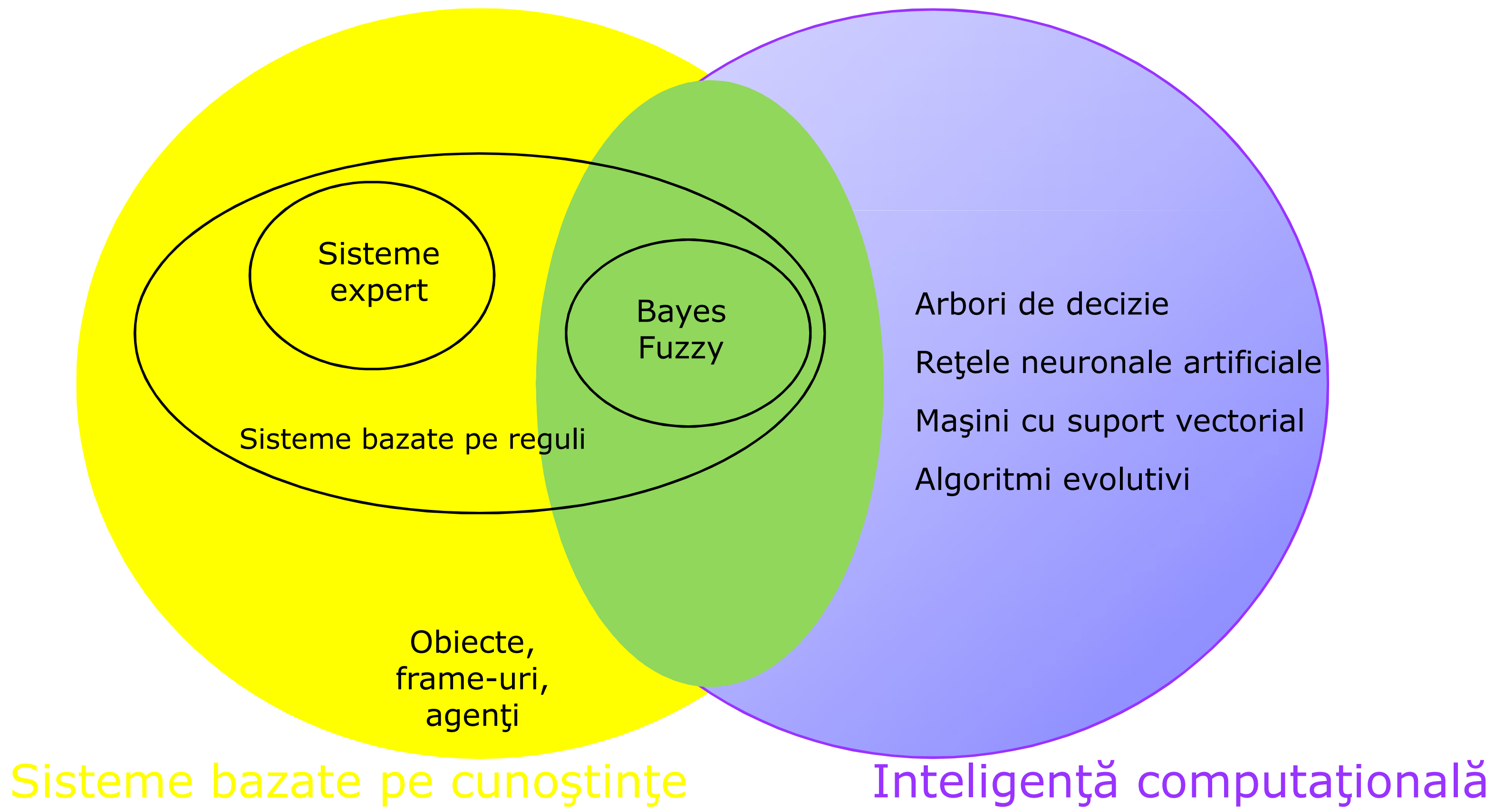
METODE INTELIGENTE DE REZOLVARE A PROBLEMELOR REALE

Laura Dioşan

Tema 4

Sisteme inteligente



Sisteme inteligente – SIS – Învăţare automată

- Tipologie
  - În funcţie de experienţa acumulată în timpul învăţării:
    - SI cu învăţare supervizată
    - SI cu învăţare nesupervizată
    - SI cu învăţare activă
    - SI cu învăţare cu întărire
  - În funcţie de modelul învăţat (algoritmul de învăţare):
    - Arbori de decizie
    - **Reţele neuronale artificiale**
    - Algoritmi evolutivi
    - Maşini cu suport vectorial
    - Modele Markov ascunse

Sisteme inteligente – SIS – RNA

- Reţele 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 → $ax + by + c = 0$ (dacă $m = 2$)
      - un plan → $ax + by + cz + d = 0$ (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?
      - Reţele neuronale artificiale (RNA)
      - Maşini cu suport vectorial (MSV)

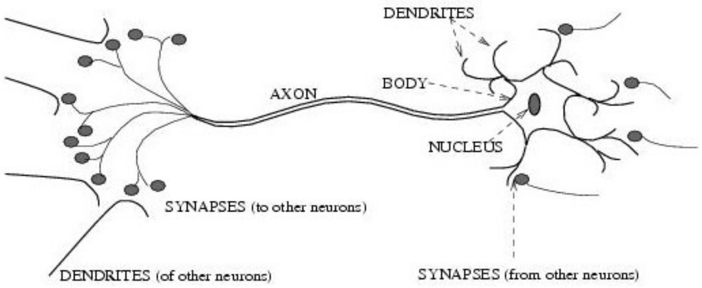
  - De ce RNA?

  - Cum învaţă creierul?

Sisteme inteligente – SIS – RNA

- Scop → De ce RNA?
  - Unele sarcini pot fi efectuate foarte uşor de către oameni, însă sunt greu de codificat sub forma unor algoritmi
    - Recunoaşterea formelor
      - vechi prieteni
      - caractere scrise de mână
      - vocea
    - Diferite raţionamente
      - conducerea autovehiculelor
      - cântatul la pian
      - jucarea baschetului
      - înnotul
  - Astfel de sarcini sunt dificil de definit formal şi este dificilă aplicarea unui proces de raţionare pentru efectuarea lor

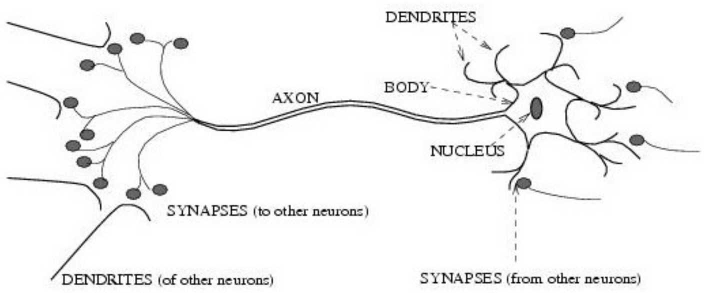
Sisteme inteligente – SIS – RNA



- Scop → Cum învaţă creierul?
  - Creierul uman - componenţă
    - Aproximativ 10.000.000.000 de neuroni conectaţi prin sinapse
    - Fiecare neuron
      - are un corp (soma), un axon şi multe dendrite
      - poate fi într-una din 2 stări:
        - activ – dacă informaţia care intră în neuron depăşeşte un anumit prag de stimulare –
        - pasiv – altfel
    - Sinapsă
      - Legătura între axon-ul unui neuron şi dendritele altui neuron
      - Are rol în schimbul de informaţie dintre neuroni
      - 5.000 de conexiuni / neuron (în medie)
    - În timpul vieţii pot să apară noi conexiuni între neuroni

Sisteme inteligente – SIS – RNA

- Scop → Cum învaţă creierul?
  - Cum "învaţă" (procesează informaţii)?
    - Conexiunile care de-a lungul trăirii unor experienţe s-au dovedit utile devin permanente (restul sunt eliminate)
    - Creierul este interesat de noutăţi
    - Modelul de procesare a informaţiei
      - Învăţare
      - Depozitare
      - Amintire
    - Memoria
      - Tipologie
        - De scurtă durată
        - Imediată → 30 sec.
        - De lucru
        - De lungă durată
      - Capacitate
        - Creşte odată cu vârsta
        - Limitată → învăţarea unei poezii pe strofe
      - Influenţată şi de stările emoţionale
    - Creierul
      - reţea de neuroni
      - sistem foarte complex, ne-liniar şi paralel de procesare a informaţiei
    - Informaţia este depozitată şi procesată de întreaga reţea, nu doar de o anumită parte a reţelei → informaţii şi procesare globală
    - Caracteristica fundamentală a unei reţele de neuroni → învăţarea ➔ reţele neuronale artificiale (RNA)

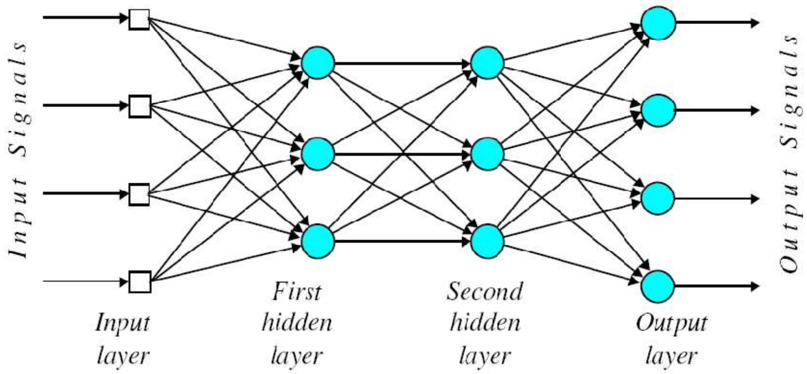


Sisteme inteligente – SIS – RNA

- Definire
  - Ce este o RNA?
  - RN biologice vs. RN artificiale
  - Cum învaţă reţeaua?

Sisteme inteligente – SIS – RNA

- Definire → Ce este o RNA?
  - O structură similară unei reţele neuronale bilogice
  - O mulţime de noduri (unităţi, neuroni, elemente de procesare) dispuse ca într-un graf pe mai multe straturi (*layere*)
    - Nodurile
      - au intrări şi ieşiri
      - efectuează un calcul simplu prin intermediul unei funcţii asociate → funcţie de activare
      - sunt conectate prin legături ponderate
        - Conexiunile între noduri conturează structura (arhitectura) reţelei
        - Conexiunile influenţează calculele care se pot efectua
    - Straturile
      - Strat de intrare
        - Conţine *m* (nr de atribute al unei date) noduri
      - Strat de ieşire
        - Conţine *r* (nr de ieşiri) noduri
      - Straturi intermediare (ascunse) – rol în "complicarea" reţelei
        - Diferite structuri
        - Diferite mărimi



Sisteme inteligente – SIS – RNA

- Definire → RN biologice vs. RN artificiale

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

- Definire → Cum învaţă reţeaua?
  - Plecând de la un set de *n* date de antrenament de forma

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

  cu $p = 1, 2, \ldots, n$, m – nr atributelor, r – nr ieşirilor
  - se formează o RNA cu *m* noduri de intrare, *r* noduri de ieşire şi 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 reţelei prin minimizarea erorii
    - diferenţa între rezultatul real ***y*** şi cel calculat de către reţea

Sisteme inteligente – SIS – RNA

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

Sisteme inteligente – SIS – RNA

- Proiectare
  - Construirea RNA pentru rezolvarea problemei P
  - Iniţializarea 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:
      - $x^d \in \mathbf{R}^m \Rightarrow x^d=(x^d_1, x^d_2,\ldots, x^d_m)$
      - $t^d \in \mathbf{R}^R \Rightarrow t^d=(t^d_1, t^d_2,\ldots, t^d_R)$,
      - cu $d = 1,2,\ldots,n,n+1,n+2,\ldots,N$
    - 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 construieşte o RNA astfel:
      - stratul de intrare conţine exact m noduri (fiecare nod va citi una dintre proprietăţile de intrare ale unei instanţe a problemei – $x^d_1, x^d_2,\ldots, x^d_m$)
      - stratul de ieşire poate conţine R noduri (fiecare nod va furniza una dintre proprietăţile de ieşire ale unei instanţe a problemei $t^d_1, t^d_2,\ldots, t^d_R$)
      - unul sau mai multe straturi ascunse cu unul sau mai mulţi neuroni pe fiecare strat

Sisteme inteligente – SIS – RNA

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

Sisteme inteligente – SIS – RNA

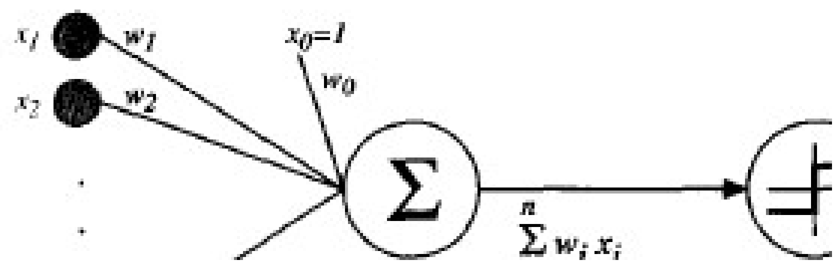
Proiectare

- Iniţializarea parametrilor RNA
  - Iniţializarea ponderile între oricare 2 noduri de pe straturi diferite
  - Stabilirea funcţiei de activare corespunzătoare fiecărui neuron (de pe straturile ascunse)

- Antrenarea (învăţarea) RNA
  - Scop:
    - stabilirea valorii optime a ponderilor dintre 2 noduri
  - Algoritm
    - Se caută valorile optime ale ponderilor între oricare 2 noduri ale reţelei prin minimizarea erorii (diferenţa între rezultatul real y şi cel calculat de către reţea)
  - Cum învaţă reţeaua?
    - Reţeaua = mulţime de unităţi primitive de calcul interconectate între ele ➔
      - Învăţarea reţelei = ∪ învăţarea unităţilor primitive
    - Unităţi primitive de calcul
      - Perceptron
      - Unitate liniară
      - Unitate sigmoidală

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Neuronul ca element simplu de calcul
    - Structura neuronului
      - Fiecare nod are intrări şi ieşiri
      - Fiecare nod efectuează un calcul simplu
    - Procesarea neuronului
      - Se transmite informaţia neuronului
      - Neuronul procesează informaţia
      - Se citeşte răspunsul neuronului
    - Învăţarea neuronului – algoritmul de învăţare a ponderilor care procesează corect informaţiile
      - Se porneşte cu un set iniţial de ponderi oarecare
      - Cât timp nu este îndeplinită o condiţie de oprire
        - Se procesează informaţia şi se stabileşte calitatea ponderilor curente
        - Se modifică ponderile astfel încât să se obţină rezultate mai bune



Sisteme inteligente – SIS – RNA

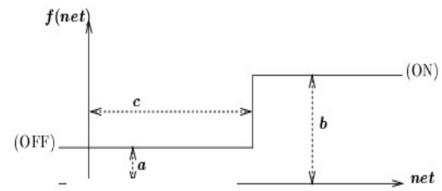
- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Neuronul ca element simplu de calcul
    - Structura neuronului
      - Fiecare nod are intrări şi ieşiri
      - Fiecare nod efectuează un calcul simplu prin intermediul unei funcţii asociate
    - Procesarea neuronului
      - Se transmite informaţia neuronului → se calculează suma ponderată a intrărilor

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

      - Neuronul procesează informaţia → se foloseşte o funcţie de activare:
        - Funcţia constantă
        - Funcţia prag
        - Funcţia rampă
        - Funcţia liniară
        - Funcţia sigmoidală
        - Funcţia Gaussiană
        - Funcția Relu

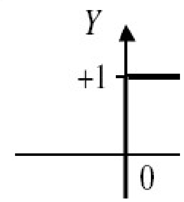
Sisteme inteligente – SIS – RNA

- Proiectare→ Antrenarea RNA → Cum învaţă reţeaua?
  - Funcţia de activare a unui neuron
    - Funcţia constantă f(net) = const
    - Funcţia prag (c - pragul)



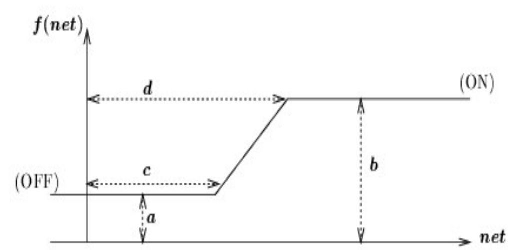
$$f(net) = \begin{cases} a, & \text{dacă } net < c \\ b, & \text{dacă } net > c \end{cases}$$

- Pentru a=+1, b =-1 şi c = 0 ➔ funcţia semn
- Funcţie discontinuă



- Funcţia rampă

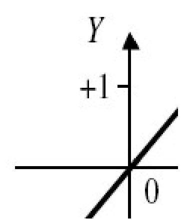
$$f(net) = \begin{cases} a, & \text{dacă } net \le c \\ b, & \text{dacă } net \ge d \\ a + \dfrac{(net - c)(b - a)}{d - c}, & \text{altfel} \end{cases}$$



- Funcţia liniară

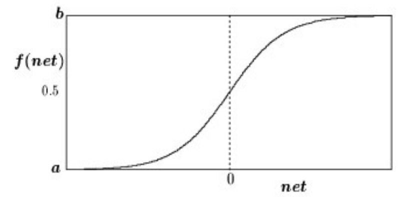
$$f(net) = a * net + b$$

- Pentru a = 1 şi b = 0 ➔ funcţia identitate f(net)=net
- Funcţie continuă



Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Funcţia de activare a unui neuron
    - Funcţia sigmoidală
      - În formă de S
      - Continuă şi diferenţiabilă în orice punct
      - Simetrică rotaţional faţă de un anumit punct (net = c)
      - Atinge asimptotic puncte de saturaţie



$$\lim_{net \to -\infty} f(net) = a \qquad \lim_{net \to \infty} f(net) = b$$

      - Exemple de funcţii sigmoidale:

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

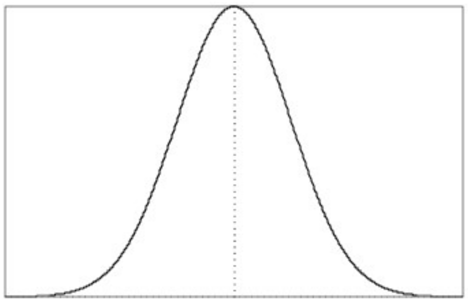
$$f(net) = \tanh(x \cdot net - y) + z \qquad \text{unde} \quad \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

      - Pentru y=0 şi z = 0 → a=0, b = 1, c=0
      - Pentru y=0 şi z = -0.5 → 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 învaţă reţeaua?
  - Funcţia de activare a unui neuron
    - Funcţia Gaussiană
      - În formă de clopot
      - Continuă
      - Atinge asimptotic un punct de saturaţie

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

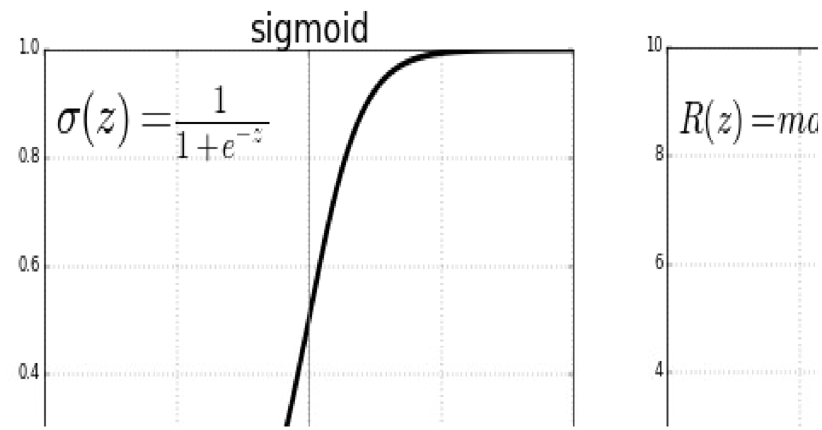


      - Are un singur punct de optim (maxim) – atins când net = μ
      - Exemplu

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

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Funcţia de activare a unui neuron
    - Funcţia ReLU
      - În formă de rampă
      - Continuă, monotonă
      - Derivata ei este monotonă
      - Codomeniu pozitiv [0, ∞)

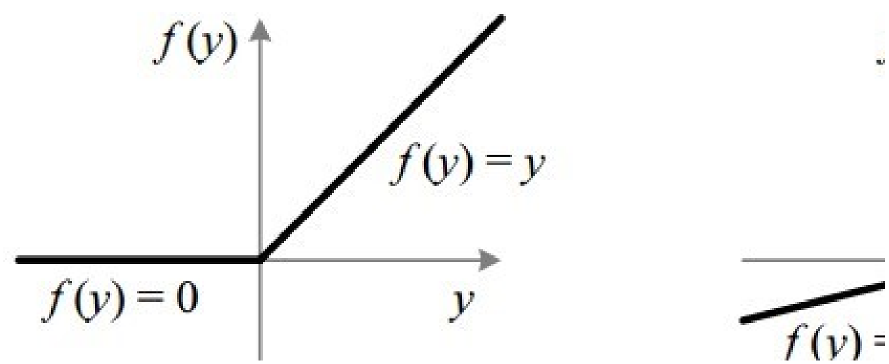


$$f(net) = \max(0, net)$$

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

- Variantă: Leaky ReLU
  - Compensează problemele cu argumentele negative dint ReLU

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



Sisteme inteligente – SIS – RNA

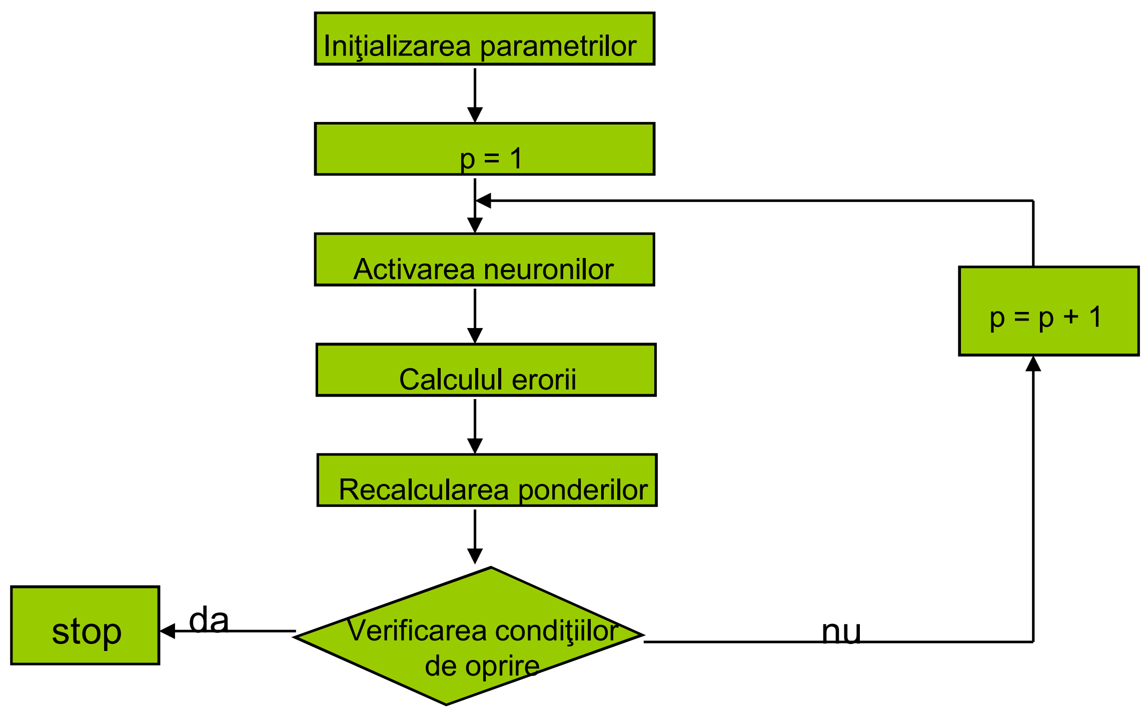
- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Neuronul ca element simplu de calcul
    - Structura neuronului
    - Procesarea neuronului
      - Se transmite informaţia neuronului → se calculează suma ponderată a intrărilor

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

      - Neuronul procesează informaţia → se foloseşte o funcţie de activare:
        - Funcţia constantă
        - Funcţia prag
        - Funcţia rampă
        - Funcţia liniară
        - Funcţia sigmoidală
        - Funcţia Gaussiană
      - Se citeşte răspunsul neuronului → se stabileşte dacă rezultatul furnizat de neuron coincide sau nu cu cel dorit (real)

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă reţeaua?
  - Neuronul ca element simplu de calcul
    - Structura neuronului
    - Procesarea neuronului
    - Învăţarea neuronului
      - Algoritm



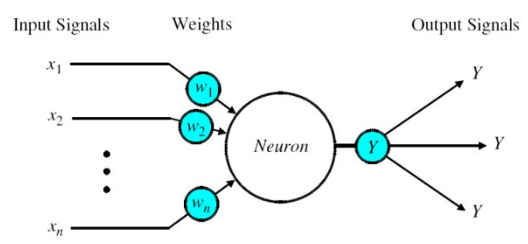
Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - 2 reguli de bază
      - Regula perceptronului → algoritmul perceptronului
        1. Se porneste cu un set de ponderi oarecare
        2. Se stabileşte calitatea modelului creat pe baza acestor ponderi pentru **UNA** dintre datele de intrare
        3. Se ajustează ponderile în funcţie 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 stabileşte calitatea modelului creat pe baza acestor ponderi pentru **TOATE** dintre datele de intrare
        3. Se ajustează ponderile în funcţie 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 stabileşte în funcţie de toate datele de intrare (tot setul de antrenament)

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - Pp că avem un set de date de antrenament de forma:
      - (x^d, t^d), cu:
        - $x^d \in \mathbf{R}^m$ ➔ $x^d=(x^d_1, x^d_2, \ldots, x^d_m)$
        - $t^d \in \mathbf{R}^R$ ➔ $t^d=(t^d_1, t^d_2, \ldots, t^d_R)$, şi R = 1 (adică $t^d=(t^d_1)$)
        - cu d = 1,2,...,n
      - RNA = unitate primitivă de calcul (un neuron) ➔ o reţea cu:
        - m noduri de intrare
        - legate de neuronul de calcul prin ponderile w_i, i =1,2,...,m şi
        - cu un nod de ieşire



Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - Algoritmul perceptronului
      - Se bazează pe minimizarea erorii asociată unei instanţe din setul de date de antrenament
      - Modificarea ponderilor pe baza erorii asociate unei instanţe din setul de antrenament

Iniţializare ponderi din reţea

    w_i = random(a,b), unde i=1,2,...,m

d = 1

Cât timp mai există exemple de antrenament clasificate incorect

    Se activează neuronul şi se calculează ieşirea

        Perceptron → funcţia de activare este funcţia semn (funcţie prag de tip discret, nediferenţiabil)

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

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

        unde η - rată de învăţare

    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 învaţă RNA?
  - Învăţarea neuronului
    - Algoritmul scădere după gradient
      - Se bazează pe eroarea asociată întregului set de date de antrenament
      - Modificarea ponderilor în direcţia dată de cea mai abruptă pantă a reducerii erorii E(**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 funcţie de w (se stabileşte gradientul erorii E)

$$\nabla E(\mathbf{w}) = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_m}\right)$$

      - Gradientul erorii E se calculează în funcţie de funcţia de activare a neuronului (care trebuie să fie diferenţiabilă, deci continuă)
        - Funcţia liniară $f(net) = \sum_{i=1}^{m} w_i x_i^d$
        - Funcţia sigmoidală $f(net) = \frac{1}{1+e^{-\mathbf{wx}}} = \frac{1}{1+e^{-\sum_{i=1}^{m} w_i x_i^d}}$
      - Cum se ajustează ponderile?

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

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - Algoritmul scădere după gradient → calcularea gradientului erorii
      - Funcţia liniară

$$f(net)=\sum_{i=1}^{m} w_i x_i^d$$

$$\frac{\partial E}{\partial w_i}=\frac{\partial \frac{1}{2}\sum_{d=1}^{n}\left(t^d-o^d\right)^2}{\partial w_i}=\frac{1}{2}\sum_{d=1}^{n}\frac{\partial\left(t^d-o^d\right)^2}{\partial w_i}=\frac{1}{2}\sum_{d=1}^{n}2\left(t^d-o^d\right)\frac{\partial\left(t^d-\mathbf{wx}^d\right)}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i}=\sum_{d=1}^{n}\left(t^d-o^d\right)\frac{\partial\left(t^d-w_1x_1^d-w_2x_2^d-\ldots-w_mx_m^d\right)}{\partial w_i}=\sum_{d=1}^{n}\left(t^d-o^d\right)\left(-x_i^d\right)$$

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

      - Funcţie sigmoidală

$$f(net)=\frac{1}{1+e^{-\mathbf{wx}}}=\frac{1}{1+e^{-\sum_{i=1}^{m}w_ix_i^d}}$$

$$y=s(z)=\frac{1}{1+e^{-z}}\Rightarrow\frac{\partial s(z)}{\partial z}=s(z)\left(1-s(z)\right)$$

$$\frac{\partial E}{\partial w_i}=\frac{\partial \frac{1}{2}\sum_{d=1}^{n}\left(t^d-o^d\right)^2}{\partial w_i}=\frac{1}{2}\sum_{d=1}^{n}\frac{\partial\left(t^d-o^d\right)^2}{\partial w_i}=\frac{1}{2}\sum_{d=1}^{n}2\left(t^d-o^d\right)\frac{\partial\left(t^d-sig(\mathbf{wx}^d)\right)}{\partial w_i}=\sum_{d=1}^{n}\left(t^d-o^d\right)\left(1-o^d\right)o^d\left(-x_i^d\right)$$

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

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului
    - Algoritmul scădere după gradient (ASG)

ASG simplu	ASG stocastic
Iniţializare ponderi din reţea w_i = random(a,b), unde i=1,2,...,m d = 1 Cât timp nu este îndeplinită condiţia de oprire Δw_i=0, unde i=1,2,...,m Pentru fiecare exemplu de antrenament (x^d,t^d), unde d=1,2,...,n Se activează neuronul si se calculează ieşirea o^d funcţia de activare = funcţia liniară → $o^d=\mathbf{w}\mathbf{x}^d$ funcţia de activare = funcţia sigmoid → $o^d=sig(\mathbf{w}\mathbf{x}^d)$ Pentru fiecare pondere w_i, unde i =1,2,...,m Se stabileşte ajustarea ponderii $\Delta w_i = \Delta w_i - \eta \frac{\partial E}{\partial w_i}$ Pentru fiecare pondere w_i, unde i =1,2,...,m Se ajustează fiecare pondere w_i $w_i = w_i + \Delta w_i$	Iniţializare ponderi din reţea w_i = random(a,b), unde i=1,2,...,m d = 1 Cât timp nu este îndeplinită condiţia de oprire Δw_i=0, unde i=1,2,...,m Pentru fiecare exemplu de antrenament (x^d,t^d), unde d=1,2,...,n Se activează neuronul si se calculează ieşirea o^d funcţia de activare = funcţia liniară → $o^d=\mathbf{w}\mathbf{x}^d$ funcţia de activare = funcţia sigmoid → $o^d=sig(\mathbf{w}\mathbf{x}^d)$ Pentru fiecare pondere w_i, unde i =1,2,...,m Se stabileşte ajustarea ponderilor $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$ Se ajustează ponderea w_i $w_i = w_i + \Delta w_i$

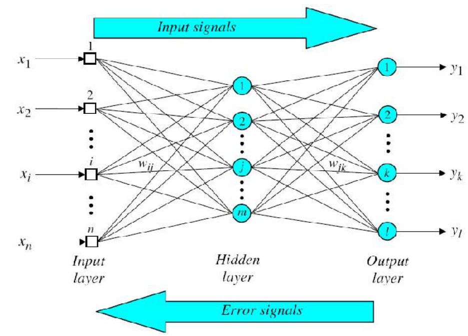
Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă RNA?
  - Învăţarea neuronului

Diferenţe	**Algoritmul perceptronului**	**Algoritmul scădere după gradient (regula Delta)**
Ce reprezintă o^d	$o^d=\text{sign}(\mathbf{wx}^d)$	$o^d=\mathbf{wx}^d$ sau $o^d=\text{sig}(\mathbf{wx}^d)$
Cum converge	Într-un nr finit de paşi (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 ne-liniar)
Ce tip de ieşire are neuronul	Discretă şi cu prag	Continuă şi fără prag

Sisteme inteligente – SIS – RNA

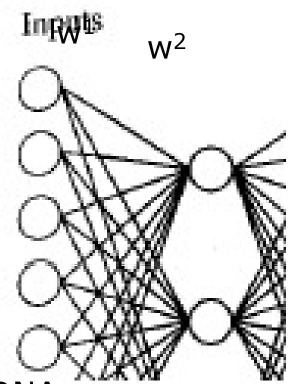
- Proiectare → Antrenarea RNA
  - Cum învaţă reţeaua?
    - Reţeaua = mulţime de unităţi primitive de calcul interconectate între ele ➔
      - Învăţarea reţelei = ∪ învăţarea unităţilor primitive
    - Reţea cu mai mulţi neuroni aşezaţi pe unul sau mai multe straturi ➔ RNA este capabilă să înveţe un model mai complicat (nu doar liniar) de separare a datelor
    - Algoritmul de învăţare a ponderilor ➔ backpropagation
      - Bazat pe algoritmul scădere după gradient
      - Îmbogăţit cu:
        - Informaţia se propagă în RNA înainte (dinspre stratul de intrare spre cel de ieşire)
        - Eroarea se propagă în RNA înapoi (dinspre stratul de ieşire spre cel de intrare)



Se iniţializează ponderile
Cât timp nu este îndeplinită condiţia de oprire
    Pentru fiecare exemplu (x^d,t^d)
        Se activează fiecare neuron al reţelei
            Se propagă informaţia înainte şi se calculează ieşirea corespunzătoare fiecărui neuron al reţelei
        Se ajustează ponderile
            Se stabileşte şi se propagă eroarea înapoi
                Se stabilesc erorile corespunzătoare neuronilor din stratul de ieşire
                Se propagă aceste erori înapoi în toată reţeaua → se distribuie erorile pe toate conexiunile existente în reţea proporţional cu valorile ponderilor asociate acestor conexiuni
            Se modifică ponderile

Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA
  - Cum învaţă o întreagă RNA?
    - Pp că avem un set de date de antrenament de forma:
      - (x^d, t^d), cu:
        - $x^d \in \mathbf{R}^m$ → $x^d=(x^d_1, x^d_2,..., x^d_m)$
        - $t^d \in \mathbf{R}^R$ → $t^d=(t^d_1, t^d_2,..., t^d_R)$
        - cu d = 1,2,...,n
    - Presupunem 2 cazuri de RNA
      - O RNA cu un singur strat ascuns cu H neuroni → RNA_1
        - m neuroni pe stratul de intrare,
        - R neuroni pe stratul de ieşire,
        - H neuroni pe stratul ascuns
        - Ponderile între stratul de intrare şi cel ascuns w_{ih}^1 cu i=1,2,...,m, h = 1,2,...,H
        - Ponderile între stratul ascuns şi cel de ieşire w_{hr}^2 cu h = 1,2,...,H şi r = 1,2,...,R
      - O RNA cu p straturi ascunse, fiecare strat cu Hi (i =1,2,...,p) neuroni → RNA_p
        - m neuroni pe stratul de intrare,
        - R neuroni pe stratul de ieşire,
        - P straturi ascunse
        - Hp neuroni pe stratul ascuns p, p =1,2,...,P
        - Ponderile între stratul de intrare şi primul strat ascuns $w_{ih_1}^1$ cu i=1,2,...,m, h1 = 1,2,...,H1
        - Ponderile între primul strat ascuns şi cel de-al doilea strat ascuns $w_{h_1h_2}^2$ cu h1 = 1,2,...,H1 şi h2 = 1,2,...,H2
        - Ponderile între cel de-al doilea strat ascuns şi cel de-al treilea strat ascuns $w_{h_2h_3}^3$ cu h2 = 1,2,...,H2 şi h3 = 1,2,...,H3
        - ...
        - Ponderile între cel de-al p-1 strat ascuns şi ultimul strat ascuns $w_{h_{p-1}h_p}^p$ cu hp-1 = 1,2,...,Hp-1 şi hp = 1,2,...,Hp
        - Ponderile între ultimul strat ascuns şi cel de ieşire $w_{h_pr}^{p+1}$ cu hp = 1,2,...,Hp şi r = 1,2,...,R



Sisteme inteligente – SIS – RNA

- Proiectare → Antrenarea RNA → Cum învaţă o întreagă RNA?
  - Algoritmul backpropagation pentru RNA_1

Se iniţializează ponderile w_{ih}^1 şi w_{hr}^2 cu i=1,2,...,m, h = 1,2,...,H şi r = 1,2,...,R

Cât timp nu este îndeplinită condiţia de oprire

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

Se activează fiecare neuron al reţelei

Se propagă informaţia înainte şi se calculează ieşirea corespunzătoare fiecărui neuron al reţelei

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

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

Se ajustează ponderile

Se stabileşte şi se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieşire

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

Se modifică ponderile între nodurile de pe stratul ascuns şi stratul de ieşire

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

Se propagă erorile nodurilor de pe stratul de ieşire înapoi în toată reţeaua → se distribuie erorile pe toate conexiunile existente în reţea proporţional cu valorile ponderilor asociate acestor conexiuni

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

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

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

Sisteme inteligente – SIS – RNA

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

Se iniţializează ponderile $w^1_{ih_1}, w^2_{h_1h_2}, \ldots, w^p_{h_{p-1}h_p}, w^{p+1}_{h_pr}$

Cât timp nu este îndeplinită condiţia de oprire

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

Se activează fiecare neuron al reţelei

Se propagă informaţia înainte şi se calculează ieşirea corespunzătoare fiecărui neuron al reţelei

$$o^d_{h_1} = \sum_{i=1}^{m} w^1_{ih_1} x^d_i \text{ sau } o^d_{h_1} = sig\left(\sum_{i=1}^{m} w^1_{ih_1} x^d_i\right), \text{cu } h_1 = 1,2,\ldots,H_1$$

$$o^d_{h_2} = \sum_{h_1=1}^{H_1} w^2_{h_1h_2} o^d_{h_1} \text{ sau } o^d_{h_2} = sig\left(\sum_{h_1=1}^{H_1} w^2_{h_1h_2} o^d_{h_1}\right), \text{cu } h_2 = 1,2,\ldots,H_2$$

...

$$o^d_{h_p} = \sum_{h_{p-1}=1}^{H_{p-1}} w^p_{h_{p-1}h_p} o^d_{h_{p-1}} \text{ sau } o^d_{h_p} = sig\left(\sum_{h_{p-1}=1}^{H_{p-1}} w^p_{h_{p-1}h_p} o^d_{h_{p-1}}\right), \text{cu } h_p = 1,2,\ldots,H_p$$

$$o^d_r = \sum_{h_p=1}^{H_p} w^{p+1}_{h_pr} o^d_{h_p} \text{ sau } o^d_r = sig\left(\sum_{h_p=1}^{H_p} w^{p+1}_{h_pr} o^d_{h_p}\right), \text{cu } r = 1,2,\ldots,R$$

Sisteme inteligente – SIS – RNA

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

Se iniţializează ponderile $w^1_{ih_1}, w^2_{h_1h_2}, \ldots, w^p_{h_{p-1}h_p}, w^{p+1}_{h_pr}$

Cât timp nu este îndeplinită condiţia de oprire

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

Se activează fiecare neuron al reţelei

Se ajustează ponderile

Se stabileşte şi se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieşire

$$\delta^d_r = t^d_r - o^d_r \text{ sau } \delta^d_r = o^d_r(1-o^d_r)(t^d_r - o^d_r), \text{ cu } r = 1,2,\ldots,R$$

Se modifică ponderile între nodurile de pe ultimul strat ascuns şi stratul de ieşire

$$w^{p+1}_{h_pr} = w^{p+1}_{h_pr} + \eta\delta^d_r o^d_{h_p}, \text{ unde } \mathrm{h_p} = 1,2,\ldots,H_p \text{ şi } \mathrm{r} = 1,2,\ldots,R$$

Sisteme inteligente – SIS – RNA

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

Se iniţializează ponderile $w^1_{ih_1}, w^2_{h_1h_2}, \ldots, w^p_{h_{p-1}h_p}, w^{p+1}_{h_pr}$

Cât timp nu este îndeplinită condiţia de oprire

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

Se activează fiecare neuron al reţelei

Se ajustează ponderile

Se stabileşte şi se propagă eroarea înapoi

Se stabilesc erorile corespunzătoare neuronilor din stratul de ieşire

Se modifică ponderile între nodurile de pe ultimul strat ascuns şi stratul de ieşire

Se propagă (pe starturi) aceste erori înapoi în toată reţeaua → se distribuie erorile pe toate conexiunile existente în reţea proporţional cu valorile ponderilor asociate acestor conexiuni şi se modifică ponderile corespunzătoare

$$\delta^d_{h_p} = \sum_{r=1}^{R} w^{p+1}_{h_pr}\delta^d_r \text{ sau } \delta^d_{h_p} = o^d_{h_p}\left(1-o^d_{h_p}\right)\sum_{r=1}^{R} w^{p+1}_{h_pr}\delta^d_r$$

$$w^{p+1}_{h_pr} = w^{p+1}_{h_pr} + \eta\delta^d_r o^d_{h_p}, \text{ unde } \mathrm{h_p} = 1,2,\ldots,H_p \text{ şi } \mathrm{r} = 1,2,\ldots,R$$

$$\delta^d_{h_{p-1}} = \sum_{h_p=1}^{H_p} w^p_{h_{p-1}h_p}\delta^d_{h_p} \text{ sau } \delta^d_{h_{p-1}} = o^d_{h_{p-1}}\left(1-o^d_{h_{p-1}}\right)\sum_{h_p=1}^{H_p} w^p_{h_{p-1}h_p}\delta^d_{h_p}$$

$$w^p_{h_{p-1}h_p} = w^p_{h_{p-1}h_p} + \eta\delta^d_{h_p} o^d_{h_{p-1}}, \text{ unde } h_{\mathrm{p-1}} = 1,2,\ldots,H_{p-1} \text{ şi } h_p = 1,2,\ldots,H_p$$

…

$$\delta^d_{h_1} = \sum_{h_2=1}^{H_2} w^2_{h_1h_2}\delta^d_{h_2} \text{ sau } \delta^d_{h_1} = o^d_{h_1}\left(1-o^d_{h_1}\right)\sum_{h_2=1}^{H_2} w^2_{h_1h_2}\delta^d_{h_2}$$

$$w^1_{ih_1} = w^1_{ih_1} + \eta\delta^d_{h_1} x^d_i, \text{ unde } i = 1,2,\ldots,m \text{ şi } h_1 = 1,2,\ldots,H_1$$

Sisteme inteligente – SIS – RNA

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

Sisteme inteligente – SIS – RNA

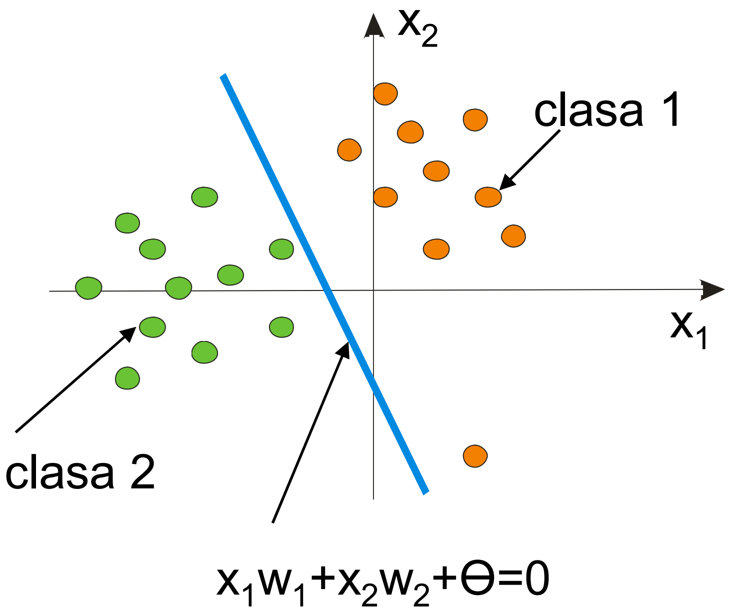
- Proiectare
  - Construirea RNA pentru rezolvarea problemei P
  - Iniţializarea parametrilor RNA
  - Antrenarea RNA
  - **Testarea RNA**

Sisteme inteligente – SIS – RNA

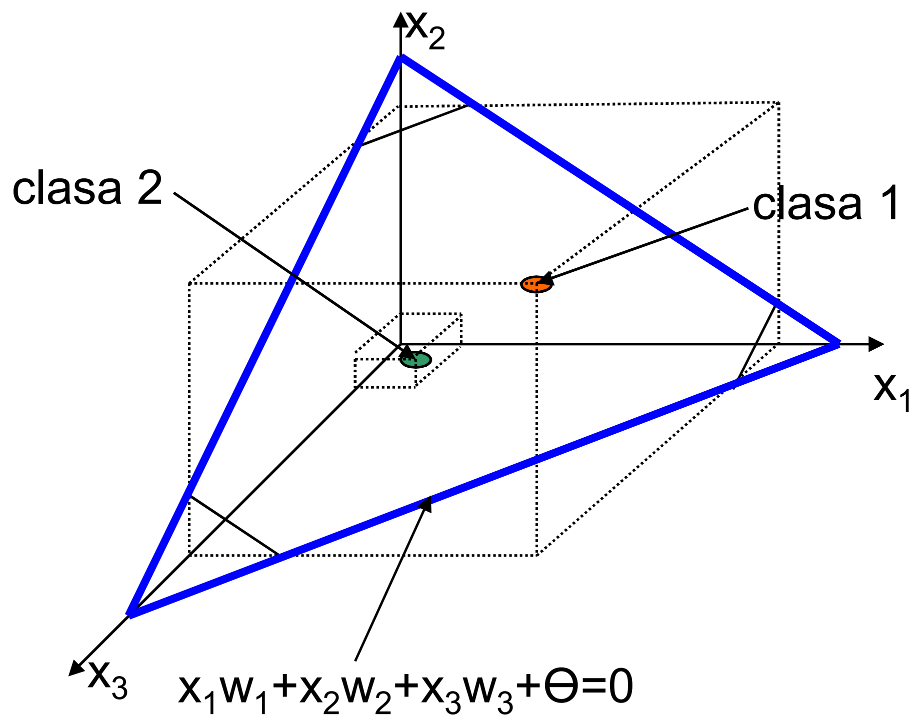
- Proiectare
  - Testarea RNA
    - Se decodifică modelul învăţat de RNA
      - prin combinarea ponderilor cu intrările
      - ţinând cont de funcţiile de activare a neuronilor şi de structura reţelei

Sisteme inteligente – SIS – RNA

- Exemplu



Clasificare binară cu m=2 intrări



Clasificare binară cu m=3 intrări

Sisteme inteligente – SIS – RNA

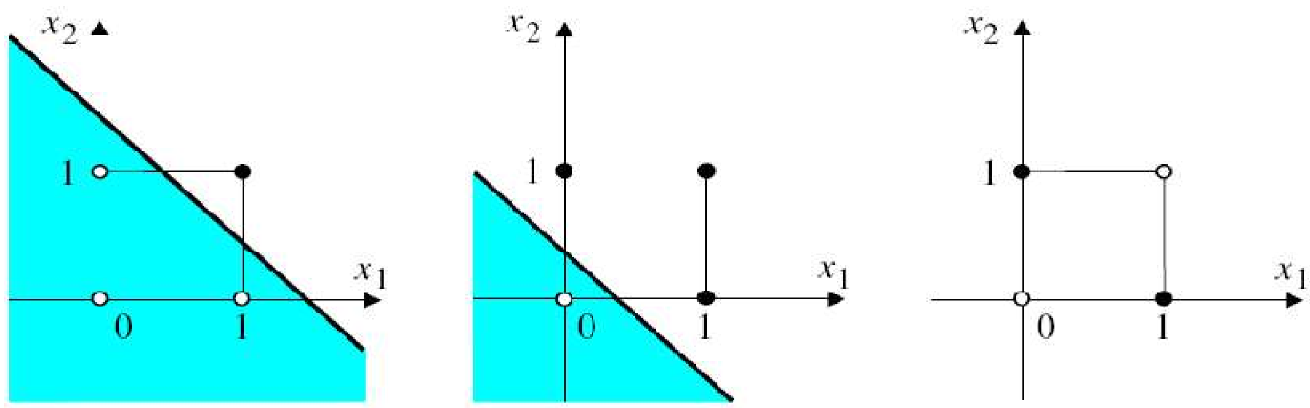
- Exemplu
  - Perceptron pentru rezolvarea problemei *ŞI logic*

Epoch	Inputs x_1	Inputs x_2	Desired output Y_d	Initial weights w_1	Initial weights w_2	Actual output Y	Error e	Final weights w_1	Final weights 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 învăţa operaţiile AND şi OR, dar nu poate învăţa operaţia 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
  - soluţii
    - Neuron cu un prag continu
    - Mai mulţi neuroni

Sisteme inteligente – SIS – RNA

- Tipologie
  - RNA feed-forward
    - Informaţia se procesează şi circulă de pe un strat pe altul
    - Conexiunile între noduri nu formează cicluri
    - Se folosesc în special pentru învăţarea supervizată
    - Funcţiile de activare a nodurilor → liniare, sigmoidale, gaussiene
  - RNA recurente (cu feedback)
    - Pot conţine conexiuni între noduri de pe acelaşi strat
    - Conexiunile între noduri pot forma cicluri
    - RNA de tip Jordan
    - RNA de tip Elman
    - RNA de tip Hopfield

      } pentru învăţarea supervizată
    - RNA auto-organizate → pentru învăţarea nesupervizată
      - De tip Hebbian
      - De tip Kohonen (*Self organised maps*)

Sisteme inteligente – SIS – RNA

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

- Dificultăţi şi limite
  - RNA se confruntă cu problema overfitting-ului chiar şi când modelul se învaţă prin validare încrucişată
  - 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)
      - $attribute\_data_i = (atr_{i1}, atr_{i2}, \ldots, atr_{im})$, m – # attributes (characteristics, features) for an input data
      - $label_i \in \{Label_1, Label_2, \ldots, Label_{\#classes}\}$
    - 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) $\sum_i \sum_{j, j \neq yi} \max(C_j - D_{yi} + \Delta, 0)$
    - Softmax loss $\sum_i [- \ln(\exp(D_{yi}) / \sum_{j, j \neq yi} \exp(Cj))]$
    - 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 $\max(w^T{}_1x+b_1,w^T{}_2x+b_2)$
  - Exponential linear units (ELU) $f(x) = x$, if $x > 0$
    $\alpha(\exp(x) - 1)$, if $x \le 0$

A linear classifier

$f(x, w) = w \cdot x + b$,

$w \in R^{\#classes \times \#features}$

$x \in R^{\#features \times 1}$

$b \in R^{\#classes}$

A non linear classifier

$f(x, w) = w_2 \max(0, w_1 \cdot x + b_1) + b_2$,

$w1 \in R^{PARAM \times \#features}$

$x \in R^{\#features \times 1}$

$b1 \in R^{PARAM}$

$w2 \in R^{\#classes \times PARAM}$

$b2 \in R^{\#classes}$

Classical ANN

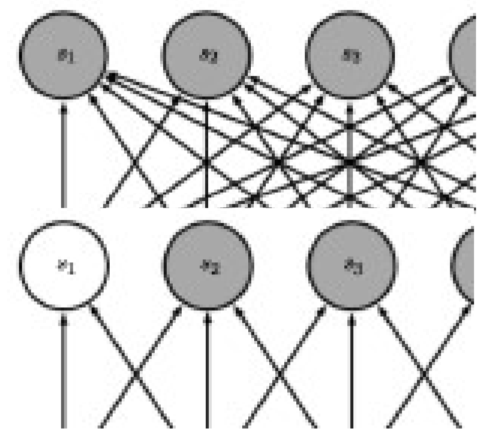
- Architectures – special graphs with nodes placed on layers
  - Layers
    - Input layer – size = input's size (#features)
    - Hidden layers – various sizes (#layers, # neurons/layer)
    - Output layers – size = output size (e.g. # classes)
  - Topology
    - Full connected layers (one-way connections, 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)

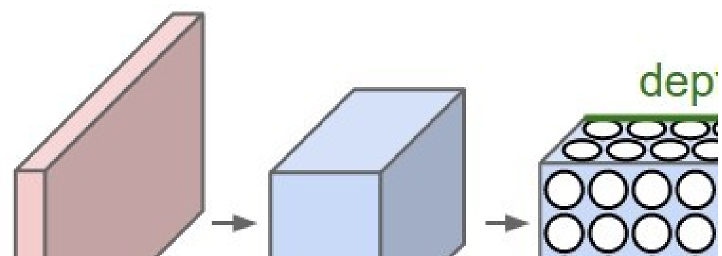
Convolutional Neural Networks

- More layers
- More nodes/layer

- Topology of connections
  - Regular NNs → fully connected
    - O(#inputs x #outputs)
  - Conv NNs → partially connected
    - connect each neuron to only a local region of the input volume
    - O(#someInputs x #outputs)



- Topology of layers
  - Regular NNs → linear layers
  - Conv NNs → 2D/3D layers (width, height, depth)



Conv NNs

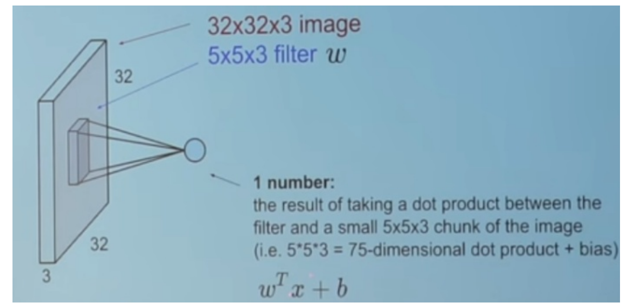
- Layers of a Conv NN
  - Convolutional Layer ➔ feature map
    - Convolution
    - Activation (thresholding)
  - Pooling/Aggregation Layer ➔ size reduction
  - Fully-Connected Layer ➔ answer

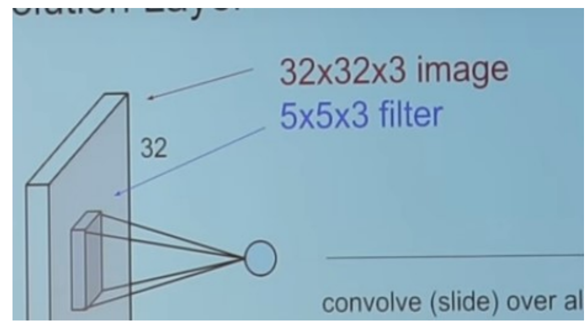
Conv NNs

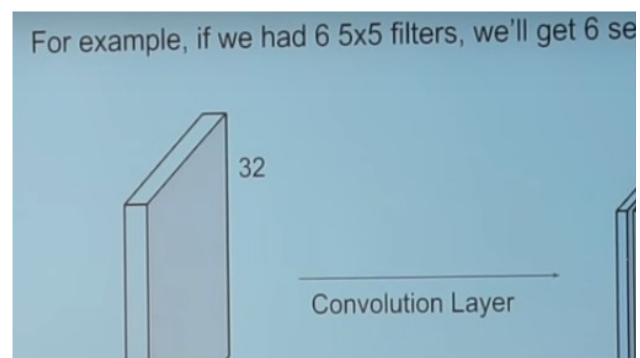
- Convolutional layer
  - Aim
    - *learn data-specific kernels*
    - Perform a liniar operation
  - Filters or Local receptive fields or Kernels
    - Content
      - Convolution (signal theory) vs. Cross-correlation
      - a little (square/cube) window on the input pixels
    - How it works?
      - 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 → 1D input (I_{length})
    - a grayscale image → 2D input (I_{Width} & I_{Height})
    - an RGB image → 3D input (I_{Width}, I_{Height} & I_{Depth} = 3)
  - Consider a set of filters (kernels) $F_1, F_2, \ldots, F_{\#filters}$
    - A filter must have the same # dimensions as the input
      - A signal → 1D filter
        - $F_{length} << I_{length}$
      - a grayscale image → 2D filter
        - $F_{width} << I_{width}$ & $F_{height} << I_{height}$
      - an RGB image → 3D filter
        - $F_{width} << I_{width}$ & $F_{height} << I_{height}$ &
        - $F_{depth} = I_{Depth} = 3$
  - 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



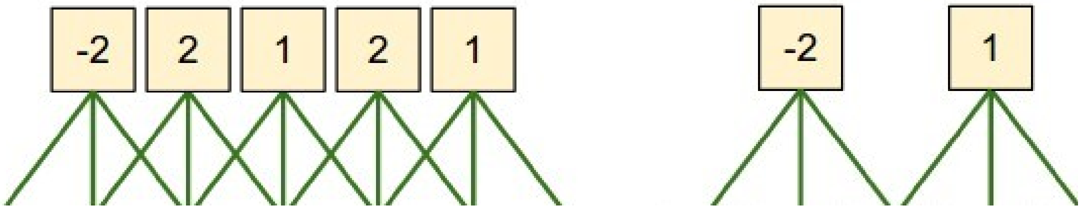




***Images taken from Andrej Karpathy's lectures about Conv NNs

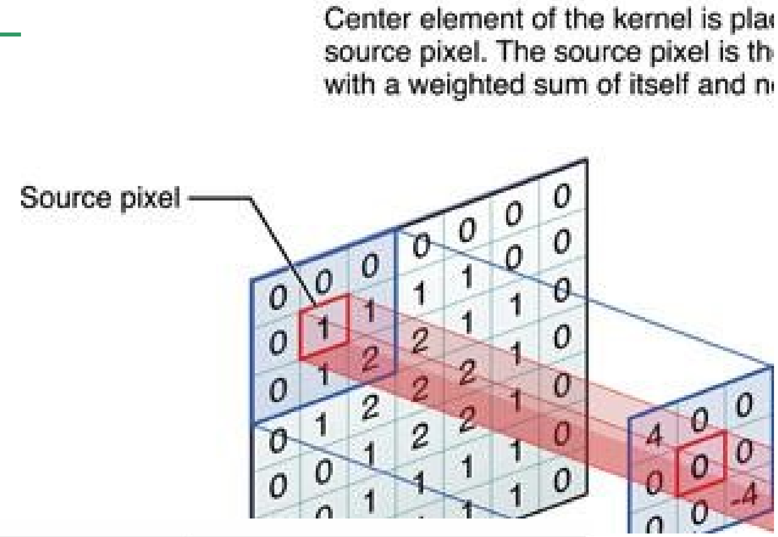
Conv NNs

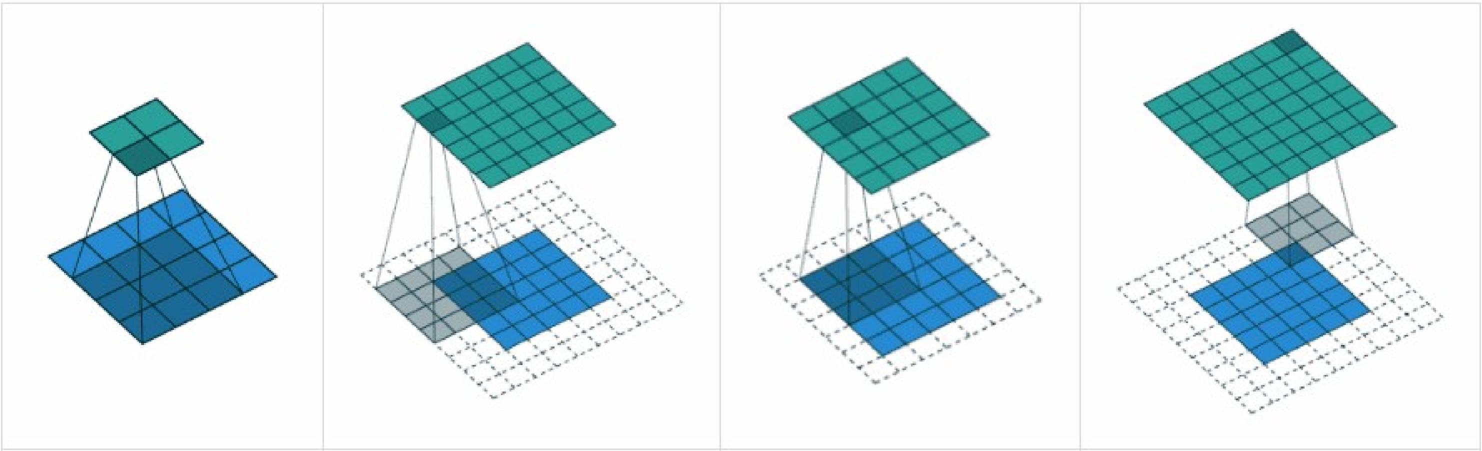
- Convolutional layer - Hyperparameters
  - input volume size N (L or W_I & H_I or W_I & H_I & D_I)
  - size of zero-padding of input volume P (P_L or P_W & P_H or P_W & P_H & P_D)
  - the receptive field size (filter size) F (F_L, F_W & F_H, F_W & F_H & F_D)
  - stride of the convolutional layer S (S_L, S_W & S_H, S_W & S_H & S_D)
  - # of filters (K)
    - depth of the output volume
  - # neurons of an activation map = $(N + 2P - F)/S+1$
  - Output size (O or W_O & H_O or W_O & H_O & D_O)
    - $K * [(N + 2P - F)/S+1]$
  - N = L = 5, P = 1,
    - F = 3, S = 1
    - F = 3, S = 2



Conv NNs

- Convolutional Layer – How does it work?





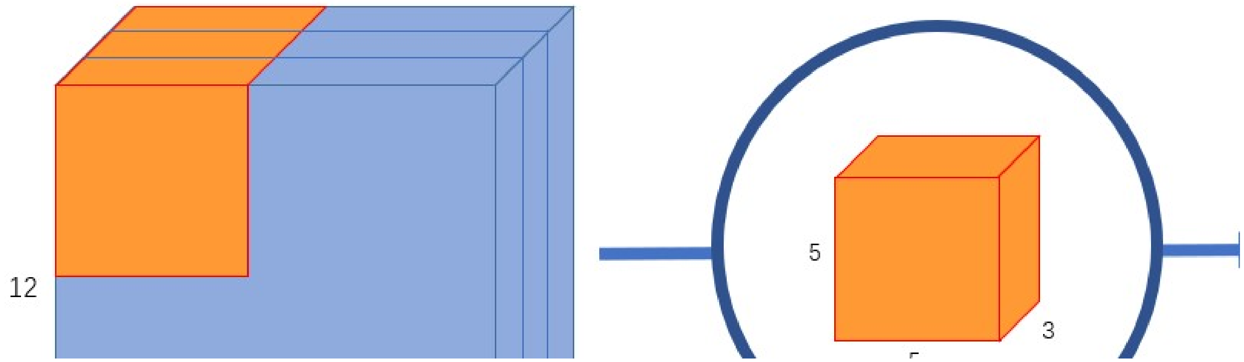
$N = W_I = H_I = 4$, $F = F_W = F_H = 3$, $P = 0$, $S = 1$, $W_O = H_O = 2$	$N = W_I = H_I = 5$, $F = F_W = F_H = 4$, $P = 2$, $S = 1$, $W_O = H_O = 6$	$N = W_I = H_I = 5$, $F = F_W = F_H = 4$, $P = 1$, $S = 1$, $W_O = H_O = 5$	$N = W_I = H_I = 5$, $F = F_W = F_H = 3$, $P = 2$, $S = 1$, $W_O = H_O = 7$

Conv NNs

- Convolutional layer – typology
  - Classic convolution
  - Transposed convolution (deconvolution)
  - Dilated convolution
  - Spatial separable (depthwise separable) convolution
  - Grouped convolutions

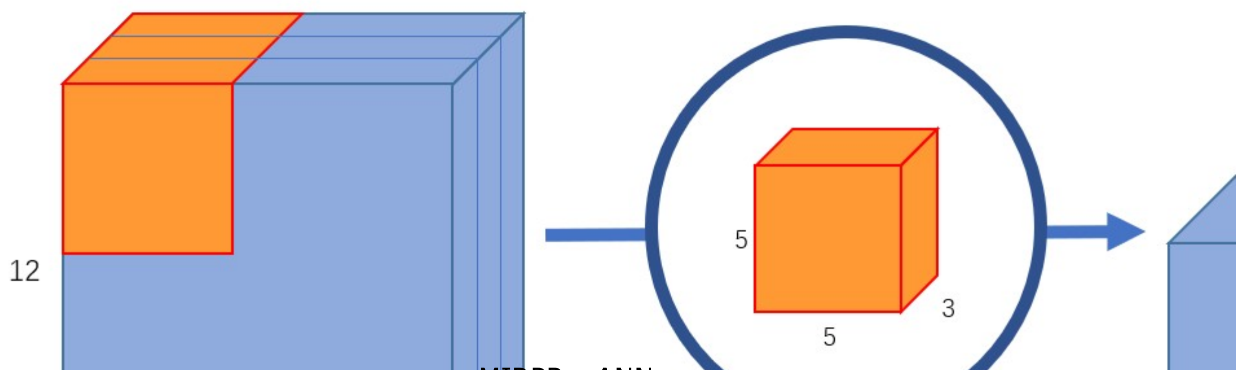
Conv NNs

- Convolutional layer – typology
  - classic convolution
    - one filter, D channels



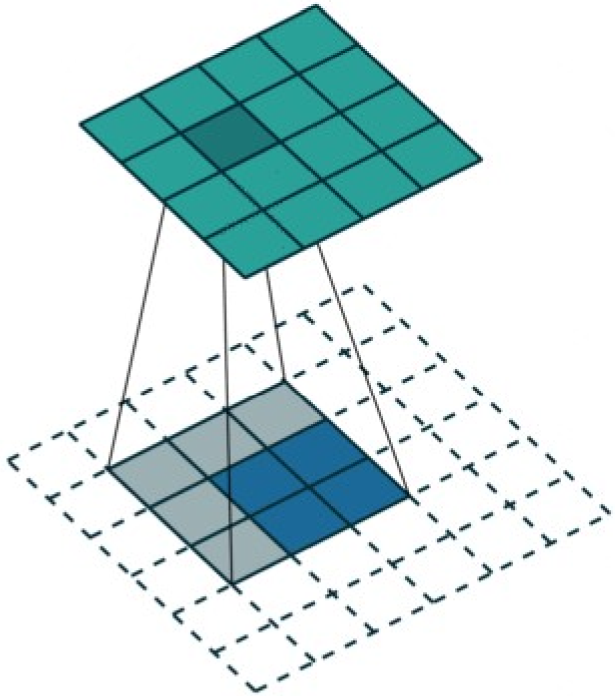
    - more filters (K), D channels

* K = * 256



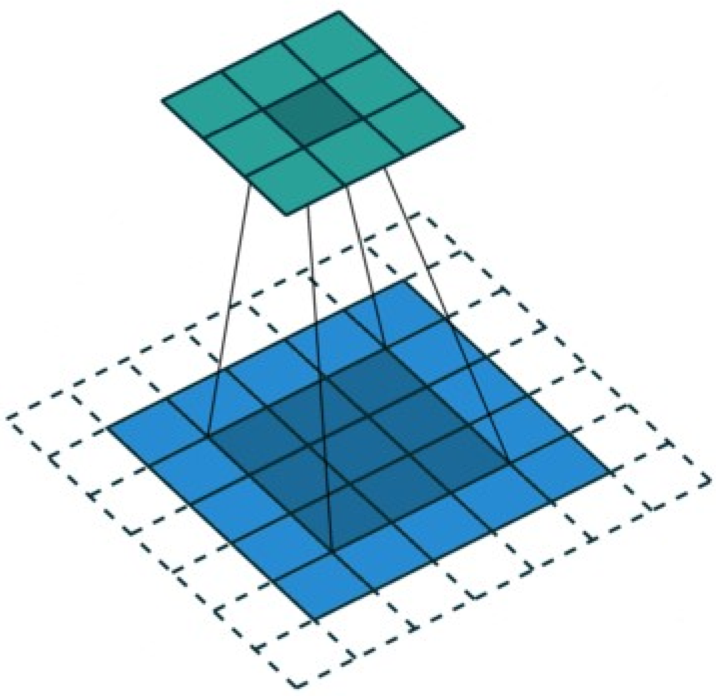
Conv NNs

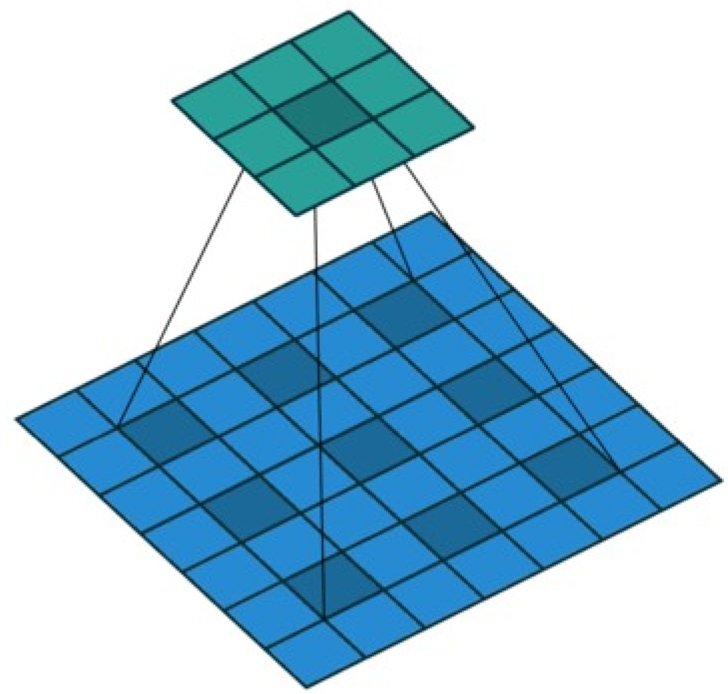
- Convolutional layer – typology
  - Transposed convolution (deconvolution)
    - See https://arxiv.org/pdf/1603.07285.pdf
    - Up-sampling - How?
      - $Img_{Large} * F = Img_{Small}$
      - $Img_{Large} * F * F^T = Img_{Small} * F^T$
      - $Img_{Large} * I = Img_{Small} * F^T$
      - $Img_{Large} = Img_{Small} * F^T$



Conv NNs

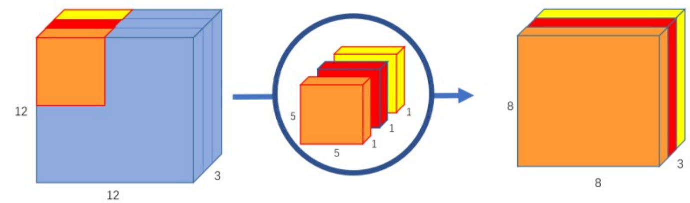
- Convolutional layer – typology
  - Dilated convolutions (atrous convolutions)
    - See https://arxiv.org/pdf/1511.07122.pdf





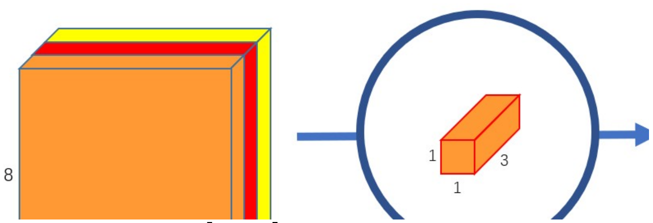
Conv NNs

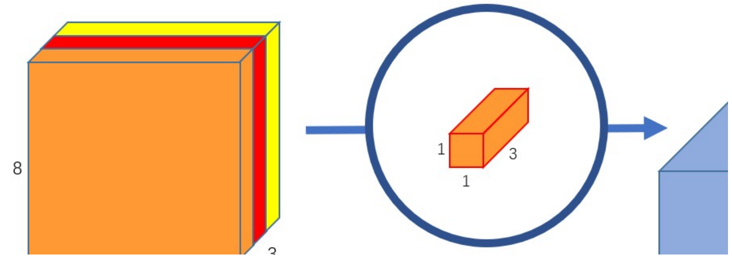
- Convolutional layer – typology
  - Spatial separable convolution (Depth-wise separable convolution)
    - Split the convolution into
      - A depthwise convolution



      - A pointwise convolution

* K = * 256



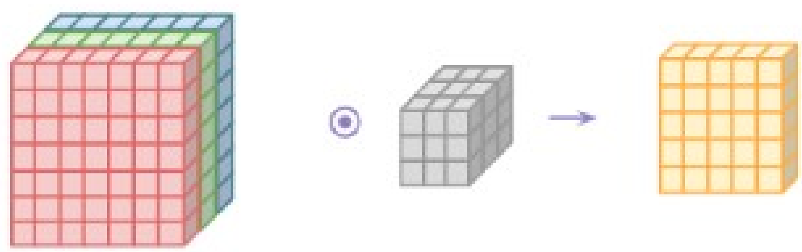


    - E.g. Sobel operator

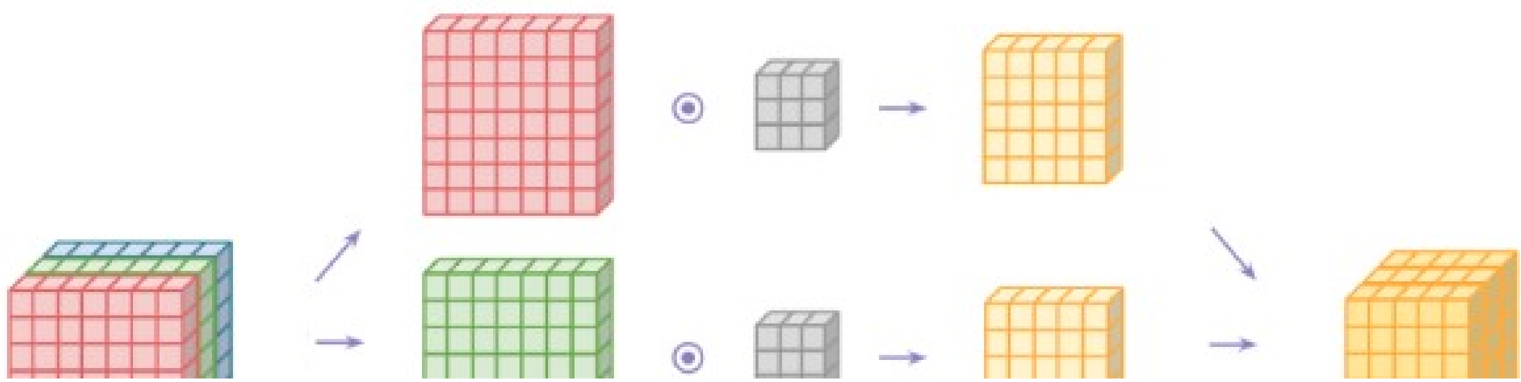
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \times$$

Conv NNs

- Convolutional layer – typology
  - Standard Convolution



  - Depthwise Separable Convolution



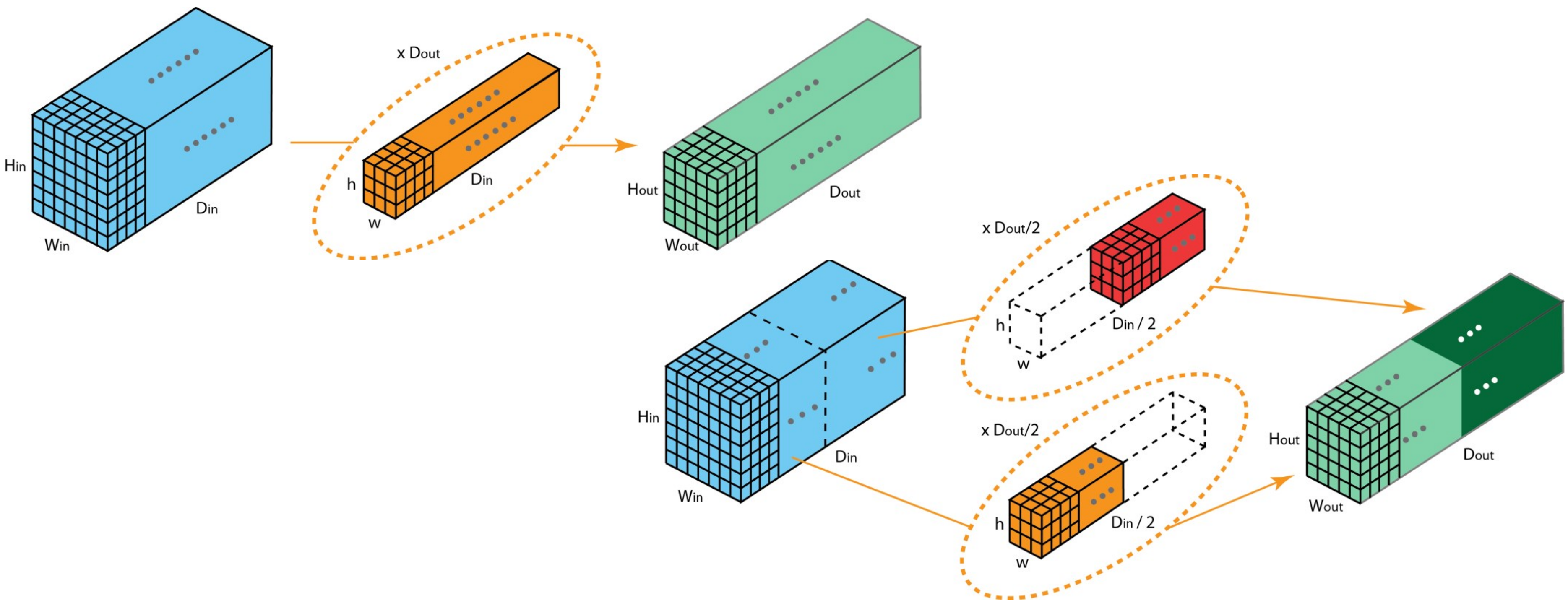
Conv NNs

- Convolutional layer – typology
  - Classic *vs*. spatial separable

classic	Spatial separable
Image: W_I x H_I x D_I = 12 x 12 x 3 1 Filter: F_W x F_H x F_D = 5 x 5 x 3 No Padding: P = 0 Stride 1: S = 1 Output: W_O x H_O x D_O = 8 x 8 x 1 1* (5 * 5 * 3) * (8 * 8) = 4800 => 4 800 ops.	Image: W_I x H_I x D_I = 12 x 12 x 3 3 Filters: F_W x F_H x F_D = 5 x 5 x 1 1 Filter: F'_W x F'_H x F'_D = 1 x 1 x 3 No Padding: P = 0 Stride 1: S = 1 Output: W_O x H_O x D_O = 8 x 8 x 256 3 * (5 * 5) * (8 * 8) = 4800 1* (1 * 1 * 3) * (8 * 8) = 192 => 4 992 ops
Image: W_I x H_I x D_I = 12 x 12 x 3 256 Filters: F_W x F_H x F_D = 5 x 5 x 3 No Padding: P = 0 Stride 1: S = 1 Output: W_O x H_O x D_O = 8 x 8 x 256 256 * (5 * 5 * 3) * (8 * 8) = 1228800 => 1 228 800 ops	Image: W_I x H_I x D_I = 12 x 12 x 3 3 Filters: F_W x F_H x F_D = 5 x 5 x 1 256 Filters: F'_W x F'_H x F'_D = 1 x 1 x 3 No Padding: P = 0, Stride 1: S = 1 Output: W_O x H_O x D_O = 8 x 8 x 256 3 * (5 * 5) * (8 * 8) = 4800 256 * (1 * 1 * 3) * (8 * 8) = 49152 => 53 952 ops

Conv NNs

- Convolutional layer – typology
  - Grouped convolutions
    - See https://arxiv.org/pdf/1605.06489.pdf
    - Efficient training (More GPUs) => model parallelisation
    - Fewer parameters
    - Better representations



Conv NNs

- Convolutional layer – typology
  - Classic *vs*. grouped convolutions

Classic	Group
Image: $W_I \times H_I \times D_I = 12 \times 12 \times D_I$ D_O Filter: $F_W \times F_H \times D_I = 5 \times 5 \times D_I$ No Padding: $P = 0$ Stride 1: $S = 1$ Output: $W_O \times H_O \times D_O = 8 \times 8 \times D_O$ #operations: $F_W \times F_H \times D_I \times D_O = 5 \times 5 \times D_I \times D_O$	Image: $W_I \times H_I \times D_I = 12 \times 12 \times D_I$ $D_O/2$ Filters: $F_W \times F_H \times (D_I / 2) = 5 \times 5 \times (D_I / 2)$ $D_O/2$ Filters: $F_W \times F_H \times (D_I / 2) = 5 \times 5 \times (D_I / 2)$ No Padding: $P = 0$ Stride 1: $S = 1$ Output: $W_O \times H_O \times D_O = 8 \times 8 \times D_O$ #operations: $[F_W \times F_H \times (D_I / 2) \times (D_O / 2)] * 2$ $= 5 \times 5 \times D_I \times D_O / 2$

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 → 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

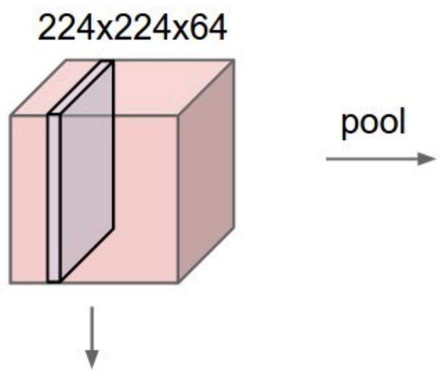
Conv NNs

- 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*11*11*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**

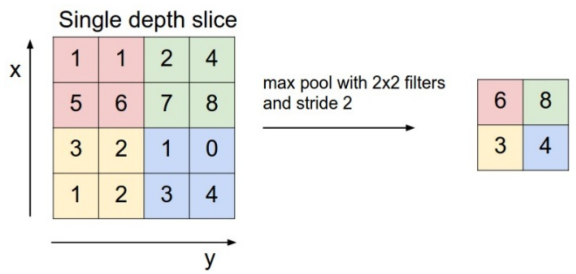
Convolutional Neural Networks

- CNN important strong points:
  - Sparse interactions
  - Parameter sharing
  - Equivariant representations
    - Pro: Translation
    - Cons: rotation, scale
  - allows for working with inputs of variable size

Conv NNs

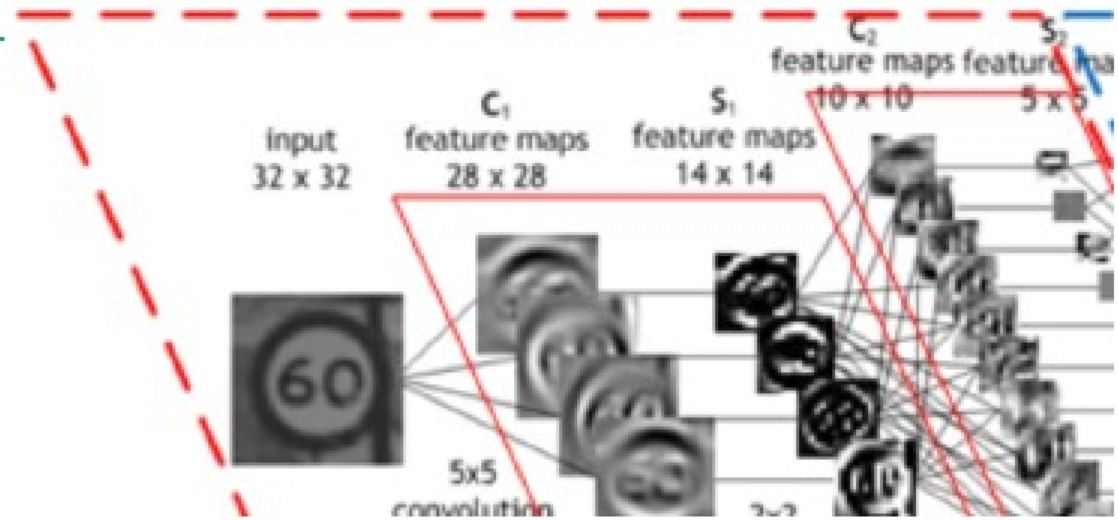


- Pooling layer
  - Aim
    - progressively reduce the spatial size of the representation
      - to reduce the amount of parameters and computation in the network
      - to also control overfitting
    - a subsampling step
      - downsample the spatial dimensions of the input.
    - simplify the information in the output from the convolutional layer
  - How it works
    - takes each feature map output from the convolutional layer and prepares a condensed feature map
    - each unit in the pooling layer may summarize a region in the previous layer
    - apply pooling filters to each feature map separately
      - Pooling filter size (spatial extent of pooling) PF
      - Pooling filter stride PS
      - No padding



Conv NNs

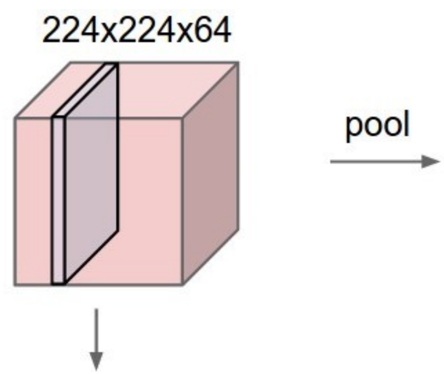
- Pooling layer
  - How it works



- resizes it spatially, using
  - the MAX operation
  - the average operation
  - L^p norm: $\sqrt[p]{\sum x^p}$
    - L^2-norm operation (square root of the sum of the squares of the activations in a rectangular neighbourhood/region) ←→ p =2
  - Log prob PROB: $\frac{1}{b}\log\left(\sum e^{bx}\right)$

Conv NNs

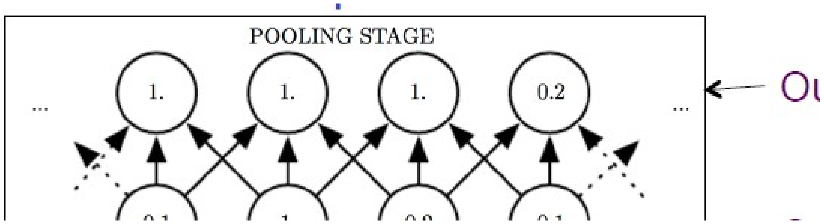
- Pooling layer
  - Two reasons:
    - Dimensionality reduction

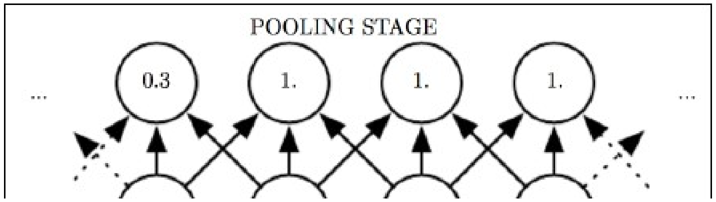


    - Invariance to transformation (rotation, translation)

Conv NNs

- Pooling layer
  - Two reasons:
    - Invariance to transformation (rotation, translation)
      - Small translations – e.g. Max pooling

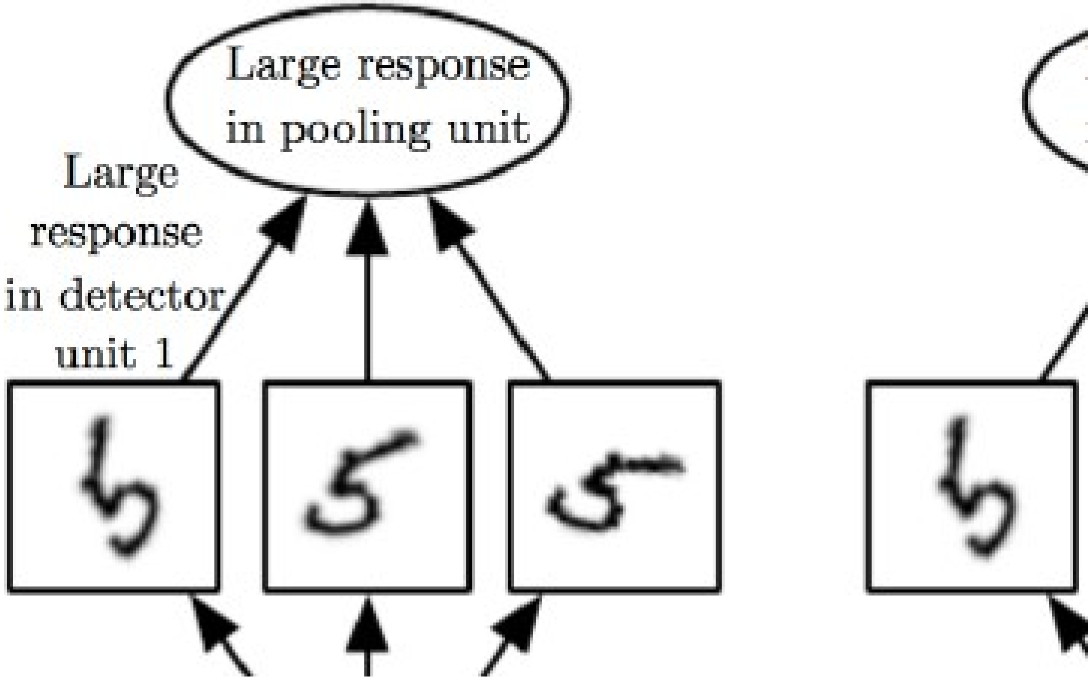




      - When? => if we care about whether a feature is present rather than exactly where it is

Conv NNs

- Pooling layer
  - Two reasons:
    - Invariance to transformation (rotation, translation)
      - Rotations

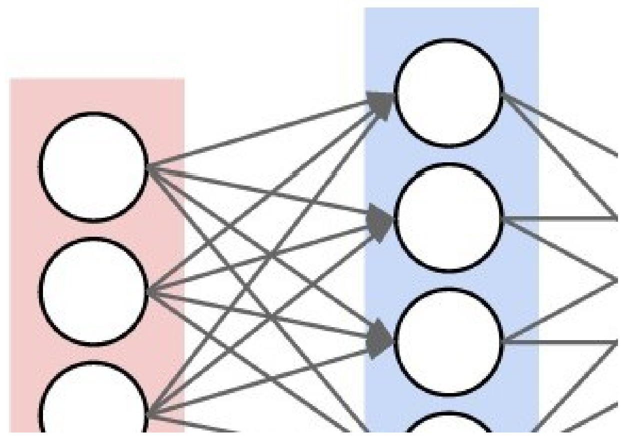


Conv NNs

- Pooling layer
  - Size conversion
    - Input:
      - K x N
    - Output
      - K x [(N– PF)/PS + 1]
  - Typology
    - Local pooling (patch-based pooling)
    - Global pooling (image-based pooling)
  - 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 ←→ 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
    - $S >= 1$
    - $P = (F - 1) / 2$
  - Pool layers
    - $F <= 3$, $S = 2$

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 σ(**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]
        - W1 = [1, 0, 0, 0]
        - W2 = [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
      - λ1|w|+λ2w2λ1|w|+λ2w2
    - 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
      - $1/2n \sum_x \| D - C \|^2$
    - Cross-entropy loss (negative log likelihood)
      - $-1/n \sum_x [D \ln C + (1 - D) \ln(1 - C)]$
  - 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* random(D,H)
  - Calibrating the variances with 1/sqrt(#Inputs)
    - w = random (#Inputs) / sqrt(#Inputs)
  - Sparse initialization
  - Initializing the biases
  - In practice
    - w = 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

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
        - $\alpha = \alpha_0 \exp(-kt)$,

        where α_0, k are hyperparameters and t is the iteration number (but you can also use units of epochs).
      - 1/t decay
        - $\alpha = \alpha_0/(1+kt)$

        where α_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