Computer Vision 3

Ș.l. dr. ing. Mihai DOGARIU
www.mdogariu.aimultimedialab.ro

1

Structura cursului

M1. Introducere
M2. Fundamentele Învățării Adânci (Deep Learning Fundamentals)
M3. Învățare Adâncă Supervizată (Supervised Deep Learning)
M4. Învățare Adâncă Nesupervizată (Unsupervised Deep Learning)
M5. Învățare Consolidată (Reinforcement Learning)

2

M2. Fundamentele Învățării Adânci

2.1. Structura unei rețele neuronale
2.2. Procesul de bază al învățării
2.3. Terminologie
2.4. Considerații practice

3

M2.1. Structura unei rețele neuronale

4

Definiția învățării

Învățare (learning) = procesul de a dobândi noi înțelegeri, cunoștințe, comporatamente, aptitudini, valori, atitudini sau preferințe [1].

Învățarea mașinilor (machine learning) = spunem despre un sistem că „învață” din experiența E cu privire la o clasă de sarcini de lucru T și o măsură de performanță P, dacă performanța sa în rezolvarea sarcinilor T, măsurată prin P, crește cu experiența E [2].

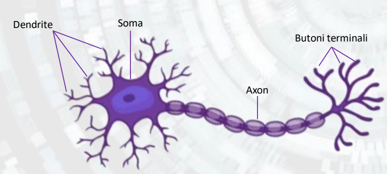
Exemplu:
- T = prognoză meteo (plouă/nu plouă)
- P = procentajul de zile prezise corect
- E = datele meteo ale precedenților 3 ani

5

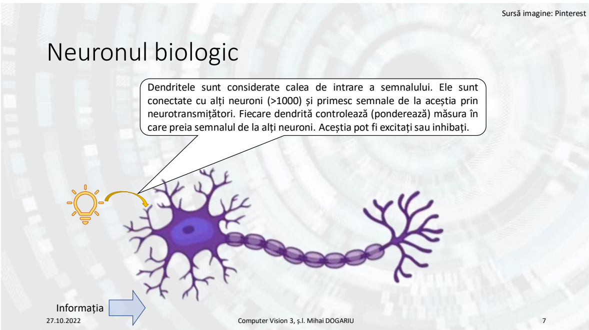
Sursă imagine: Pinterest

Neuronul biologic

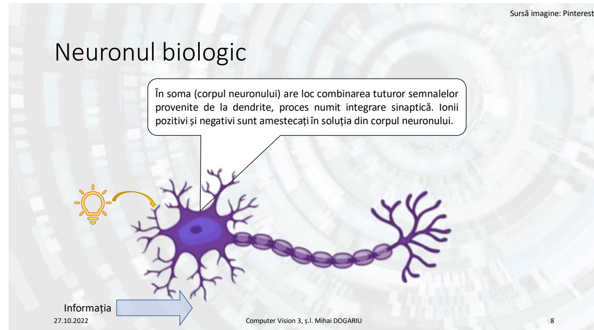
- Celula fundamentală a sistemului nervos



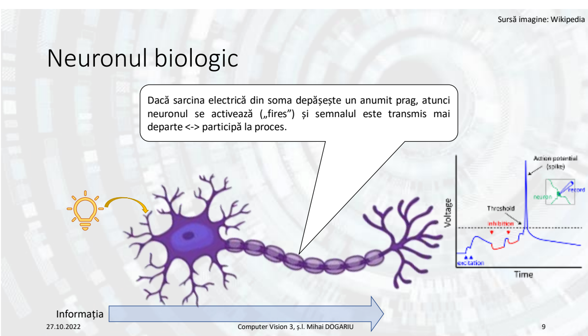
6


Sursă imagine: Pinterest
Neuronul biologic
Dendritele sunt considerate calea de intrare a semnalului. Ele sunt conectate cu alți neuroni (>1000) și primesc semnale de la aceștia prin neurotransmițători. Fiecare dendrită controlează (ponderează) măsura în care preia semnalul de la alți neuroni. Aceștia pot fi excitați sau inhibați.
Informația
27.10.2022
Computer Vision 3, ș.l. Mihai DOGARIU
7

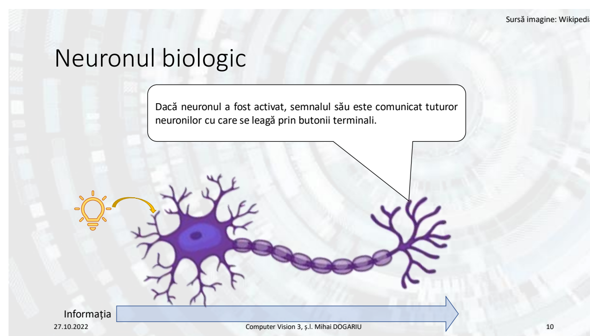
7


Sursă imagine: Pinterest
Neuronul biologic
În soma (corpul neuronului) are loc combinarea tuturor semnalelor provenite de la dendrite, proces numit integrare sinaptică. Ionii pozitivi și negativi sunt amestecați în soluția din corpul neuronului.
Informația
27.10.2022
Computer Vision 3, ș.l. Mihai DOGARIU
8

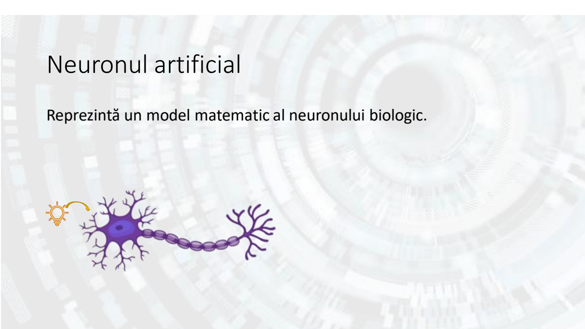
8


Sursă imagine: Wikipedia
Neuronul biologic
Dacă sarcina electrică din soma depășește un anumit prag, atunci neuronul se activează („fires") și semnalul este transmis mai departe <-> participă la proces.
Action potential
(spike)
Threshold
inhibition
excitation
neuron
record
Voltage
Time
Informația
27.10.2022
Computer Vision 3, ș.l. Mihai DOGARIU
9

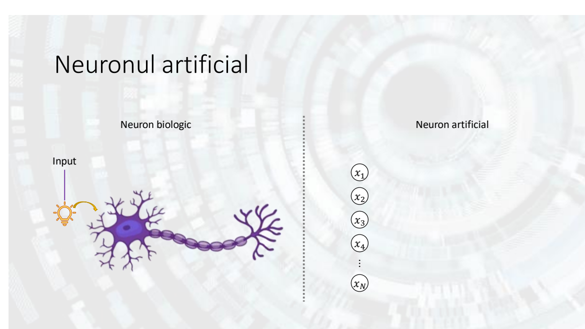
9


Sursă imagine: Wikipedia
Neuronul biologic
Dacă neuronul a fost activat, semnalul său este comunicat tuturor neuronilor cu care se leagă prin butonii terminali.
Informația
27.10.2022
Computer Vision 3, ș.l. Mihai DOGARIU
10

10

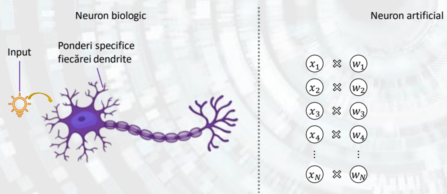

Neuronul artificial
Reprezintă un model matematic al neuronului biologic.
27.10.2022
Computer Vision 3, ș.l. Mihai DOGARIU
11

11


Neuronul artificial
Neuron biologic
Neuron artificial
Input
x_1
x_2
x_3
x_4
⋮
x_N
27.10.2022
Computer Vision 3, ș.l. Mihai DOGARIU
12

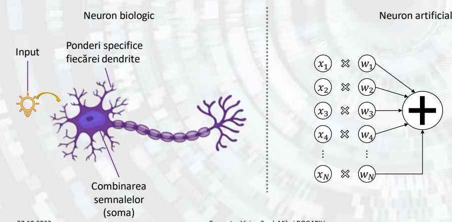
12

Neuronul artificial



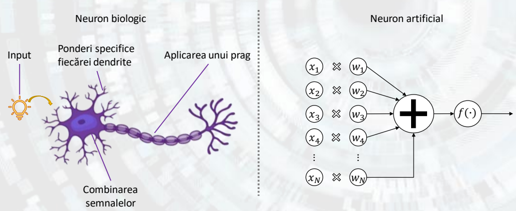
13

Neuronul artificial



14

Neuronul artificial



15

Neuronul artificial

Modelul matematic:

$$y = f\left(\sum_{i=i}^{n} w_i x_i\right)$$

w_i – ponderi
x_i – vector de intrare
n – dimensiunea vectorului de intrare
$f(\cdot)$ – funcție de activare
y – ieșirea neuronului

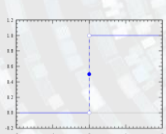
16

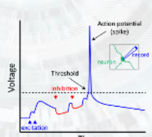
Sursă imagine: Wikipedia

Neuronul artificial – funcția de activare

Rol: implementează pragul de activare al neuronului biologic – ce fel de funcție putem aplica?

Funcția prag (Heaviside): $H(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$





17

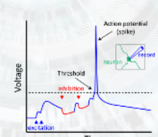
Sursă imagine: Wikipedia

Neuronul artificial – funcția de activare

Fiecare neuron are un prag propriu de activare = polarizarea neuronului. Pentru a păstra funcția de activare Heaviside, includem polarizarea în formularea modelului neuronului artificial.

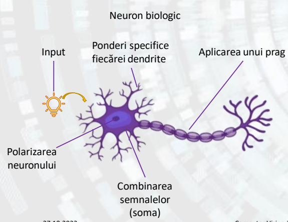
$$f\left(\sum_{i=1}^{n} w_i x_i\right) = \begin{cases} 1, \sum_{i=1}^{n} w_i x_i \geq prag \\ 0, \sum_{i=1}^{n} w_i x_i < prag \end{cases}$$

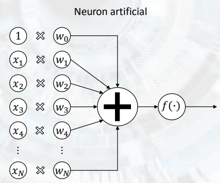
$$H\left(\sum_{i=1}^{n} w_i x_i - prag\right) = \begin{cases} 1, \sum_{i=1}^{n} w_i x_i - prag \geq 0 \\ 0, \sum_{i=1}^{n} w_i x_i - prag < 0 \end{cases}$$



18

Neuronul artificial





19

Neuronul artificial

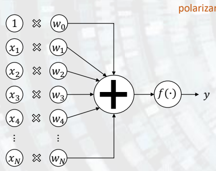
Modelul matematic:

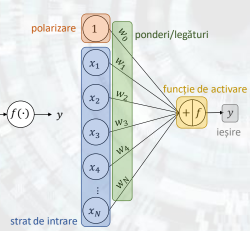
$$y = f\left(\sum_{i=0}^{n} w_i x_i\right) = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

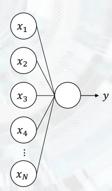
w_i – ponderi; $w_0 \rightarrow b$ - polarizare (bias)
x_i – vector de intrare; $x_0 = 1$
n – dimensiunea vectorului de intrare
$f(\cdot)$ – funcție de activare; $f(\cdot) = H(\cdot) \Rightarrow$ perceptron (clasificator binar)
y – ieșirea neuronului

20

Neuronul artificial – reprezentare grafică



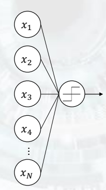




21

Perceptronul

- Caz particular al neuronului artificial
- Folosește funcția de activare Heaviside
- Poate fi folosit pentru a implementa doar modele liniar separabile

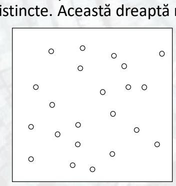


22

Sursă imagine: Pinterest

Spațiu liniar separabil

Un spațiu bidimensional de puncte făcând parte din 2 tipuri (clase) diferite de puncte este liniar separabil dacă există o dreaptă care separă planul în cele 2 clase distincte. Această dreaptă nu este unică.

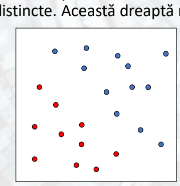


23

Sursă imagine: Pinterest

Spațiu liniar separabil

Un spațiu bidimensional de puncte făcând parte din 2 tipuri (clase) diferite de puncte este liniar separabil dacă există o dreaptă care separă planul în cele 2 clase distincte. Această dreaptă nu este unică.

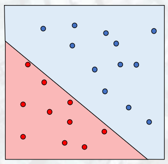


24

Sursă imagine: Pinterest

Spațiu liniar separabil

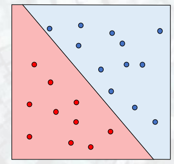
Un spațiu bidimensional de puncte făcând parte din 2 tipuri (clase) diferite de puncte este liniar separabil dacă există o dreaptă care separă planul în cele 2 clase distincte. Această dreaptă nu este unică.



Sursă imagine: Pinterest

Spațiu liniar separabil

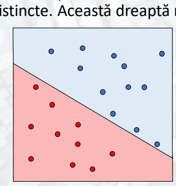
Un spațiu bidimensional de puncte făcând parte din 2 tipuri (clase) diferite de puncte este liniar separabil dacă există o dreaptă care separă planul în cele 2 clase distincte. Această dreaptă nu este unică.



Sursă imagine: Pinterest

Spațiu liniar separabil

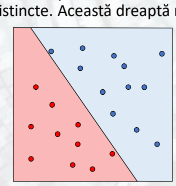
Un spațiu bidimensional de puncte făcând parte din 2 tipuri (clase) diferite de puncte este liniar separabil dacă există o dreaptă care separă planul în cele 2 clase distincte. Această dreaptă nu este unică.



Sursă imagine: Pinterest

Spațiu liniar separabil

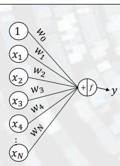
Un spațiu bidimensional de puncte făcând parte din 2 tipuri (clase) diferite de puncte este liniar separabil dacă există o dreaptă care separă planul în cele 2 clase distincte. Această dreaptă nu este unică.



Sursă imagine: veryicon

Învățarea unui perceptron

Învățarea = procesul iterativ prin care sunt găsite valorile tuturor parametrilor perceptronului astfel încât modelul final să clasifice correct în mod binar datele de intrare.



Ce parametri sunt ficși?
- input
- output
- funcția de transfer

Ce parametri pot fi ajustați/învățați?
- ponderile
- polarizarea

Sursă imagine: veryicon

Învățarea unui perceptron

Notații:
- $D = \{(\boldsymbol{x}_1, \widehat{y_1}), (\boldsymbol{x}_2, \widehat{y_2}), \dots, (\boldsymbol{x}_n, \widehat{y_n})\}$ – setul de date de antrenare;
- $\boldsymbol{x}_j$ – al *j*-lea vector de intrare *n*-dimensional;
- $x_{j,i}$ – a *i*-a componentă a celui de-al *j*-lea vector de intrare;
- $\widehat{y}_j$ – a *j*-a ieșire dorită, corespunzând lui $\boldsymbol{x}_j$ <-> groundtruth;
- $y = f(\boldsymbol{x})$ – ieșirea sistemului pentru vectorul de intrare $\boldsymbol{x}$;
- α – rata de învățare, ce va avea o valoare între 0 și 1;
- w_i – ponderea de pe poziția *i*, care se va înmulți cu componenta *i* a vectorului de intrare;
- $x_{j,0} = 1, \forall j$;
- w_0 – polarizarea;
- $w_i(t)$ – valoarea ponderei w_i la momentul de timp *t*.

Sursă imagine: veryicon

Învățarea unui perceptron

<u>Algoritm:</u>

1. inițializarea ponderilor la o valoare mică.
2. pentru fiecare pereche $(\boldsymbol{x}_j, \hat{y}_j)$ din setul de date de antrenare:
   2.1. calculăm ieșirea sistemului:
   $$y_j(\mathrm{t}) = f\left(\sum_{i=0}^{n} w_i x_{j,i}\right)$$
   2.2. actualizăm ponderile, $\forall i, 0 \le i \le n$:
   $$w_i(\mathrm{t}+1) = w_i(t) + \alpha * \left(\hat{y}_j - y_j(\mathrm{t})\right) * x_{j,i}$$
3. repetăm pasul 2 până când eroarea iterației este mai mică decât un prag prestabilit sau până când s-au rulat un anumit număr de iterații.

31

Sursă imagine: veryicon

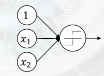
Învățarea unui perceptron

Aplicație

Să se realizeze un perceptron care implementează funcția logică AND.

AND	0	1
0	0	0
1	0	1

	Input		Output
j=1	0	0	0
j=2	0	1	0
j=3	1	0	0
j=4	1	1	1
	$x_{j,1}$	$x_{j,2}$	$\hat{y}_j$



$$D = \{((0,0),0), ((0,1),0), ((1,0),0), ((1,1),1)\}$$
$$\alpha = 1$$

32

Sursă imagine: veryicon

Învățarea unui perceptron

1. Inițializăm toate ponderile (o singură dată):
$$w_0 = w_1 = w_2 = 0$$
2.1. calculăm ieșirea sistemului:
$$\begin{aligned} y_1(0) &= f\left(\sum_{i=0}^{n} w_i x_{1,i}\right) \\ &= f\left(w_0 x_{1,0} + w_1 x_{1,1} + w_2 x_{1,2}\right) \\ &= f(0*1+0*0+0*0) \\ &= f(0) \\ &= 1 \end{aligned}$$

$$f(x) = \begin{cases} 1, x \ge 0 \\ 0, x < 0 \end{cases}$$
$$\begin{array}{ll} t = 0 & j = 1 \\ w_0 = 0; & x_{1,0} = 1 \\ w_1 = 0; & x_{1,1} = 0 \\ w_2 = 0; & x_{1,2} = 0 \\ & \widehat{y_1} = 0 \end{array}$$

33

Sursă imagine: veryicon

Învățarea unui perceptron

2.2. actualizăm ponderile, $\forall i, 0 \le i \le n$:
$$\begin{aligned} w_0(1) &= w_0(0) + \alpha * \left(\widehat{y_1} - y_1(0)\right) * x_{1,0} \\ w_1(1) &= w_1(0) + \alpha * \left(\widehat{y_1} - y_1(0)\right) * x_{1,1} \\ w_2(1) &= w_2(0) + \alpha * \left(\widehat{y_1} - y_1(0)\right) * x_{1,2} \end{aligned}$$

$$\begin{aligned} w_0(1) &= 0 + 1*(0-1)*1 = -1 \\ w_1(1) &= 0 + 1*(0-1)*0 = 0 \\ w_2(1) &= 0 + 1*(0-1)*0 = 0 \end{aligned}$$

$$f(x) = \begin{cases} 1, x \ge 0 \\ 0, x < 0 \end{cases}$$
$$\begin{array}{ll} t = 0 & j = 1 \\ w_0 = 0; & x_{1,0} = 1 \\ w_1 = 0; & x_{1,1} = 0 \\ w_2 = 0; & x_{1,2} = 0 \\ & \widehat{y_1} = 0 \end{array}$$

34

Sursă imagine: veryicon

Învățarea unui perceptron

2.1. calculăm ieșirea sistemului:
$$\begin{aligned} y_2(1) &= f\left(\sum_{i=0}^{n} w_i x_{2,i}\right) \\ &= f\left(w_0 x_{2,0} + w_1 x_{2,1} + w_2 x_{2,2}\right) \\ &= f(-1*1+0*0+0*1) \\ &= f(-1) \\ &= 0 \end{aligned}$$

$$f(x) = \begin{cases} 1, x \ge 0 \\ 0, x < 0 \end{cases}$$
$$\begin{array}{ll} t = 1 & j = 2 \\ w_0 = -1; & x_{2,0} = 1 \\ w_1 = 0; & x_{2,1} = 0 \\ w_2 = 0; & x_{2,2} = 1 \\ & \widehat{y_2} = 0 \end{array}$$

35

Sursă imagine: veryicon

Învățarea unui perceptron

2.2. actualizăm ponderile, $\forall i, 0 \le i \le n$:
$$\begin{aligned} w_0(2) &= w_0(1) + \alpha * \left(\widehat{y_2} - y_2(1)\right) * x_{2,0} \\ w_1(2) &= w_1(1) + \alpha * \left(\widehat{y_2} - y_2(1)\right) * x_{2,1} \\ w_2(2) &= w_2(1) + \alpha * \left(\widehat{y_2} - y_2(1)\right) * x_{2,2} \end{aligned}$$

$$\begin{aligned} w_0(2) &= -1 + 1*(0-0)*1 = -1 \\ w_1(2) &= 0 + 1*(0-0)*0 = 0 \\ w_2(2) &= 0 + 1*(0-0)*1 = 0 \end{aligned}$$

$$f(x) = \begin{cases} 1, x \ge 0 \\ 0, x < 0 \end{cases}$$
$$\begin{array}{ll} t = 1 & j = 2 \\ w_0 = -1; & x_{2,0} = 1 \\ w_1 = 0; & x_{2,1} = 0 \\ w_2 = 0; & x_{2,2} = 1 \\ & \widehat{y_2} = 0 \end{array}$$

36

Sursă imagine: veryicon

Învățarea unui perceptron

2.1. calculăm ieșirea sistemului:

$$y_3(2) = f\left(\sum_{i=0}^{n} w_i x_{3,i}\right)$$
$$= f\left(w_0 x_{3,0} + w_1 x_{3,1} + w_2 x_{3,2}\right)$$
$$= f(-1 * 1 + 0 * 1 + 0 * 0)$$
$$= f(-1)$$
$$= 0$$

$f(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$

$t = 2$ $j = 3$
$w_0 = -1;$ $x_{3,0} = 1$
$w_1 = 0;$ $x_{3,1} = 1$
$w_2 = 0;$ $x_{3,2} = 0$
$\widehat{y_3} = 0$

37

Sursă imagine: veryicon

Învățarea unui perceptron

2.2. actualizăm ponderile, $\forall i, 0 \leq i \leq n$:

$$w_0(3) = w_0(2) + \alpha * \left(\widehat{y_3} - y_3(2)\right) * x_{3,0}$$
$$w_1(3) = w_1(2) + \alpha * \left(\widehat{y_3} - y_3(2)\right) * x_{3,1}$$
$$w_2(3) = w_2(2) + \alpha * \left(\widehat{y_3} - y_2(2)\right) * x_{3,2}$$

$$w_0(3) = -1 + 1 * (0 - 0) * 1 = -1$$
$$w_1(3) = 0 + 1 * (0 - 0) * 1 = 0$$
$$w_2(3) = 0 + 1 * (0 - 0) * 0 = 0$$

$f(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$

$t = 2$ $j = 3$
$w_0 = -1;$ $x_{3,0} = 1$
$w_1 = 0;$ $x_{3,1} = 1$
$w_2 = 0;$ $x_{3,2} = 0$
$\widehat{y_3} = 0$

38

Sursă imagine: veryicon

Învățarea unui perceptron

2.1. calculăm ieșirea sistemului:

$$y_4(3) = f\left(\sum_{i=0}^{n} w_i x_{4,i}\right)$$
$$= f\left(w_0 x_{4,0} + w_1 x_{4,1} + w_2 x_{4,2}\right)$$
$$= f(-1 * 1 + 0 * 1 + 0 * 1)$$
$$= f(-1)$$
$$= 0$$

$f(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$

$t = 3$ $j = 4$
$w_0 = -1;$ $x_{4,0} = 1$
$w_1 = 0;$ $x_{4,1} = 1$
$w_2 = 0;$ $x_{4,2} = 1$
$\widehat{y_4} = 1$

39

Sursă imagine: veryicon

Învățarea unui perceptron

2.2. actualizăm ponderile, $\forall i, 0 \leq i \leq n$:

$$w_0(4) = w_0(3) + \alpha * \left(\widehat{y_4} - y_4(3)\right) * x_{4,0}$$
$$w_1(4) = w_1(3) + \alpha * \left(\widehat{y_4} - y_4(3)\right) * x_{4,1}$$
$$w_2(4) = w_2(3) + \alpha * \left(\widehat{y_4} - y_4(3)\right) * x_{4,2}$$

$$w_0(4) = -1 + 1 * (1 - 0) * 1 = 0$$
$$w_1(4) = 0 + 1 * (1 - 0) * 1 = 1$$
$$w_2(4) = 0 + 1 * (1 - 0) * 1 = 1$$

$f(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$

$t = 3$ $j = 4$
$w_0 = -1;$ $x_{4,0} = 1$
$w_1 = 0;$ $x_{4,1} = 1$
$w_2 = 0;$ $x_{4,2} = 1$
$\widehat{y_4} = 1$

40

Sursă imagine: veryicon

Învățarea unui perceptron

2.1. calculăm ieșirea sistemului:

$$y_1(4) = f\left(\sum_{i=0}^{n} w_i x_{1,i}\right)$$
$$= f\left(w_0 x_{1,0} + w_1 x_{1,1} + w_2 x_{1,2}\right)$$
$$= f(0 * 1 + 1 * 0 + 1 * 0)$$
$$= f(0)$$
$$= 1$$

$f(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$

$t = 4$ $j = 1$
$w_0 = 0;$ $x_{1,0} = 1$
$w_1 = 1;$ $x_{1,1} = 0$
$w_2 = 1;$ $x_{1,2} = 0$
$\widehat{y_1} = 0$

S-a efectuat o trecere completă prin baza de date

41

Sursă imagine: veryicon

Învățarea unui perceptron

2.2. actualizăm ponderile, $\forall i, 0 \leq i \leq n$:

$$w_0(5) = w_0(4) + \alpha * \left(\widehat{y_1} - y_1(4)\right) * x_{1,0}$$
$$w_1(5) = w_1(4) + \alpha * \left(\widehat{y_1} - y_1(4)\right) * x_{1,1}$$
$$w_2(5) = w_2(4) + \alpha * \left(\widehat{y_1} - y_1(4)\right) * x_{1,2}$$

$$w_0(5) = 0 + 1 * (0 - 1) * 1 = -1$$
$$w_1(5) = 1 + 1 * (0 - 1) * 0 = 1$$
$$w_2(5) = 1 + 1 * (0 - 1) * 0 = 1$$

$f(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$

$t = 4$ $j = 1$
$w_0 = 0;$ $x_{1,0} = 1$
$w_1 = 1;$ $x_{1,1} = 0$
$w_2 = 1;$ $x_{1,2} = 0$
$\widehat{y_1} = 0$

42

Sursă imagine: veryicon

Învățarea unui perceptron

Iterația #1

	$x_{j,0}$	$x_{j,1}$	$x_{j,2}$	$\hat{y}_j$	y_j	Δy	w_0	w_1	w_2
j=1	1	0	0	0	1	-1	-1	0	0
j=2	1	0	1	0	0	0	-1	0	0
j=3	1	1	0	0	0	0	-1	0	0
j=4	1	1	1	1	0	1	0	1	1

Iterația #2

	$x_{j,0}$	$x_{j,1}$	$x_{j,2}$	$\hat{y}_j$	y_j	Δy	w_0	w_1	w_2
j=1	1	0	0	0	1	-1	-1	1	1
j=2	1	0	1	0	1	-1	-2	1	0
j=3	1	1	0	0	0	0	-2	1	0
j=4	1	1	1	1	0	1	-1	2	1

43

Sursă imagine: veryicon

Învățarea unui perceptron

Iterația #3

	$x_{j,0}$	$x_{j,1}$	$x_{j,2}$	$\hat{y}_j$	y_j	Δy	w_0	w_1	w_2
j=1	1	0	0	0	0	0	-1	2	1
j=2	1	0	1	0	1	-1	-2	2	0
j=3	1	1	0	0	1	-1	-3	1	0
j=4	1	1	1	1	0	1	-2	2	1

Iterația #4

	$x_{j,0}$	$x_{j,1}$	$x_{j,2}$	$\hat{y}_j$	y_j	Δy	w_0	w_1	w_2
j=1	1	0	0	0	0	0	-2	2	1
j=2	1	0	1	0	0	0	-2	2	1
j=3	1	1	0	0	1	-1	-3	1	1
j=4	1	1	1	1	0	1	-2	2	2

44

Sursă imagine: veryicon

Învățarea unui perceptron

Iterația #5

	$x_{j,0}$	$x_{j,1}$	$x_{j,2}$	$\hat{y}_j$	y_j	Δy	w_0	w_1	w_2
j=1	1	0	0	0	0	0	-2	2	2
j=2	1	0	1	0	1	-1	-3	2	1
j=3	1	1	0	0	0	0	-3	2	1
j=4	1	1	1	1	1	0	-3	2	1

Iterația #6

	$x_{j,0}$	$x_{j,1}$	$x_{j,2}$	$\hat{y}_j$	y_j	Δy	w_0	w_1	w_2
j=1	1	0	0	0	0	0	-3	2	1
j=2	1	0	1	0	0	0	-3	2	1
j=3	1	1	0	0	0	0	-3	2	1
j=4	1	1	1	1	1	0	-3	2	1

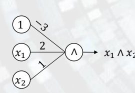
45

Sursă imagine: veryicon

Învățarea unui perceptron

Iterația #6

	$x_{j,0}$	$x_{j,1}$	$x_{j,2}$	$\hat{y}_j$	y_j	Δy	w_0	w_1	w_2
j=1	1	0	0	0	0	0	-3	2	1
j=2	1	0	1	0	0	0	-3	2	1
j=3	1	1	0	0	0	0	-3	2	1
j=4	1	1	1	1	1	0	-3	2	1



Dacă nu apare nicio eroare pe parcursul unei iterații => s-a ajuns la convergența algoritmului

46

Concluzii perceptron

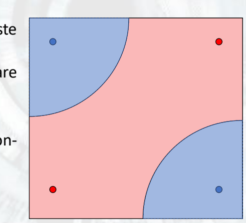
- Clasificator liniar binar ⇔ împarte planul datelor cu un hiperplan conform ecuației unei drepte de ecuație $\boldsymbol{w} \cdot \boldsymbol{x} = b$;
- Funcție de activare neliniară;
- Pentru a obține moduri de separare mai complexe se pot folosi alte tipuri de ecuații neliniare.

47

Spațiu liniar separabil

Cum procedăm dacă spațiul nu este liniar separabil (e.g. funcția XOR)?

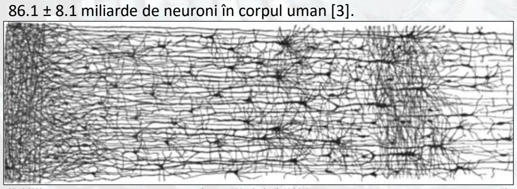
1. Combinăm mai multe funcții liniare în cascadă?
2. Aplicăm non-liniarități?
3. Combinăm mai multe funcții non-liniare în cascadă?



48

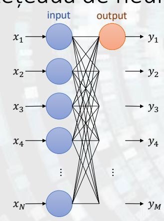
Rețeaua de neuroni unistrat

Sistemul nervos nu este format dintr-un singur neuron, ci dintr-o rețea. Numărul total de neuroni nu este cunoscut exact. Se estimează că ar fi 86.1 ± 8.1 miliarde de neuroni în corpul uman [3].



49

Rețeaua de neuroni unistrat



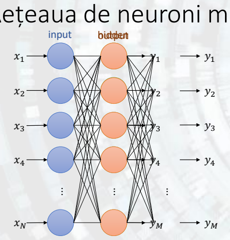
Un singur perceptron poate clasifica informația în doar 2 clase. Mai mult de atât, clasificarea se face „hard” (nu există valori intermediare)

O soluție parțială: Rețea formată din mai mulți perceptroni pe același nivel
+ Se adaptează mai multor clase;
- În continuare, nu se pot modela spații neliniar separabile;
- Toate ieșirile sunt binare => răspunde unei game restrânse de nevoi

N nu trebuie să fie neapărat egal cu M
(în practică, cele două sunt egale foarte rar)

50

Rețeaua de neuroni multistrat



+ O rețea multistrat de perceptroni poate clasifica modele mai complexe de date, e.g. funcția XOR.
- Toate ieșirile sunt binare => răspunde unei game restrânse de nevoi

N nu trebuie să fie neapărat egal cu M
(în practică, cele două sunt egale foarte rar)

51

M2.2. Procesul de bază al învățării

52

Procesul de învățare

Învățarea mașinilor (machine learning) – slide #5 = spunem despre un sistem că „învață” din experiența E cu privire la o clasă de sarcini de lucru T și o măsură de performanță P, dacă performanța sa în rezolvarea sarcinilor T, măsurată prin P, crește cu experiența E [2].

Scopul: găsirea unei funcții care asociază un set de date de intrare cu ieșirea lor corectă, în cel mai bun mod posibil.

Constă în 2 etape:

1. Propagarea înainte (forward propagation);
2. Propagarae înapoi (backpropagation).

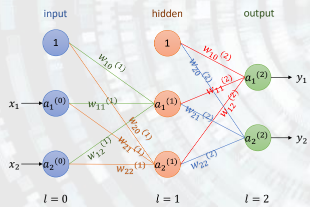
53

Propagarea înainte

- **Scopul**: obținerea ieșirii $y = M(x)$;
- M = funcția de transfer a rețelei neuronale, obținută prin compunerea tuturor funcțiilor de transfer asociate straturilor rețelei.
- Presupune parcurgerea rețelei de neuroni de la intrare către ieșire.
- Se reduce la a calcula activările pentru fiecare neuron din rețea, de la straturile inferioare (mai apropiate de input) către cele superioare (mai apropiate de output).

54

Propagarea înainte



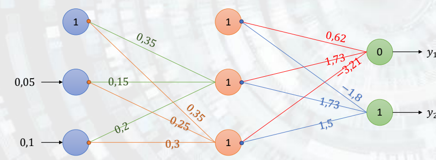
$$a_i^{(l)} = f\left(\sum_{j=0}^{n} w_{i,j}^{(l)} a_i^{(l-1)}\right)$$

$$a_0^{(l)} = 1$$

$$f(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

55

Propagarea înainte – exemplu numeric



56

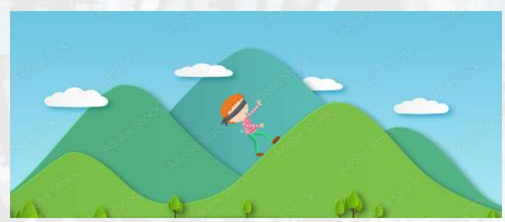
Backpropagation

- **Scopul**: învățarea reprezentării interne a unei rețele neuronale. Presupune aflarea tuturor ponderilor [și polarizărilor] care aduc ieșirea modelului la o formă cât mai apropiată, ideal identică, cu ieșirea așteptată/reală.
- Cum măsurăm ce înseamnă „cât mai apropiată”?
  - R: folosind o funcție de cost, e.g. „eroarea pătratică medie” (mean square error).
- Ce regulă folosim pentru a modifica ponderile?
  - R: scăderea gradientului (gradient descent).

57

Sursă imagine: AdobeStock

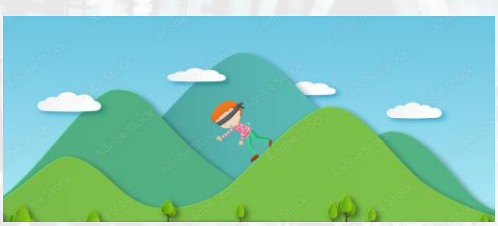
Gradient descent – intuiție



58

Sursă imagine: AdobeStock

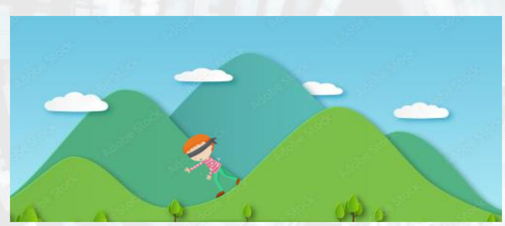
Gradient descent – intuiție



59

Sursă imagine: AdobeStock

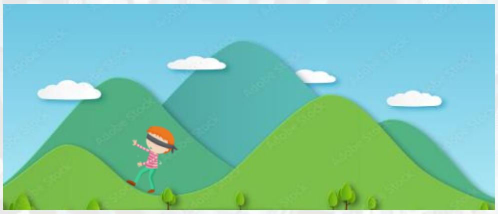
Gradient descent – intuiție



60

Sursă imagine: AdobeStock

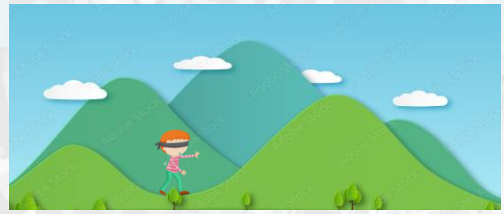
Gradient descent – intuiție



61

Sursă imagine: AdobeStock

Gradient descent – intuiție



62

Gradient descent

➢Metodă de a calcula gradientul parametrilor unei rețele neuronale.

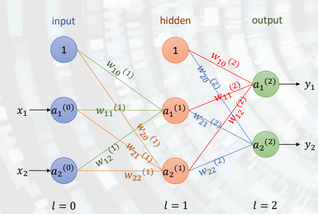
$$y = f(\boldsymbol{w} \cdot \boldsymbol{x} + b) = f(\boldsymbol{w}^T \boldsymbol{x} + b)$$

Funcție de pierdere (se calculează pe o singură pereche de eșantioane) $\mathcal{L}(y, \hat{y}) \overset{e.g.}{\Longrightarrow} \mathcal{L}(y, \hat{y}) = (y - \hat{y})^2$

Funcție de cost (se calculează pe un set de perechi de eșantioane) $J(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(y_i, \hat{y}_i)$

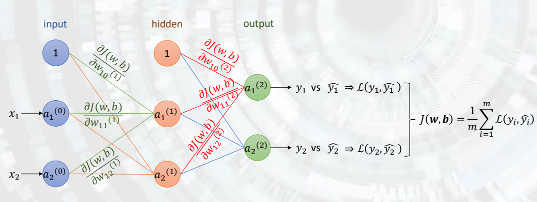
63

Gradient descent



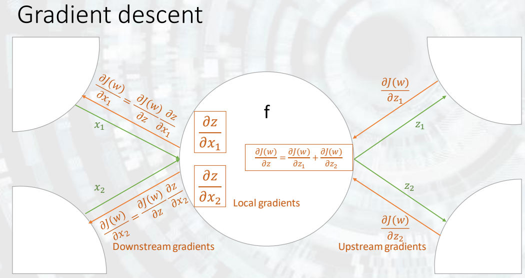
64

Gradient descent



65

Gradient descent



66

Gradient descent

➢Actualizarea ponderilor se face după regula:

$$w^{(t+1)} = w^{(t)} - \alpha \frac{\partial J(w)}{\partial w^{(t)}}$$

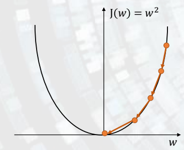
Scriere simplificată: $w \coloneqq w - \alpha \frac{\partial J(w)}{\partial w}$

➢Pentru ponderi din interiorul rețelei se aplică regula înlănțuirii derivatelor parțiale:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Gradient descent

➢Exemplu grafic: considerăm cazul simplificat în care funcția de cost depinde de o singură variabilă și alegem funcția pătratică.

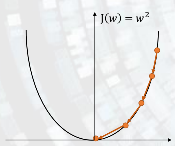


$$w \coloneqq w - \alpha \frac{\partial J(w)}{\partial w}$$

În cazul optim găsim w a.î. $J(w) = 0$, însă, în practică, acest lucru nu se întâmplă aproape niciodată

Gradient descent

➢Exemplu grafic: considerăm cazul simplificat în care funcția de cost depinde de o singură variabilă și alegem funcția pătratică.

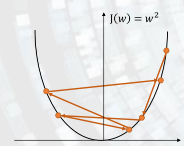


$$w \coloneqq w - \alpha \frac{\partial J(w)}{\partial w}$$

- $\frac{\partial J(w)}{\partial w}$ ne indică direcția în care trebuie făcută modificarea
- α ne indică amplitudinea modificării

Gradient descent – impactul ratei de învățare

➢Același exemplu ca mai devreme, însă avem rată de învățare **mare**.

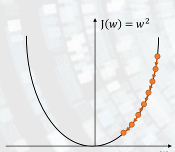


$$w \coloneqq w - \alpha \frac{\partial J(w)}{\partial w}$$

- Algoritmul poate rata punctul de optim, deoarece fiecare iterație aduce un deplasament mare.
+ Datorită „pașilor” mari, deplasarea se face mai rapid.

Gradient descent – impactul ratei de învățare

➢Același exemplu ca mai devreme, însă avem rată de învățare **mică**.

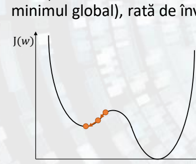


$$w \coloneqq w - \alpha \frac{\partial J(w)}{\partial w}$$

- Algoritmul ajustează ponderile foarte lent.
+ Datorită „pașilor” mici, sunt șanse mai mici să se rateze punctul optim.
De ce nu folosim mereu rată de învățare mică?

Gradient descent – impactul ratei de învățare

➢Funcție de cost neconvexă (un minim local nu este neapărat și minimul global), rată de învățare **mică**.



$$w \coloneqq w - \alpha \frac{\partial J(w)}{\partial w}$$

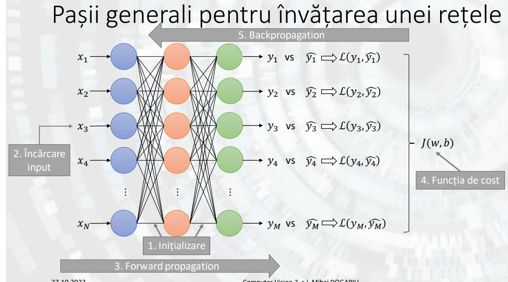
- Există posibilitatea ca algoritmul să se blocheze într-un minim local.

Pașii generali pentru învățarea unei rețele

1. Inițializarea parametrilor rețelei
2. Încărcarea datelor de intrare
3. Forward propagation
4. Calcularea funcției de cost prin compararea datelor de ieșire cu ieșirea reală (groundtruth)
5. Backpropagation + actualizarea parametrilor
6. Repetarea pașilor 3 – 5 până la convergență/optimizare.

73

Pașii generali pentru învățarea unei rețele



74

M2.3. Terminologie

75

Terminologie

1. Funcții de activare
2. Funcții de cost
3. Rata de învățare (learning rate)
4. Parametri vs hiper-parametri
5. Baze de date
6. Optimizatori
7. Regularizare
8. Straturi populare

76

1. Funcții de activare

Funcție de activare = operator diferențiabil care transformă semnalul de intrare în cel de ieșire.

➢Funcția de activare decide dacă un neuron va fi activat sau nu.

➢Poate fi liniară sau neliniară.

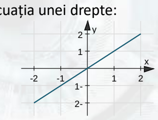
➢Se alege în funcție de aplicația vizată.

77

Funcții de activare liniare

Sunt echivalente cu funcțiile de gradul 1. Reprezintă ecuația unei drepte:

$$f(x) = ax + b$$



➢$\frac{\partial f}{\partial x} = a \leftarrow constantă;$

\+ poate produce rezultate de orice valoare, codomeniul ei putând fi chiar $\mathbb{R}$;

\+ se folosesc pentru a rezolva probleme simple, precum regresia liniară;

\- nu se pretează pentru sarcini mai dificile;

\- compunerea mai multor funcții linare este tot o funcție liniară.

78

Funcții de activare liniare

Prezența funcțiilor liniare în toate straturile unei rețele de neuroni anulează prezența mai multor straturi. O astfel de rețea poate fi echivalată cu un singur strat.

$$y = \boldsymbol{wx} + b$$
$$f(x) = a\boldsymbol{x} + ct$$

$$\begin{aligned} f(y) &= a(\boldsymbol{wx} + b) + ct \\ &= a\boldsymbol{wx} + ab + ct \\ &= (a\boldsymbol{w})\boldsymbol{x} + (ab + ct) \\ &= \boldsymbol{a}'\boldsymbol{x} + ct' \end{aligned}$$

Funcții de activare neliniare

Corpul uman răspunde neliniar la stimuli. Exemplu – servirea felului de mâncare preferat.

Scenariu:

1. felul de mâncare preferat, în compania persoanei preferate, în mediul preferat => neuronii sunt intens activați, senzație puternică.
2. Cazul (1) + ați servit cu o oră mai devreme o pizza mare, deci sunteți sătuli => neuronii sunt mai puțin activați, senzație mai slabă.
3. Cazul (2) + dar nu ați mai mâncat felul preferat de mai mult de o lună => neuronii sunt mai puternic activați decât la (2), dar mai slab decât în cazul (1)

Preferăm utilizarea funcțiilor neliniare pentru a modela sarcini mai complexe!

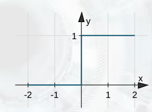
Funcții de activare neliniare

Sunt funcții cu comportament neliniar, capabile să modeleze spații de date mai complexe. Cu ajutorul lor, se pune în valoarea capacitatea de a generaliza a rețelelor neuronale. Cele mai utilizate:

1. treaptă (perceptron);
2. logistică (sigmoid);
3. softplus.
4. tangentă hiperbolică (tanh);
5. ReLU și variantele sale.

Funcția treaptă

$$f(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$



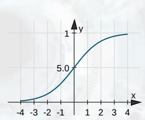
➢ $\frac{\partial f}{\partial x} = \begin{cases} 0, & x \neq 0 \\ nedefinit, & x = 0 \end{cases}$

\+ cea mai simplă cale de a implementa pragul de activare al neuronului;

\- derivata sa nu participă la procesul de antreanre (vezi slide 67).

Funcția logistică (sigmoid)

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$



➢ $\frac{\partial f}{\partial x} = f(x)\big(1 - f(x)\big)$;

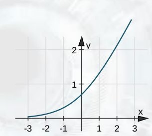
➢Se mai numește și funcție de strivire, deoarece comprimă valorile lui x din $(-\infty, +\infty)$ în $(0,1)$.

\+ Este una dintre cele mai populare funcții de activare, folosită mai ales atunci când ne dorim ca ieșirea să reprezinte o probabilitate.

\- argumentele foarte mari sau foarte mici sunt reduse la aceeași valoare;

Funcția softplus

$$f(x) = \sigma(x) = \ln(1 + e^{x})$$



➢ $\frac{\partial f}{\partial x} = \frac{1}{1+e^{-x}}$;

➢Derivata softplus este chiar funcția logistică.

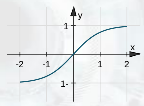
\+ codomeniul este format numai din valori pozitive ⇔ sparsity.

\+ utilizată pentru modelarea piețelor financiare.

\- complexă din punct de vedere computațional.

Funcția tangentă hiperbolică

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



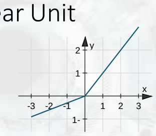
- $\frac{\partial f}{\partial x} = 1 - f(x)^2$;
- Comprimă valorile asemănător funcției logistice, însă în intervalul $(-1,1)$.

\+ spre deosebire de sigmoid, poate produce și valori negative.

\+ de obicei, converge mai repede decât sigmoid.

Funcția Parametric Rectified Linear Unit (PReLU)

$$f(x) = \begin{cases} x, x \geq 0 \\ \alpha x, x < 0 \end{cases}$$



- $\frac{\partial f}{\partial x} = \begin{cases} 1, & x > 0 \\ \alpha, & x < 0 \\ nedefinit, & x = 0 \end{cases}$; $\alpha = \begin{cases} 0 & \Rightarrow & ReLU \\ 0.01 & \Rightarrow & Leaky\ ReLU \\ else & \Rightarrow & PReLU \end{cases}$;
- Cea mai populară funcție de activare.

\+ complexitate de calcul redusă;

Funcția softmax

$$f(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{J} e^{x_i}}$$

- Nu se aplică pe o singură valoare, ci pe un vector de valori.
- Este utilizată pentru probleme de clasificări multi-clasă.
- Este folosită în ultimul strat al rețelei.
- Transformă ieșirea într-o distribuție de probabilități.

Funcții de activare - probleme

1. Gradient dispărând (vanishing gradient)
   - Derivata funcției de activare ia valori mici, propagarea înapoi în rețea devine din ce în ce mai mică, straturile de început ale rețelei ajung să nu se mai modifice pe parcursul antrenării.
   - Funcții afectate: treaptă, sigmoid, tanh (funcții cu codomeniu limitat la ambele capete).
   - Soluții: batch normalization, gradient clipping, greedy layer-wise pre-training, rețele reziduale, inițializare mai bună, alte funcții (e.g. ReLU)

Funcții de activare - probleme

2. Gradient explodând (exploding gradient)
   - Acumularea unor gradienți de valori mari duce la modificări foarte mari ale parametrilor între iterații succesive => algoritmul de antrenare nu mai converge niciodată.
   - Funcții afectate: toate. Nu depinde de funcții, ci de valorile inițiale ale ponderilor.
   - Soluții: gradient clipping, inițializări mai bune, batch normalization.

Funcții de activare - probleme

3. Dying ReLU
   - ReLU transformă orice valoare negativă în 0 => derivata pentru valorile negative va fi mereu 0 => există posibilitatea ca o anumită cale a rețelei să fie mereu anulată (moartă).
   - Funcții afectate: ReLU.
   - Soluții: scăderea ratei de învățare, inițializare mai potrivită, Leaky ReLU

Funcții de activare – ce/când folosim?

- ➢Folosim sigmoid pentru **regresie logistică** și **clasificare binară**;
- ➢În general, tanh este mai bună decât sigmoid fiind centrată în 0;
- ➢Sigmoid și tanh ar trebui evitate datorită **gradientului dispărând**;
- ➢Folosim softmax pentru **clasificări multi-clasă**;
- ➢Rețelele **recurente** folosesc tanh, sigmoid;
- ➢Rețelele convoluționale și complet conectate folosesc ReLU în straturile **ascunse**;
- ➢Dacă anumiți neuroni nu sunt activați niciodată, înlocuim ReLU cu Leaky ReLU. Dacă nici atunci nu sunt activați, folosim Parametric ReLU.

91

Funcții de activare – teorema aproximării universale

Orice funcție continuă $f: [0,1]^n \rightarrow [0,1]$ poate fi aproximată de o rețea neuronală cu un număr finit de neuroni per strat și cu un număr finit de straturi.

92

2. Funcții de cost

Tip de

Funcție de pierdere = o funcție calculată **într-un singur punct** care măsoară penalizarea suferită în urma luării unei decizii (distanța între un punct prezis și valoarea reală).

Parte din

Funcție de cost = funcție de pierdere calculată la nivelul unui **întreg set de date**, în loc de un singur punct. Mai generală decât funcția de pierdere.

Funcție obiectiv = funcție care trebuie optimizată, fie prin minimizare (caz în care este sinonimă cu funcția de cost), fie prin maximizare.

93

Funcții de cost – exemple

Funcții de pierdere:

$$\mathcal{L}(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$
$$\mathcal{L}(y_i) = \max(0, 1 - t_i y_i), t \in \{-1, 1\}, y = \boldsymbol{wx} + b$$

Funcții de cost:

$$MSE = \left[\frac{1}{N}\right] L2 = J(y_i, \hat{y}_i) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$
$$L1 = J(y_i, \hat{y}_i) = \left[\frac{1}{N}\right]\sum_{i=1}^{N}|y_i - \hat{y}_i|$$
$$CE = -\sum_{c=1}^{M} \mathbb{1}_{[y \equiv \hat{y}]} \log(p(y \equiv \hat{y}))$$

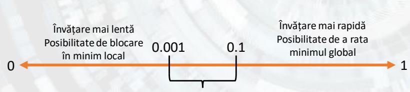
Funcții obiectiv:

$$w = \arg\max_{w} \sum_{i=1}^{M} \log p_{model}(\hat{y}_i | x_i, w)$$

94

3. Rata de învățare

Rata de învățare = parametru ajustabil care determină mărimea pasului făcut în scopul minimizării funcției de cost la fiecare iterație. Reprezintă „viteza” cu care un algoritm învață.



95

4. Parametri vs hiper-parametri

Parametri = variabile configurabile care reprezintă starea internă a modelului unei rețele neuronale. Ei sunt deduși din datele de intrare prin învățare. Exemplu: ponderile.

Hiper-parametri = variabile configurabile, externe modelului unei rețele neuronale și care influențează procesul de învățare. Ei sunt setați de către utilizator. Exemplu: rata de învățare, numărul de epoci, dimensiunea unui batch etc.

96

Parametri vs hiper-parametri – inițializatori

- Gradienți prea mici (datorită valorilor extreme, în special mici, ale ponderilor) => vanishing gradient
- Parametri prea mari => exploding gradient
- Metode de a inițializa parametrii:
  1. all 0-s: toate ponderile rețelei sunt inițializate cu 0 (sau orice altă valoare constantă).
     - Funcție de activare unde $f(0) \neq 0 \Rightarrow$ toți gradienții vor fi în aceeași direcție –problema simetriei. Nu putem învăța tot spațiul datelor cu această constrângere.
     - Funcție de activare unde $f(0) = 0 \Rightarrow$ toate ieșirile vor fi $0 \Rightarrow$ toți gradienții vor fi 0 și rețeaua nu va învăța nimic.

97

Parametri vs hiper-parametri – inițializatori

- Metode de a inițializa parametrii:
  2. Inițializare aleatoare:
     - $w_i \sim N(0,1)$
     - $w_i \sim U(0,1)$
     - Rezolvă problema simetriei, însă nu și problemele gradienților (vanishing/exploding)
  3. Inițializare Xavier/Glorot (după autorul articolului [7], Xavier Glorot):
     - $w_i \sim U(-\sqrt{\frac{\sigma}{fan_{in}+fan_{out}}}, \sqrt{\frac{\sigma}{fan_{in}+fan_{out}}})$
     - $w_i \sim N(0, \sqrt{\frac{2}{fan_{in}+fan_{out}}})$
     - Adaptată pentru funcțiile sigmoid și tanh.
     - Păstrează gradienții în aproximativ aceeași scară.

98

Parametri vs hiper-parametri – inițializatori

- Metode de a inițializa parametrii:
  4. Inițializare He (după autorul articolului [8], Kaiming He):
     - $w_i \sim N(0, \sqrt{\frac{4}{fan_{in}+fan_{out}}})$
     - Adaptată pentru funcțiile ReLU și variațiile sale.
     - Păstrează gradienții în aproximativ aceeași scară.

99

5. Baze de date

Bază de date = o grupare de elemente cu proprietăți comune. Reprezintă „experiența” pe care o întâlnește un algoritm de învățare conform definiției din [2] (slide #5).

$$D = \{((x_i, y_i)|T), 1 \leq i \leq M\}$$

input output sarcina dimensiunea

100

Baze de date

- Conțin, de obicei, exemple de date de intrare și de ieșire. Scopul este găsirea unui algoritm care poate găsi funcția de corespondență dintre cele două.

- O bază de date bună conține suficiente exemple (10 x nr. param.) și uniform distribuite între clase:
  - pentru regresie: 10 exemple per variabila de predicție
  - pentru clasificare: 1000 exemple per clasă

101

Baze de date – dataset splits

- În timpul antrenării modelului, antrenarea se face pe un subset al bazei de date, denumit generic „train”. Optimizarea hiper-parametrilor se face raportându-ne la rezultatele obținute testând modelul pe baza „val” ⇔ optimizăm în raport cu „val”.

train (70%)	val (15%)	test (15%)

- După găsirea celui mai bun set de hiper-parametri, modelul se re-antrenează pe baza „train+val” și se verifică puterea de generalizare a modelului obținut prin testarea pe baza „test”, nemaivăzută până atunci de către model.

train + val (85%)	test (15%)

102

Baze de date - parcurgere

Iterație = setul de procese realizate între două actualizări succesive ale ponderilor în timpul antrenării.

Batch = grup de instanțe din baza de date de la intrarea sistemului, ce poate fi rulat în paralel în rețea.

Batch size = numărul de instanțe dintr-un batch.

Epocă = termen ce semnifică o trecere completă a tuturor datelor din baza de date de antrenare prin algoritmul de învățare.

6. Optimizatori

Optimizator = algoritm utilizat pentru a optimiza (minimiza sau maximiza) o funcție obiectiv.

- (Batch, Stochastic, Mini-batch) Gradient Descent
- Momentum
- Learning rate scheduling
- Adagrad
- RMSProp
- Adam

Optimizatori – Batch Gradient Descent

- Pași Batch Gradient Descent. La fiecare iterație:
  - Se execută forward propagation;
  - Se calculează funcția de cost pentru toată baza de date:

$$J(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y_i, \hat{y}_i)$$

  - Se actualizează ponderile, pe baza gradientului și a ratei de învățare:

$$\boldsymbol{w} \coloneqq \boldsymbol{w} - \alpha \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \alpha \nabla_w J(\boldsymbol{w})$$

- Astfel calculat, gradientul reprezintă o aproximare perfectă a întregului model.
- Limitare: este nevoie să rulăm **toată** baza de date de antrenare pentru a face o singură actualizare => necesită resurse de calcul foarte mari.
- Convergența este lentă.
- 1 epocă = 1 iterație.

Optimizatori – Stochastic Gradient Descent

- Pași Stochastic Gradient Descent. La fiecare iterație:
  - Se execută forward propagation pe exemplul curent;
  - Se calculează funcția de cost pentru exemplul curent;
  - Se actualizează ponderile, pe baza gradientului și a ratei de învățare;
  - Se trece la următoarea iterație ⇔ următorul exemplu din baza de date și se repetă algoritmul.
- Astfel calculat, gradientul caracterizează doar câte un exemplu, în parte (nu toată baza de date) – foarte bun pentru exemplul curent, discutabil pentru întregul proces.
- Ponderile se actualizează rapid, însă zgomotul introdus (stochastic) este mai mare.
- 1 epocă = N iterații, unde N este numărul de exemple din baza de date.

Optimizatori – Mini-batch Gradient Descent

- Pași Mini-batch Gradient Descent. La fiecare iterație:
  - Se formează un grup (batch) de exemple de intrare. De obicei, numărul de exemple este o putere a lui 2: 32, 64, 128.
  - Pentru fiecare grup (batch):
    - Se execută forward propagation pe întregul batch;
    - Se calculează funcția de cost pentru întregul batch;
    - Se actualizează ponderile, pe baza gradientului și a ratei de învățare;
    - Se trece la următoarea iterație ⇔ următorul batch din baza de date și se repetă algoritmul.
- Astfel calculat, gradientul caracterizează câte un batch, în parte – compromis între toată baza de date și un singur exemplu.
- Ponderile se actualizează mai rapid decât în cazul batch GD, dar mai lent decât în cazul SGD. Zgomotul introdus este, de asemenea, între cele 2 metode.
- 1 epocă = $\frac{M}{N}$ iterații, unde M este numărul de exemple din baza de date și N este dimensiunea unui batch.

Optimizatori – Momentum

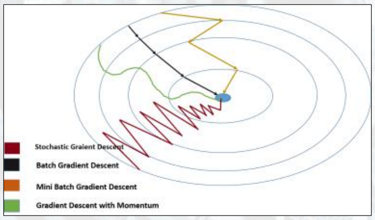
- O adaptare a algoritmilor Gradient Descent, în care se ține cont de actualizările precedente:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \alpha \nabla_w J(\boldsymbol{w}) + \beta\big(\boldsymbol{w}(t) - \boldsymbol{w}(t-1)\big)$$

  - $\beta \in [0,1]$ = factor de descompunere
  - $\beta = 0 \Rightarrow$ Gradient Descent;
  - $\beta = 0.9$ este o valoare bună de început
- Se obține o accelerare a convergenței și o atenuare a oscilației gradienților în direcții diferite ⇔ favorizează gradienții care se deplasează în aceeași direcție.

Sursă imagine: Bouzouitina Hamdi

Optimizatori – comparație



109

Optimizatori – programarea ratei de învățare

- Rata de învățare poate fi modificată programatic, conform unei reguli stabilite anterior:
  - Bazată pe timp:
$$\alpha(t+1) = \frac{\alpha(t)}{1+dt}$$
  - Bazată pe numărul iterației:
$$\alpha(t) = \alpha_0 d^{\left\lfloor \frac{1+t}{r} \right\rfloor}$$
  - Exponențială:
$$\alpha(t) = \alpha_0 e^{-dt}$$
- $\alpha_0 =$ rata inițială, $d =$ factor de descompunere, $r =$ rata de reducere

110

Optimizatori - Adaptive Gradient (AdaGrad)

- Publicat în 2011 [4], scalează rata de învățare pentru fiecare parametru, în parte:
$$w_i(t+1) = w_i(t) - \frac{\alpha}{\sqrt{G_{t,ii} + \varepsilon}} \nabla_w J(w_i(t))$$
$$G_{t,ii} = \sum_{\tau=1}^{t} \nabla_w J(w_i(\tau))^2$$
- Parametrii care sunt actualizați mai rar vor primi actualizări cu valori mai mari.
- Parametrii care sunt actualizați mai des vor primi actualizări cu valori mai mici.

111

Optimizatori – Root Mean Square Propagation (RMSProp)

- Adaptare a AdaGrad care previne acumularea prea mare a pătratelor gradientului:
$$v(w,t) \coloneqq \gamma v(w,t-1) + (1-\gamma)\left(\nabla_w J\big(w\,(t)\big)^2\right)$$
$$w(t+1) = w(t) - \frac{\alpha}{\sqrt{v(w,t)}} \nabla_w J(w(t))$$
- $v =$ medie glisantă de descompunere a amplitudinii gradienților
- $\gamma =$ factor de uitare
- Este unul dintre cei mai des utilizați optimizatori [5].

112

Optimizatori - Adaptive Moment Estimation (Adam)

- În prezent, cel mai popular algoritm (120k citări):
$$m_w(t+1) = \beta_1 m_w(t) + (1-\beta_1)\nabla_w J(w(t))$$
$$v_w(t+1) = \beta_2 v_w(t) + (1-\beta_2)(\nabla_w J(w(t)))^2$$
$$\hat{m}_w = \frac{m_w(t+1)}{1-{\beta_1}^t}, \qquad \hat{v}_w = \frac{v_w(t+1)}{1-{\beta_2}^t}$$
$$w(t+1) = w(t) - \alpha \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \varepsilon}$$
$$\beta_1 = 0.9, \qquad \beta_2 = 0.999, \qquad \alpha = 0.001, \qquad \varepsilon = 10^{-8}$$
- $\hat{m}_w$ și $\hat{v}_w$ sunt momentele estimate de gradele 1, respectiv 2, ale căror polarizare este corectată.
- Fiecare pondere va avea propria rată de învățare.

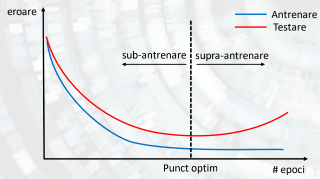
113

7. Regularizare

- Învățarea unui model are 2 faze:
  1. Antrenarea unui model pe datele de antrenare – optimizare;
  2. Testarea capacității de predicție pe un set de date (nemaiîntâlnit) de test – generalizare.
- Un model performant este definit de 2 caracteristici:
  1. Eroarea de antrenare este mică.
  2. Diferența dintre eroarea de antrenare și cea de testare este mică.

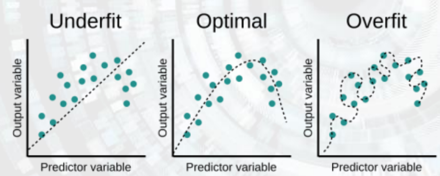
114

Regularizare



Sursă imagine: educative.io

Regularizare



Regularizare

Caz	Eroarea de antrenare	Eroarea de testare	Soluție
sub-antrenare (high bias)	↑	↑	- Creșterea complexității modelului (adăugare straturi/neuroni) - Creșterea numărului de iterații pentru antrenarea modelului
supra-antrenare (high variance)	↓	↑	- Adăugarea mai multor date în setul de antrenare - **Regularizare** - Oprire timpurie (early stopping)
antrenare optimă	↓	↓	N/A

Regularizare

Regularizare = modificarea algoritmului de învățare pentru a crește capacitatea de generalizare și a preveni supra-antrenarea.

➢Tipuri de regularizare:
1. Regularizare L1, L2;
2. Dropout [9];
3. Oprire timpurie (early stopping);
4. Augmentarea seturilor de date

Regularizare - L1, L2

➢Regularizare L1 (Regresie Least Absolute Shrinkage and Selection Operator – LASSO):

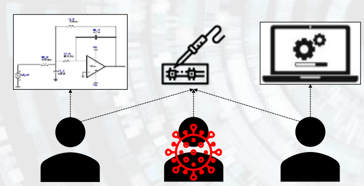
$$J'(w) = J(w) + \alpha \sum_{i=1}^{N} \|w_i\|_1 = J(w) + \alpha \sum_{i=1}^{N} |w_i|$$

➢Regularizare L2 (Regresie Ridge):

$$J'(w) = J(w) + \alpha \sum_{i=1}^{N} \|w_i\|_2^2 = J(w) + \alpha \sum_{i=1}^{N} {w_i}^2$$

Surse imagini:hardwarebee, flaticon, vectorstock

Regularizare – dropout



Scenariu - construirea unui robot:
- Echipă de 3 studenți;
- Fiecare student este specializat într-un domeniu;
- Unul dintre studenți nu mai poate lucra;
- Ceilalți 2 studenți trebuie să îi preia sarcinile;
- Cunoștințele celor 2 studenți rămași în echipă se îmbunătățesc.

Regularizare – dropout

- La fiecare iterație, un număr de neuroni sunt complet ignorați (dropped out) în procesul de învățare.
- Fie h activarea intermediară a unui neuron. În urma aplicării dropout cu probabilitatea p se obține:

$$h' = \begin{cases} 0, & cu\ probabilitatea\ p \\ \dfrac{h}{1-p}, & cu\ probabilitatea\ 1-p \end{cases}$$

- În timpul inferenței/testării, nu se mai utilizează dropout.

121

Regularizare

- Oprire timpurie:
  - Decidem să oprim antrenarea după un anumit număr de iterații (e.g., 5) după care eroarea de test nu s-a îmbunătățit.
- Augmentarea seturilor de date:
  - Aplicăm diferite transformări pe datele de intrare pentru a genera imagini artificiale care să mărească setul de date de antrenare. Exemple pentru rețele ce procesează imagini: mirror, crop, zoom, contrast etc.

122

8. Straturi populare

Strat = componentă topologică fundamentală formată din noduri, ce intră în alcătuirea unei rețele neuronale. Straturile preiau informația de la straturile precedente și o transmit următoarelor straturi. Înlănțuirea mai multor straturi formează o rețea neuronală.

- Input
- Dense
- Convolutional
- Pooling
- Recurrent
- Activations
- Dropout
- Batch normalization
- Output

123

Straturi populare

- Input layer:
  - Primul strat al oricărei rețele;
  - În el se încarcă datele de intrare;
  - Poate avea mai multe dimensiuni: $num \times width \times height \times channel$.
- Dense / fully-connected layer:
  - Strat „complet conectat" ⇔ fiecare neuron de intrare este conectat la fiecare neuron de ieșire;
  - Printre cele mai utilizate straturi, în special la ieșirea unei rețele;
  - Descris în rețeaua de neuroni multistrat (slide 51).

124

Straturi populare

- Convolutional layer:
  - Realizează „convoluția" dintre un semnal de intrare și un set de trăsături, denumit filtru;
  - În mod tradițional, se aplică pe date în format bi-dimensional (imagini), cu extensii 1D, 3D;
  - Este stratul principal în majoritatea modelelor de rețele neuronale pentru computer vision;
  - Are rolul de a extrage trăsături mai reprezentative și de a micșora dimensionalitatea datelor.
  - Cazuri speciale: convoluții dilatate, convoluții 1×1.
  - Invariante la translații

125

Straturi populare – convolutional layer

padding=1

0	0	0	0	0	0	0
0	2	1	0	1	1	0
0	1	1	0	2	0	0
0	0	0	1	1	1	0
0	1	2	2	0	2	0
0	2	0	1	2	1	0
0	0	0	0	0	0	0

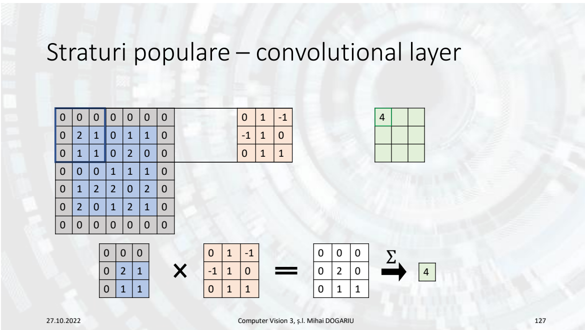
Trăsături de intrare (imagine)

0	1	-1
-1	1	0
0	1	1

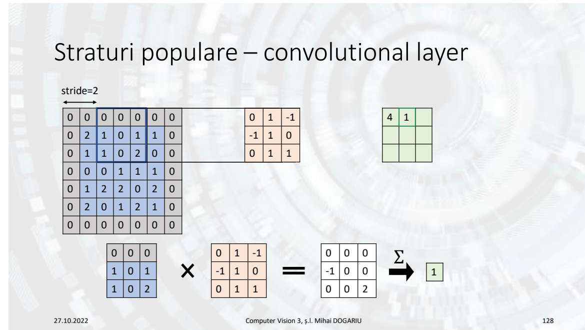
Filtru (kernel)

Trăsături de ieșire

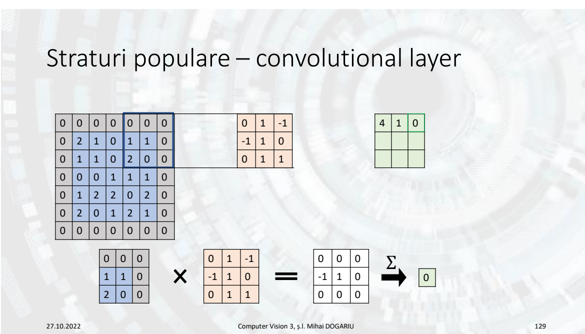
126



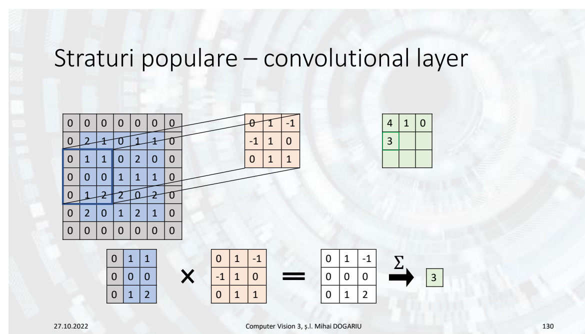
127



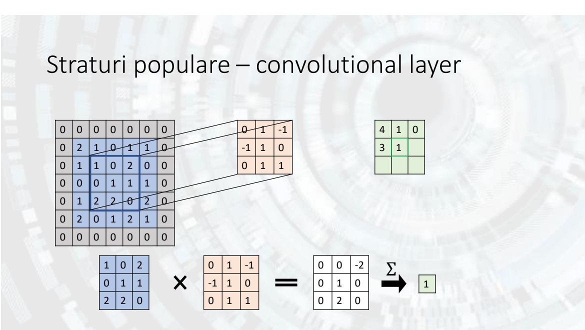
128



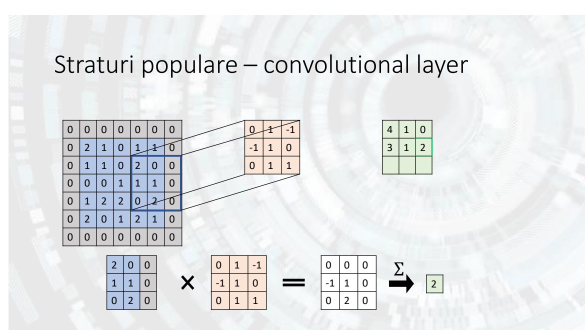
129



130

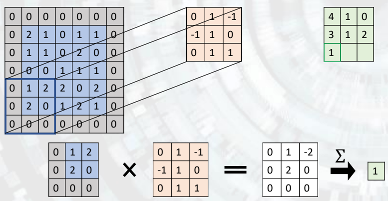


131

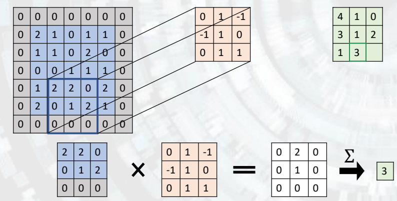


132

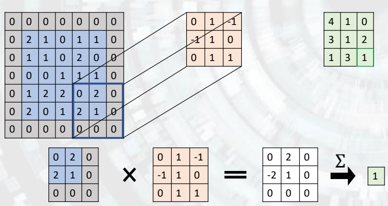
Straturi populare – convolutional layer



Straturi populare – convolutional layer



Straturi populare – convolutional layer

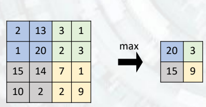


Straturi populare – convolutional layer

- Hiperparametri strat convoluțional:
  - Număr de filtre: K
  - Dimensiunea unui filtru: F
  - Pasul (stride): S
  - Bordura (padding): P
  - Pentru un input de dimensiuni $W_1 \times H_1 \times D_1$ se obține un output de dimensiuni $W_2 \times H_2 \times D_2$, unde:
    - $W_2 = \frac{W_1 - F + 2P}{S} + 1$
    - $H_2 = \frac{H_1 - F + 2P}{S} + 1$
    - $D_2 = K$

Straturi populare

- Pooling layer
  - Poate fi max/average pooling – conceptul este similar.
  - Are hiperparametri similari stratului convoluțional: dimensiunea unui filtru - F pasul (stride) - S, bordura (padding) - P



Straturi populare

- Recurrent layers:
  - Folosite pentru a modela secvențe de date ce se modifică în timp, e.g. vorbirea umană, secvențe video, serii temporale etc;
  - Păstrează informații despre starea precedentă a datelor;
  - Câteva exemple de astfel de straturi: LSTM, bidirectional LSTM, GRU, etc.
- Activation layers:
  - Toate funcțiile de activare (slide 77) sunt implementate ca straturi ce pot fi adăugate în componența unei rețele.

Straturi populare

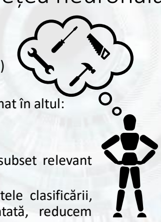
- Dropout layer - implementează mecanismul de regularizare Dropout (slide 121)
- Batch normalization [10] layer:
  - Normalizează trăsăturile la nivelul unui batch:
    - $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$ media valorilor din batch
    - $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)$ varianța valorilor din batch
    - $\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$ normalizarea valorilor din batch
    - $y_i = \gamma \hat{x_i} + \beta$ scalare și shiftare; γ și β sunt parametrii ce trebuie învățați
- Output layer:
  - Dualul stratului de intrare, reprezintă ieșirea rețelei neuronale

M2.4. Considerații practice

Dezvoltarea unui model de rețea neuronală

1. Alegerea datelor
   - Numărul exemplelor din setul de date:
     - General: 10x nr. de parametri ai modelului
     - Regresie: 10 exemple / variabilă de predicție
     - Clasificare: 1000 exemple / clasă
   - Calitatea exemplelor – etichetare făcută de specialiști, zgomot redus
   - Resurse pentru baze de date:
     - https://archive.ics.uci.edu/ml/index.php
     - https://www.kaggle.com/datasets
     - https://paperswithcode.com/datasets
     - https://datasetsearch.research.google.com/

Dezvoltarea unui model de rețea neuronală



2. Pre-procesarea datelor
   - Dataset split: train (70%), val (15%), test (15%)
   - Toate subset-urile conțin date asemănătoare
   - Formatarea datelor, trecerea lor dintr-un format în altul:
   .csv <-> .pkl <-> .json <-> database;
   - Date lipsă (Null, NaN) – cu ce le înlocuim?
   - Baze de date prea mari – folosim doar un subset relevant statistic pentru dezvoltare
   - Class imbalance – ponderăm diferit rezultatele clasificării, repetăm intrările din clasa slab reprezentată, reducem intrările din clasa puternic reprezentată, etc.
   - Normalizare – aducem descriptorii în aceeași gamă de valori

Dezvoltarea unui model de rețea neuronală

3. Antrenarea modelului
   - Încărcăm datele
   - Forward propagation
   - Calculăm funcția de cost
   - Backpropagation
   - Actualizăm ponderile
   - Rinse & repeat

Dezvoltarea unui model de rețea neuronală

4. Validarea modelului
   - Se testează modelul pe setul de date de validare pentru a-i testa capacitatea de generalizare.
   - Această validare nu se execută la fiecare iterație, ci o dată la N epoci (N=30, 100 etc.).
5. Optimizarea modelului
   - Hyperparameter tuning: ajustarea hiperparametrilor rețelei
   - Combaterea supra-antrenării cu metode de regularizare: regularizare L1, L2, dropout, oprire timpurie (early stopping), augmentarea seturilor de date
   - Pre-train + fine-tune
   - Feature transfer
   - La finalul optimizării se rulează algoritmul pe setul de date de test. Metricile oficiale sunt cele raportate pe setul de date de test.

Sfârșit M2

Bibliografie

[1] Gross, R. (2015). Psychology: The science of mind and behaviour 7th edition. Hodder Education.

[2] Mitchell, T. M. (1997). Machine learning (Vol. 1). McGraw-hill New York.

[3] Azevedo, F. A., Carvalho, L. R., Grinberg, L. T., Farfel, J. M., Ferretti, R. E., Leite, R. E., Jacob Filho, W., Lent, R., & Herculano-Houzel, S. (2009). Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. The Journal of comparative neurology.

[4] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).

[5] Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31.

[6] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[7] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

[8] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

[9] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958

[10] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR.