



UNIVERSITATEA BABEȘ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ

Sisteme inteligente

Sisteme care învață singure
– rețele 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

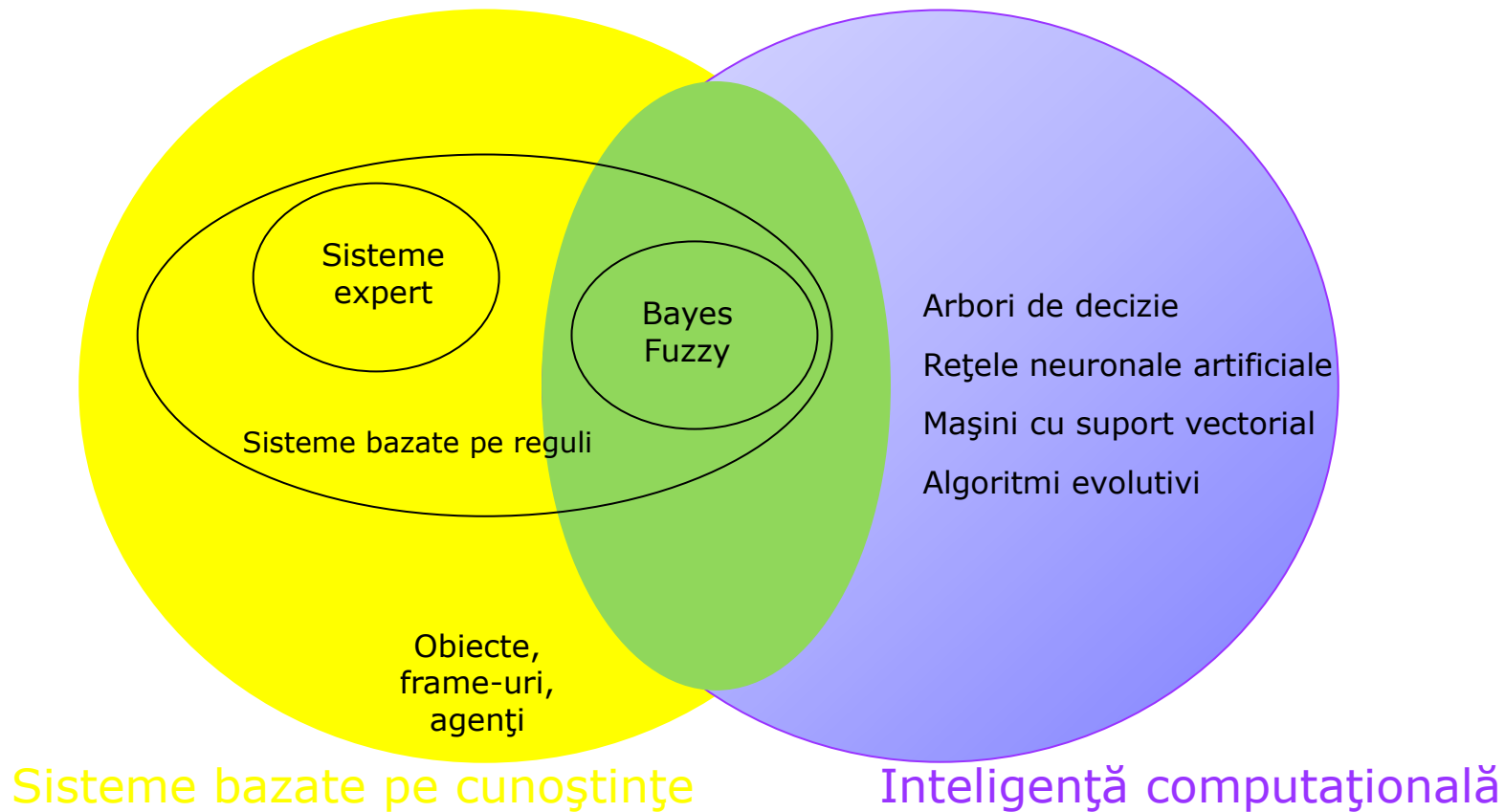
■ Strategii de căutare

- 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. 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 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 mână
 - vocea
 - Diferite raționamente
 - conducerea autovehiculelor
 - cântatul la pian
 - jucarea baschetului
 - înotul
- 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

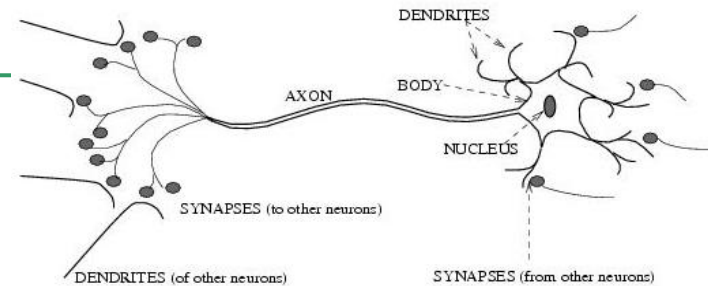
- Aproximativ 10.000.000.000 de neuroni conectați prin sinapse

□ Fiecare neuron

- are un corp (soma), un axon și multe dendrite
- poate fi într-una 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)?

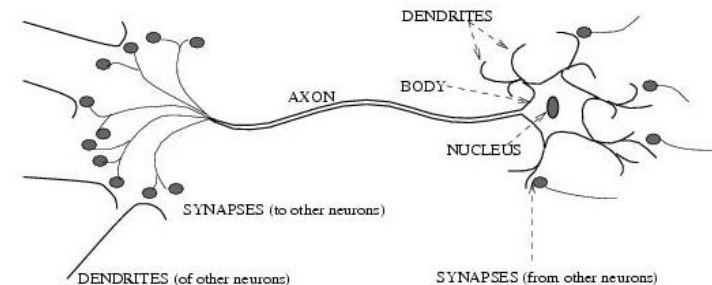
- Conexiunile care de-a lungul trăirii unor experiențe s-au dovedit utile devin permanente (restul sunt eliminate)
- Creierul este interesat de noutăți
- Modelul de procesare a informației
 - Învățare
 - Depozitare
 - Amintire

□ 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?

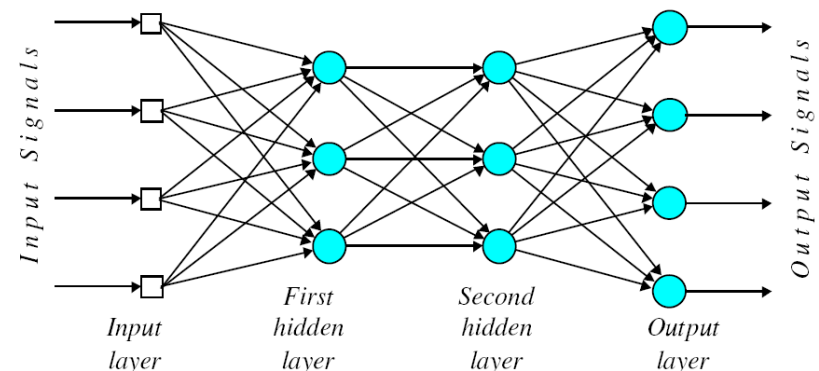
- O structură similară unei rețele neuronale biologice
- O mulțime de noduri (unități, neuroni, elemente de procesare) dispuse ca într-un graf pe mai multe straturi (*layere*)

□ 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:
 - $x^d \in \mathbf{R}^m \rightarrow x^d = (x^d_1, x^d_2, \dots, x^d_m)$
 - $t^d \in \mathbf{R}^R \rightarrow t^d = (t^d_1, t^d_2, \dots, t^d_R)$,
 - 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 – $x^d_1, x^d_2, \dots, x^d_m$)
 - stratul de ieșire poate conține R noduri (fiecare nod va furniza una dintre proprietățile de ieșire ale unei instanțe a problemei $t^d_1, t^d_2, \dots, t^d_R$)
 - 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

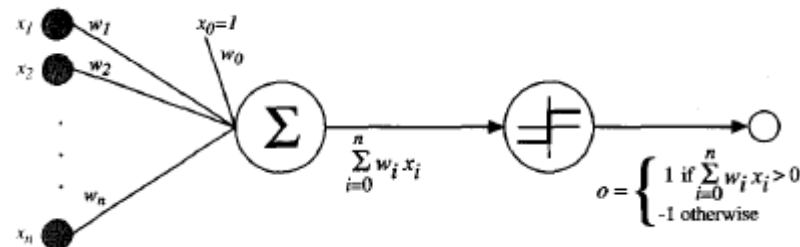
- Inițializarea ponderile între oricare 2 noduri de pe straturi diferite
- Stabilirea funcției de activare corespunzătoare fiecărui neuron (de pe straturile ascunse)

■ 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

$$net = \sum_{i=1}^n x_i w_i$$

- 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

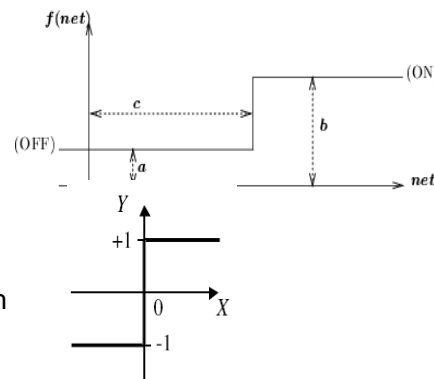
□ Proiectare → Antrenarea RNA → Cum învață rețeaua?

■ Funcția de activare a unui neuron

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

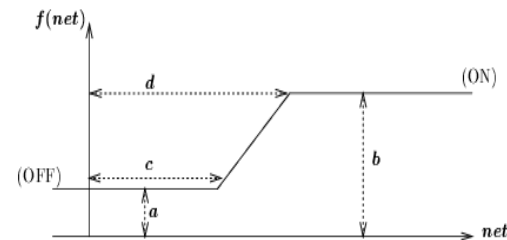
$$f(net) = \begin{cases} a, & \text{dacă } net < c \\ b, & \text{dacă } net > c \end{cases}$$

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



□ Funcția rampă

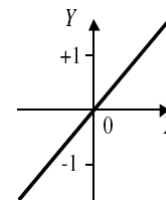
$$f(net) = \begin{cases} a, & \text{dacă } net \leq c \\ b, & \text{dacă } net \geq d \\ a + \frac{(net-c)(b-a)}{d-c}, & \text{altfel} \end{cases}$$



□ Funcția liniară

$$f(net) = a * net + b$$

- Pentru $a=1$ și $b=0$ → funcția identitate $f(net)=net$
- Funcție continuă



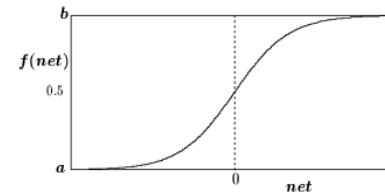
Sisteme inteligente – SIS – RNA

□ 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 asimptotic puncte de saturație



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

- Exemple de funcții sigmoidale:

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

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

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

- 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

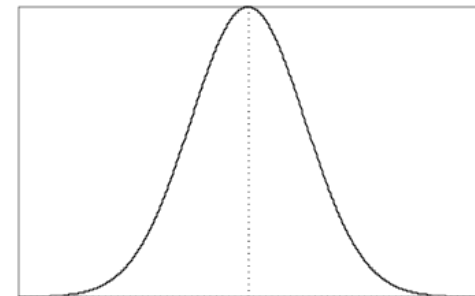
□ Funcția Gaussiană

- În formă de clopot
- Continuă
- Atinge asimptotic un punct de saturație

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

- Are un singur punct de optim (maxim) – atins când $net = \mu$
- 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 $[0, \infty)$

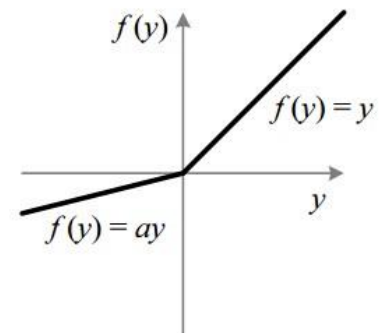
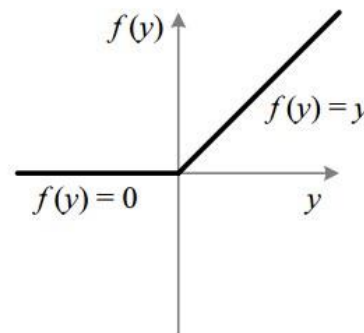
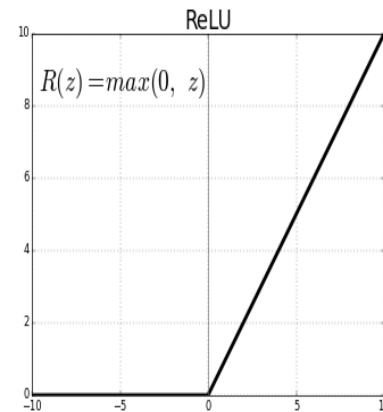
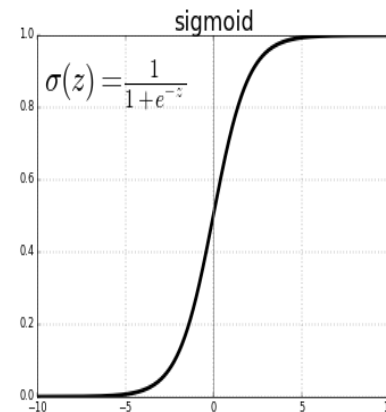
$$f(\text{net}) = \max(0, \text{net})$$

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

■ Variantă: Leaky ReLU

- Compensează problemele cu argumentele negative dint 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

$$net = \sum_{i=1}^n x_i w_i$$

- Neuronul procesează informația → se folosește o funcție de activare:

- Funcția constantă
- Funcția prag
- Funcția rampă
- Funcția liniară
- Funcția sigmoidală
- Funcția Gaussiană

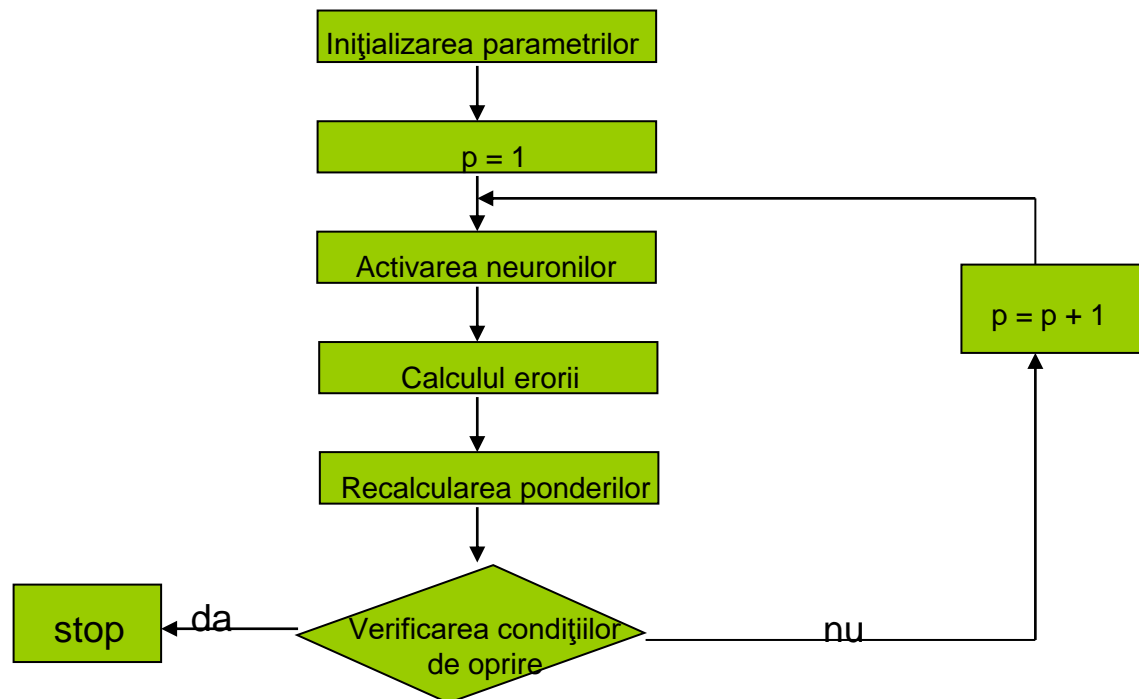
- Se citește răspunsul neuronului → se stabilește dacă rezultatul furnizat de neuron coincide sau nu cu cel dorit (real)

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:

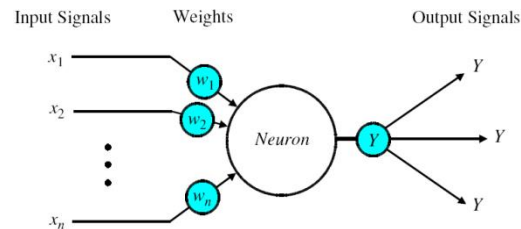
- $x^d \in \mathbf{R}^m \rightarrow x^d = (x^d_1, x^d_2, \dots, x^d_m)$

- $t^d \in \mathbf{R}^R \rightarrow t^d = (t^d_1, t^d_2, \dots, t^d_R)$, și $R = 1$ (adică $t^d = (t^d_1)$)

- cu $d = 1, 2, \dots, n$

■ RNA = unitate primitivă de calcul (un neuron) → 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

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

$d = 1$

Cât timp mai există exemple de antrenament clasificate incorect

Se activează neuronul și se calculează ieșirea

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

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

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

unde η - rată de învățare

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

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d=1}^n (t^d - o^d)^2$$

- Cum se determină cea mai abruptă pantă? → se derivează E în funcție de \mathbf{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

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

■ Funcție sigmoidală

$$\begin{aligned} f(net) &= \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}}} = \frac{1}{1 + e^{-\sum_{i=1}^m w_i x_i^d}} & y = s(z) &= \frac{1}{1 + e^{-z}} \Rightarrow \frac{\partial s(z)}{\partial z} = s(z)(1 - s(z)) \\ \frac{\partial E}{\partial w_i} &= \frac{\partial \frac{1}{2} \sum_{d=1}^n (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n \frac{\partial (t^d - o^d)^2}{\partial w_i} = \frac{1}{2} \sum_{d=1}^n 2(t^d - o^d) \frac{\partial (t^d - \text{sig}(\mathbf{w}\mathbf{x}^d))}{\partial w_i} = \sum_{d=1}^n (t^d - o^d) (1 - o^d) o^d (-x_i^d) \\ \Delta w_i &= -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d=1}^n (t^d - o^d) (1 - o^d) o^d x_i^d \end{aligned}$$

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</p> <p>$\Delta w_i = 0$, 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 $\rightarrow o^d = \mathbf{w}\mathbf{x}^d$</p> <p>funcția de activare = funcția sigmoid $\rightarrow o^d = \text{sig}(\mathbf{w}\mathbf{x}^d)$</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</p> <p>$\Delta w_i = 0$, 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 $\rightarrow o^d = \mathbf{w}\mathbf{x}^d$</p> <p>funcția de activare = funcția sigmoid $\rightarrow o^d = \text{sig}(\mathbf{w}\mathbf{x}^d)$</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

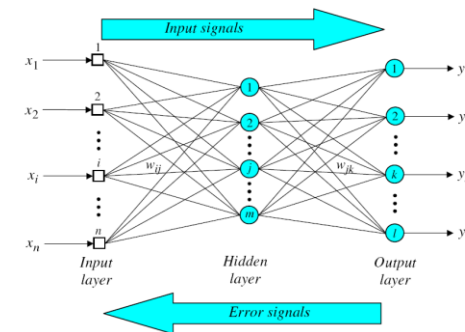
| Diferențe | Algoritmul perceptronului | Algoritmul scădere după gradient (regula Delta) |
|-----------------------------------|---|---|
| Ce reprezintă o^d | $o^d = \text{sign}(\mathbf{w}\mathbf{x}^d)$ | $o^d = \mathbf{w}\mathbf{x}^d$ sau $o^d = \text{sig}(\mathbf{w}\mathbf{x}^d)$ |
| Cum converge | Într-un nr finit de pași (până la separarea perfectă) | Asimptotic (spre eroarea minimă) |
| Ce fel de probleme se pot rezolva | Cu date liniar separabile | Cu orice fel de date (separabile liniar sau ne-liniar) |
| 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 așezați 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 algoritmul 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:

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

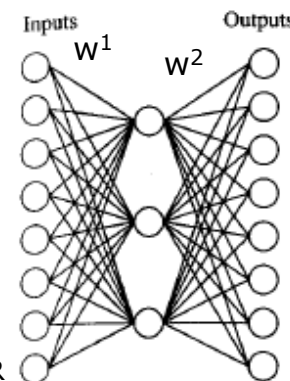
□ Presupunem 2 cazuri de RNA

■ O RNA cu un singur strat ascuns cu H neuroni → RNA₁

- 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, ..., p) neuroni → 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 $w_{ih_1}^1$ 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 $w_{h_1h_2}^2$ 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 $w_{h_2h_3}^3$ 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 $w_{ih_1}^1, w_{h_1h_2}^2, \dots, w_{h_{p-1}h_p}^p, w_{h_p r}^{p+1}$

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

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

$$o_{h_2}^d = \sum_{h_1=1}^{H_1} w_{h_1h_2}^2 o_{h_1}^d \text{ sau } o_{h_2}^d = \text{sig} \left(\sum_{h_1=1}^{H_1} w_{h_1h_2}^2 o_{h_1}^d \right), \text{ cu } h_2 = 1, 2, \dots, H_2$$

...

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

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

Sisteme inteligente – SIS – RNA

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

Se inițializează ponderile $w_{ih_1}^1, w_{h_1h_2}^2, \dots, w_{h_{p-1}h_p}^p, w_{h_p r}^{p+1}$

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 $w_{ih_1}^1, w_{h_1h_2}^2, \dots, w_{h_{p-1}h_p}^p, w_{h_p r}^{p+1}$

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

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

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

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

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

...

$$\delta_{h_1}^d = \sum_{h_2=1}^{H_2} w_{h_1h_2}^2 \delta_{h_2}^d \text{ sau } \delta_{h_1}^d = o_{h_1}^d (1 - o_{h_1}^d) \sum_{h_2=1}^{H_2} w_{h_1h_2}^2 \delta_{h_2}^d$$

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

Inteligență 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

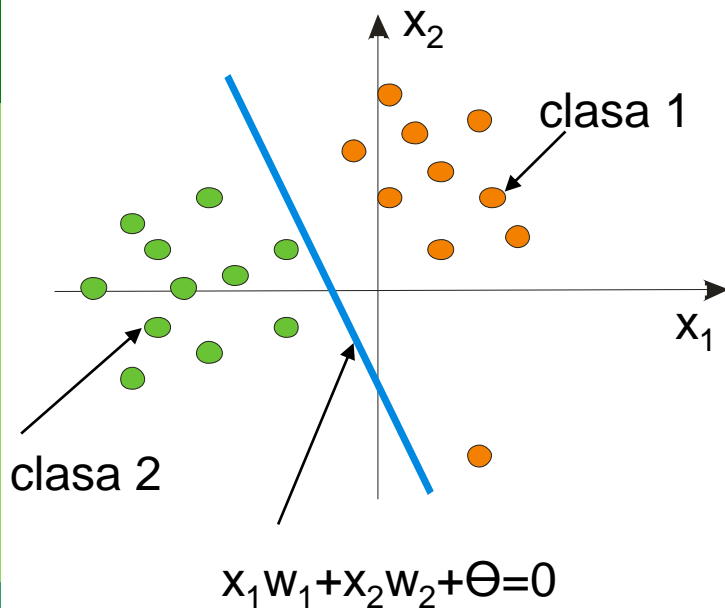
□ Proiectare

■ Testarea RNA

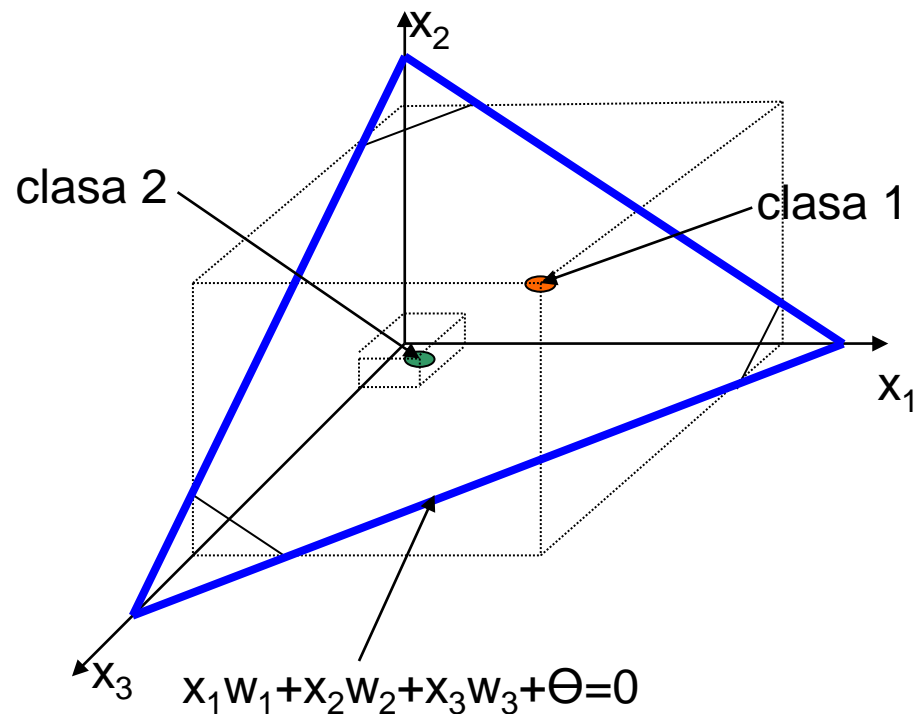
- Se decodifică modelul învățat de RNA
 - prin combinarea ponderilor cu intrările
 - ținând cont de funcțiile de activare a neuronilor și de structura rețelei

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 *ȘI 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 |

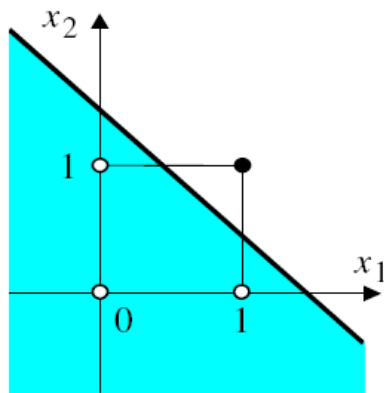
Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Sisteme inteligente – SIS – RNA

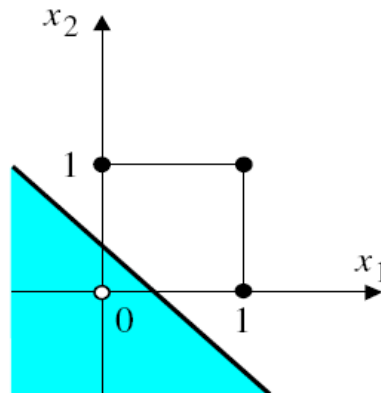
□ Exemplu

■ Perceptron - limitări

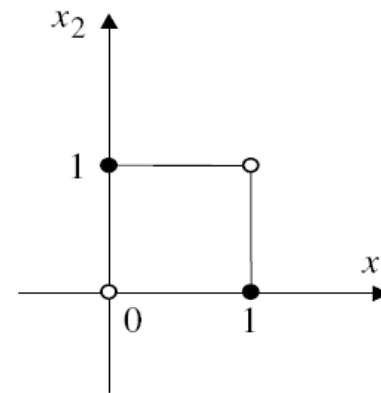
- Un perceptron poate învăța operațiile AND și OR, dar nu poate învăța operația XOR (nu e liniar separabilă)



(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 continuu
- 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 învață 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 → texture → 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 $(\text{attribute_data}_i, \text{label}_i)$, where
 - $i = 1, N$ ($N = \#$ of training data)
 - $\text{attribute_data}_i = (\text{atr}_{i1}, \text{atr}_{i2}, \dots, \text{atr}_{im})$, $m = \#$ attributes (characteristics, features) for an input data
 - $\text{label}_i \in \{\text{Label}_1, \text{Label}_2, \dots, \text{Label}_{\# \text{classes}}\}$
 - Test data – as $(\text{attribute_data}_i)$, $i = 1, n$ ($n = \#$ 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 $x < \text{theta}$
 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

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

A non linear classifier

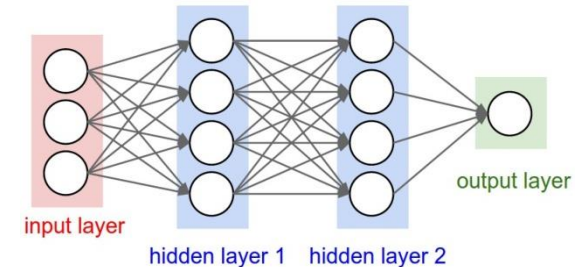
$$\begin{aligned} f(x, w) &= w_2 \max(0, w_1 \cdot x + b_1) + b_2, \\ w_1 &\in \mathbb{R}^{\text{PARAM} \times \text{\#features}} \\ x &\in \mathbb{R}^{\text{\#features} \times 1} \\ b_1 &\in \mathbb{R}^{\text{PARAM}} \\ w_2 &\in \mathbb{R}^{\text{\#classes} \times \text{PARAM}} \\ b_2 &\in \mathbb{R}^{\text{\#classes}}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

Classical ANN

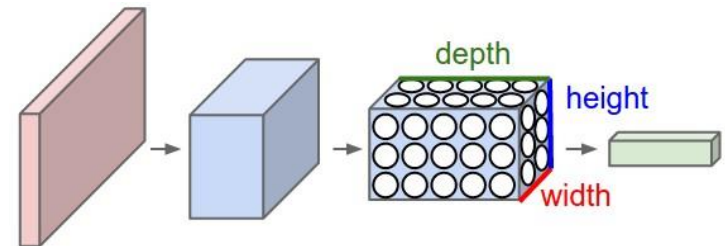
- ❑ Architectures – special graphs with nodes placed on layers
 - Layers
 - ❑ Input layer – size = input's size (#features)
 - ❑ Hidden layers – various sizes (#layers, # neurons/layer)
 - ❑ Output layers – size = output size (e.g. # classes)
 - Topology
 - ❑ Full connected layers (one-way connections, 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 \rightarrow fully connected
 - Conv NNs \rightarrow partially connected
 - connect each neuron to only a local region of the input volume



- Topology of layers
 - Regular NNs \rightarrow linear layers
 - Conv NNs \rightarrow 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
 - $F \times F$ 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_{width} & I_{height})
- an RGB image \rightarrow 3D input (I_{width} , I_{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

- $F_{\text{length}} \ll I_{\text{length}}$

■ a grayscale image \rightarrow 2D filter

- $F_{\text{width}} \ll I_{\text{width}}$ & $F_{\text{height}} \ll I_{\text{height}}$

■ an RGB image \rightarrow 3D filter

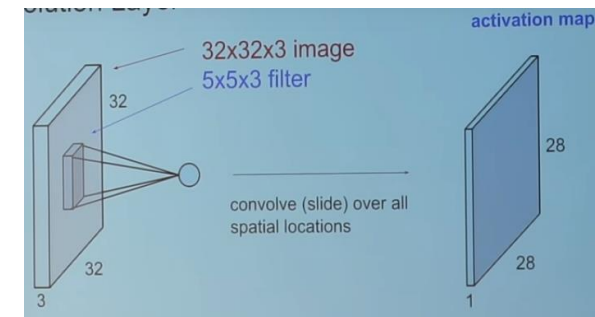
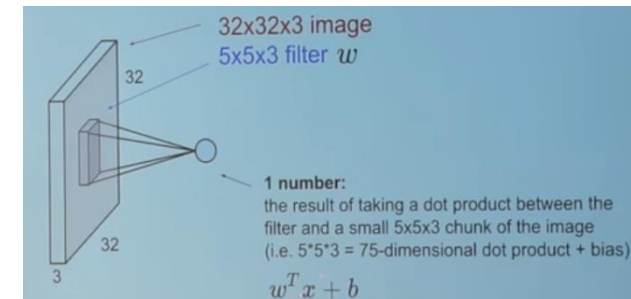
- $F_{\text{width}} \ll I_{\text{width}}$ & $F_{\text{height}} \ll I_{\text{height}}$ &
- $F_{\text{depth}} = I_{\text{depth}} = 3$

■ Apply each filter over the input

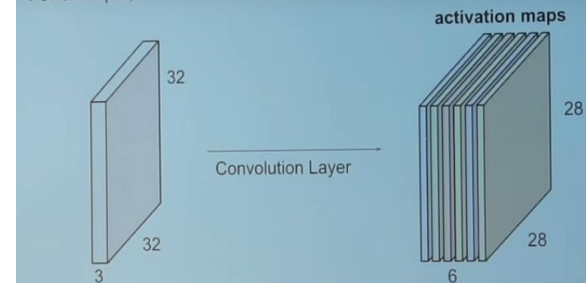
- Overlap filter over a window of the input
 - Stride
 - Padding
- Multiply the filter and the window
- Store the results in an activation map
 - # activation maps = # filters

■ Activate all the elements of each activation map

- ReLU or other activation function



For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



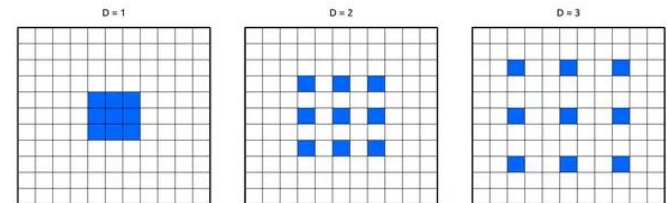
We stack these up to get a "new image" of size 28x28x6!

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 = $(N + 2P - F)/S + 1$

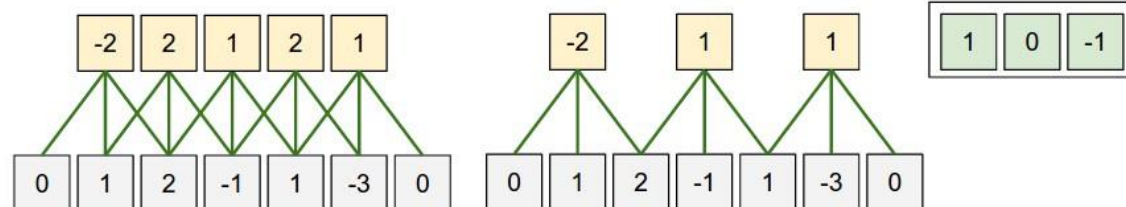
■ Output size

- $K * [(N + 2P - F)/S + 1]$

- $N = L = 5$, $P = 1$,

- $F = 3$, $S = 1$

- $F = 3$, $S = 2$



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 $[227 \times 227 \times 3]$
 - $F=11, S=4, P=0, K=96 \rightarrow$ Conv layer output volume of size $[55 \times 55 \times 96]$
 - $55 \times 55 \times 96 = 290,400$ neurons in the first Conv Layer
 - each has $11 \times 11 \times 3 = 363$ weights and 1 bias.
 - $290400 * 364 = 105,705,600$ parameters on the first layer

Conv NNs

□ Convolutional layer - parameter sharing scheme

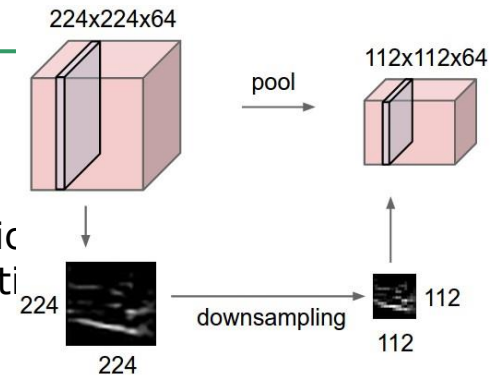
- constrain the neurons in each depth slice to use the same weights and bias
 - detect exactly the same feature, just at different locations in the input image
 - convolutional networks are well adapted to the translation invariance of images
- Example
 - 96 unique set of weights (one for each depth slice),
 - for a total of $96 \cdot 11 \cdot 11 \cdot 3 = 34,848$ 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

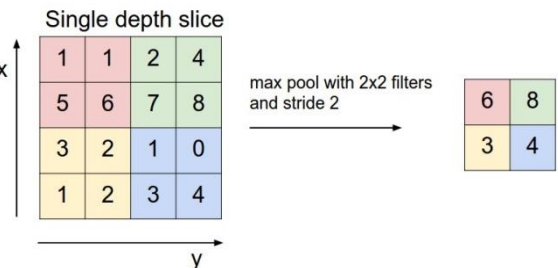
■ Aim

- 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 condenses it into a smaller feature map
- each unit in the pooling layer may summarize a region of the input feature map
- 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)
 - Lp norm $L_p: \sqrt[p]{\sum (X^p)}$
 - 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

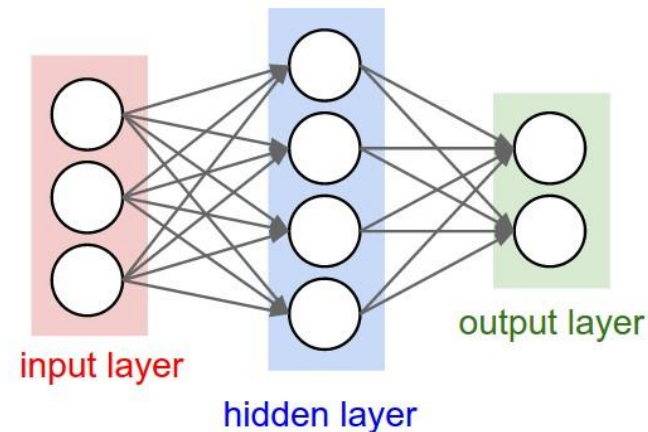
Conv NNs

□ Fully-connected layer

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

■ Various activations

- ReLU (often)



Conv NNs

□ CNN architectures

- INPUT \rightarrow $[[\text{CONV} \rightarrow \text{RELU}]^*N \rightarrow \text{POOL?}]^*M \rightarrow [\text{FC} \rightarrow \text{RELU}]^*K \rightarrow \text{FC}$

■ Most common:

- INPUT \rightarrow FC \leftrightarrow a linear classifier
- INPUT \rightarrow CONV \rightarrow RELU \rightarrow FC
- INPUT \rightarrow $[\text{CONV} \rightarrow \text{RELU} \rightarrow \text{POOL}]^*2 \rightarrow \text{FC} \rightarrow \text{RELU} \rightarrow \text{FC}$.
- INPUT \rightarrow $[\text{CONV} \rightarrow \text{RELU} \rightarrow \text{CONV} \rightarrow \text{RELU} \rightarrow \text{POOL}]^*3 \rightarrow [\text{FC} \rightarrow \text{RELU}]^*2 \rightarrow \text{FC}$
 - 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 \mathbf{z} of arbitrary real values to a $\#classes$ -dimensional vector $\sigma(\mathbf{z})$ of real values in the range $(0, 1)$ that add up to 1
- $\sigma(\mathbf{z})_j = \exp(z_j) / \sum_{k=1..\#classes} \exp(z_k)$

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 $1/\sqrt{\#Inputs}$

- ❑ $w = \text{random}(\#Inputs) / \sqrt{\#Inputs}$

■ Sparse initialization

■ Initializing the biases

■ In practice

- ❑ $w = \text{random}(\#Inputs) * \sqrt{2.0/\#Inputs}$

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> 2020

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/pointeurs/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