de pași pentru construcția soluției (nr de pași = n)

■ Pentru fiecare furnică din colonie

- Se mărește soluția parțială cu un element (furnica execută o mutare)
 - fiecare furnică k (aflată în orașul i) alege următorul oraș pe care îl vizitează (j) astfel:

$$j = \begin{cases} \arg \max_{l \in permis_k} \{[\tau_{il}]^\alpha [\eta_{il}]^\beta\}, & \text{daca } q \leq q_0 \\ J, & \text{altfel} \end{cases}$$

Regula aleatoare proporțională

■ unde:

- q – număr aleator uniform distribuit în $[0,1]$
- q_0 – parametru, $0 \leq q_0 \leq 1$ ($q_0 = 0 \rightarrow$ AS/MMAS, altfel ACO)
- J este un oraș selectat cu probabilitatea

Regula pseudo-aleatoare proporțională

$$p_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}^{(t)}]^\alpha [\eta_{ij}]^\beta}{\sum_{s=permis_k(t)} [\tau_{is}^{(t)}]^\alpha [\eta_{is}]^\beta}, & j - permis \\ 0, & altfel \end{cases}$$

unde:

- p_{ij}^k – probabilitatea de tranziție a furnicii k situată în orașul i spre orașul j
- $\eta_{ij} = \frac{1}{d_{ij}}$ – vizibilitatea din orașul i spre orașul j (atractivitatea alegerii muchiei (i,j))
- $permis_k$ – orașele pe care le mai poate vizita a k -a furnică la momentul t
- α – controlează importanța urmei (câte furnici au mai trecut pe muchia respectivă)
- β – controlează importanța vizibilității (cât de aproape se află următorul oraș)

ACO – exemplu pentru TSP

2. Cât timp nu s-a parcurs numărul necesar de pași pentru construcția soluției (nr de pași = n)

- Pentru fiecare furnică din colonie
 - Se mărește soluția parțială cu un element (furnica execută o mutare)
 - Se modifică local urma de feromon lăsată de fiecare furnică pe ultimul element adăugat în soluție

$$\tau_{ij}^{(t+1)} = (1 - \varphi)\tau_{ij}^{(t)} + \varphi * \tau_0$$

- unde:
 - φ – coeficient de degradare a feromonului; $\varphi \in [0,1]$; pentru $\varphi = 0 \rightarrow$ AS/MMAS, altfel ACO
 - τ_0 – valoarea inițială a feromonului
 - (i,j) – ultima muchie parcursă de furnică

ACO – exemplu pentru TSP

3. Se modifică urma de feromon de pe
 - drumurile parcuse de toate furnicile (AS)
 - cel mai bun drum (ACO)
 - cel mai bun drum parcurs de cea mai bună furnică (MMAS)

ACO – exemplu pentru TSP

3. Se modifică urma de feromon de pe
 - **drumurile parcurse de toate furnicile (AS)**
 - Pentru fiecare muchie
 - Se calculează cantitatea unitară de feromoni lăsată de a k -a furnică pe muchia (ij)
 - $\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & - \text{dacă a } k\text{-a furnică a folosit muchia } (i,j) \\ 0 & \end{cases}$
 - Q – cantitatea de feromon lăsată de o furnică.
 - L_k – lungimea (costul) turului efectuat de a k -a furnică
 - Se calculează cantitatea totală de feromoni de pe muchia (ij) $\Delta\tau_{ij} = \sum_{k=1}^m \Delta\tau_{ij}^k$
 - Se calculează intensitatea urmei de feromoni ca sumă între evaporarea feromonilor vechi și feromonul nou lăsat $\tau_{ij}^{(t+1)} = (1-\rho) * \tau_{ij}^{(t)} + \Delta\tau_{ij}$
 - unde ρ ($0 < \rho < 1$) – coeficientul de evaporare a urmei de feromon între 2 tururi complete

ACO – exemplu pentru TSP

3. Se modifică urma de feromon de pe
 - **cel mai bun drum** (ACO/MMAS)
 - Pentru fiecare muchie a celui mai bun drum
 - Se calculează cantitatea unitară de feromoni lăsată de cea mai bună furnică pe muchia (ij)
 - L_{best} – lungimea (costul) celui mai bun drum
 - din iterația curentă
 - din toate iterațiile executate până atunci
 - Se calculează intensitatea urmei de feromoni ca sumă între evaporarea feromonilor vechi și feromonul nou lăsat
- $$\Delta \tau_{ij} = \frac{1}{L_{best}}$$
- $$\tau_{ij}^{(t+n)} = \left[(1 - \rho) * \tau_{ij}^{(t)} + \rho * \Delta \tau_{ij}^{best} \right]_{\tau_{min}}^{\tau_{max}}$$
- unde ρ ($0 < \rho < 1$) – coeficientul de evaporare a urmei de feromon între 2 tururi complete
 - τ_{min} și τ_{max} – limitele (inferioară și superioară) feromonului;
 - pentru $\tau_{min} = -\infty$ și $\tau_{max} = +\infty \rightarrow$ ACO, altfel MMAS

ACO – proprietăți

- Proprietăți
 - Algoritm iterativ
 - Algoritm care construiește progresiv soluția pe baza
 - Informațiilor euristice
 - Urmei de feromon
 - Algoritm stocastic
- Avantaje
 - Rulare neîntreruptă și adaptabilă schimbării în timp real a datelor de intrare
 - Ex. Pt TSP graful se poate modifica dinamic
 - Feedback-ul pozitiv ajută la descoperirea rapidă a soluției
 - Calculul distribuit evită convergența prematură
 - Euristică greedy ajută la găsirea unei soluții acceptabile încă din primele stadii ale căutării
 - Interacțiunea colectivă a indivizilor
- Dezavantaje
 - Converge încet față de alte căutări euristice
 - Funcționează relativ slab pentru instanțe cu mai mult de 75 de orașe ale TSP
 - În AS nu există un proces central care să ghideze căutarea spre soluțiile bune

ACO – aplicații

- Probleme de identificare a drumului optim în grafe
 - Ex. Traveling Salesman Problem
- Probleme de atribuirি quadratice
- Probleme de optimizări în retele
- Probleme de transport



Recapitulare

□ PSO

- Algoritm de căutare locală în fascicol
- Potențialele soluții → particule caracterizate prin:
 - poziție în spațiul de căutare
 - Viteză
- Căutare cooperativă și perturbativă bazată pe
 - Poziția celei mai bune particule din grup
 - Cea mai bună poziție a particulei de până atunci (particula are memorie)

□ ACO

- Algoritm de căutare locală în fascicol
- Potențialele soluții → furnici caracterizate prin:
 - Memorie – rețin pașii făcuți în construirea soluției
 - Miros – iau decizii pe baza feromonului depus de celelalte furnici (comportament social, colectiv, colaborativ)
- Căutare cooperativă și constructivă

Cursul următor

A. Scurtă introducere în Inteligența Artificială (IA)

B. Sisteme inteligente

- Sisteme care învață singure

- Arbori de decizie
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi

- Sisteme bazate pe reguli

- Sisteme hibride

C. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare

- Strategii de căutare

- Strategii de căutare neinformate
 - Strategii de căutare informate
 - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
 - Strategii de căutare adversială

Cursul următor – Materiale de citit și legături utile

- capitolul II.5 din *S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995*
- capitolul 6 din *H.F. Pop, G. Șerban, Inteligență artificială, Cluj Napoca, 2004*
- documentele din directorul *06_adversial_minimax*

-
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
 - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop