Învățare Automată - Tema 2
Clasificarea imaginilor folosind rețele neurale

Tudor Berariu
Facultatea de Automatică și Calculatoare
Univesitatea Politehnică București

15 aprilie 2017

1 Totul în mai puțin de zece rânduri

Pentru a rezolva această temă veți implementa rețele neurale *feed-forward* pentru clasificarea imaginilor. Veți implementa mai multe tipuri de straturi: complet conectat, de convoluție, și diferite straturi nonlineare. Toate rețelele vor fi antrenate cu *stochastic gradient descent*. Tema se poate implementa în orice limbaj de programare, dar nu este permisă folosirea bibliotecilor ce implementează rețele neurale. În schimb, este permisă și încurajată folosirea bibliotecilor care implementează calcule matriceale rapide (de exemplu `BLAS` sau `numpy`). Se vor testa diferite arhitecturi și diferite valori pentru hiperparametri (rata de învățare sau impulsul la bonus). Secțiunile următoare vă oferă nu doar detaliile teoretice legate de aceste rețele, ci și o sugestie de implementare. Nu este obligatoriu să vă structurați programul în modul sugerat.

2 Problema de rezolvat

În cadrul acestei teme se va rezolva o problemă clasică din învățarea automată, *clasificarea imaginilor*. Mai precis, dându-se un set de date ce conține imagini etichetate cu clasele corecte, se cere antrenarea unui model capabil să clasifice imagini noi.

Se va încerca învățarea claselor din două seturi de date: Chars74k[1] și CIFAR-10[2].

Observație asupra implementării convoluției

Spre deosebire de implementarea ineficientă din cadrul laboratorului, pentru stratul de convoluție se sugerează implementarea funcției `im2col` și realizarea convoluției printr-o singură înmulțire de matrice. Citiți secțiunea *Implementation as Matrix Multiplication* de aici[3].

[1] `http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/`
[2] `https://www.cs.toronto.edu/~kriz/cifar.html`
[3] `http://cs231n.github.io/convolutional-networks/#conv`

3 Antrenarea rețelei *feed-forward*

Petre, de unde ești de loc?[3]

3.1 Rețeaua neurală și straturile ei

O rețea neurală *feed-forward* implementează o funcție f_{net} care proiectează date dintr-un spațiu de intrare (imagini în cazul temei, $\mathbb{R}^{3\times32\times32}$) într-un spațiu de ieșire (10 clase în contextul temei, deci $\mathbb{R}^{10}$). Rețeaua este parametrizată de o mulțime de valori $\boldsymbol{\theta}$ care se modifică în timpul antrenării.

$$\mathbf{y} = f_{net}\left(\mathbf{x}, \boldsymbol{\theta}\right) \tag{1}$$

Pentru probleme de clasificare dimensiunea spațiului de ieșire este dată de numărul de clase. În general se dorește ca ieșirea rețelei să formeze o distribuție de probabilitate peste clasele învățate (Formula 2).

$$y_k(\mathbf{x}, \boldsymbol{\theta}) \approx P(C_k|\mathbf{x}) \tag{2}$$

Având un set de date compus dintr-un număr de exemple $\{\mathbf{x}^{(n)}, \mathbf{t}^{(n)}\}_{1\leq i\leq N}$, unde fiecare vector $\mathbf{t}$ este codificat *1-of-k* (are 1 pentru clasa corectă și zero în rest), se dorește maximizarea probabilității prezise de rețea pentru clasa corectă a fiecărui exemplu:

$$\mathbf{P}(\boldsymbol{\theta}) = \prod_{n=1}^{N}\prod_{k=1}^{K} y_k(\mathbf{x}^{(n)}, \boldsymbol{\theta})^{t_k^{(n)}} \tag{3}$$

Scopul antrenării este să găsirea unor parametri $\boldsymbol{\theta}$ astfel încât valoarea $\mathbf{P}$ să fie maximă. Echivalent se minimizează expresia din Formula 4:

$$E(\boldsymbol{\theta}) = -\log\left(\mathbf{P}\left(\boldsymbol{\theta}\right)\right) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_k^{(n)} \log\left(y_k\left(\mathbf{x}^{(n)}, \boldsymbol{\theta}\right)\right) \tag{4}$$

Partea de optimizare este descrisă în Secțiunea 3.4.

3.2 Arhitectura rețelei

O rețea neurală *feed-forward* formată din L straturi, în care fiecare strat l implementează o funcție f_l (Formula 5), realizează, de fapt, o compunere a acestor funcții (Formula 6).

$$\mathbf{y}_l = f_l\left(\mathbf{y}_{l-1}, \boldsymbol{\theta}_l\right) \tag{5}$$

$$\mathbf{y}_{net} = f_{net}(\mathbf{x}, \boldsymbol{\theta}) = f_L\left(f_{L-1}\left(f_{L-2}\left(\ldots f_1(\mathbf{x}, \boldsymbol{\theta}_1)\ldots\right), \boldsymbol{\theta}_{L-1}\right), \boldsymbol{\theta}_L\right) \tag{6}$$

Prin convenție $\mathbf{y}_{net} = \mathbf{y}_L$ și $\mathbf{y}_0 = \mathbf{x}$.

Parametrii rețelei sunt reprezentați de parametrii tuturor straturilor: $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots \boldsymbol{\theta}_L\}$.

[3] `https://www.youtube.com/watch?v=Ndd8AxxqvjI`

3.3 Folosirea rețelei pentru predicție

În pasul de predicție, rețeaua calculează strat cu strat ieșirile corespunzătoare unui exemplu dat la intrare (Algoritmul 1).

Algoritmul 1 Pasul *forward* la nivelul rețelei

funcția FORWARD(net, $\mathbf{x}$) ▷ Intrări: rețeaua net și exemplul $\mathbf{x}$
  $\mathbf{y}_0 \leftarrow \mathbf{x}$
  pentru $l \leftarrow 1 \dots L$ **execută**
    $\mathbf{y}_l \leftarrow net.layers[l].forward(\mathbf{y}_{l-1})$ ▷ Intrările stratului l sunt ieșirile stratului $l-1$
  întoarce $\mathbf{y}_L$ ▷ Întoarce ieșirile ultimului strat

3.4 Antrenarea rețelei

O variantă simplă de a optimiza parametrii rețelei pentru a minimiza eroarea (Formula 4) este de a deplasa acești parametri în sens opus gradientului. Aplicarea Formulei 7 până la atingerea unui minim local duce la algoritmul *stochastic gradient descent* (Algoritmul 2).

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta * \frac{\partial E}{\partial \boldsymbol{\theta}^{(t)}} \tag{7}$$

Deoarece calculul funcției de eroare pentru întreg setul de date este costisitor, aceasta este aproximată folosind un *mini-batch* (o parte din exemple).

Algoritmul 2 Stochastic gradient descent

funcția SGD(net, $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{t}^{(n)})\}_{1 \le n \le N}, \eta$) ▷ net, setul de date, rata de învățare
  repetă
    $\mathcal{B} \leftarrow$ un *mini-batch* de dimensiune b din $\mathcal{D}$
    pentru $l \leftarrow 1 \dots L$ **execută**
      $y_l \leftarrow net.layers[l].zeroGradients()$ ▷ Se inițializează gradienții cu zero
    pentru $(\mathbf{x}, \mathbf{t}) \leftarrow \mathcal{B}$ **execută** ▷ Pentru fiecare exemplu din *mini-batch*
      $\mathbf{y} \leftarrow forward(net, \mathbf{x})$ ▷ Se calculează ieșirea rețelei
      $\boldsymbol{\delta}_L \leftarrow \frac{\partial E}{\partial \mathbf{y}}$ ▷ Gradientul erorii în raport cu ieșirile rețelei
      $backpropagate(net, \mathbf{x}, \boldsymbol{\delta}_L)$ ▷ Eroarea se propagă prin rețea către intrări
    pentru $l \leftarrow 1 \dots L$ **execută**
      $net.layers[l].updateParameters(\eta)$ ▷ Se actualizează parametrii
  până când algoritmul converge

Pentru eroarea din Formula 4, derivata în raport cu ieșirile este:

$$\boldsymbol{\delta}_{Lk} = \frac{\partial E}{\partial y_k} = -\frac{t_k}{y_k} = \begin{cases} 0 & \text{, daca } t_k = 0 \\ -1/y_k & \text{, daca } t_k = 1 \end{cases} \tag{8}$$

3.5 Propagarea erorilor prin rețea

Pentru a putea aplica Algoritmul 2 trebuie calculat gradientul erorii în raport cu toți parametrii rețelei. Acest lucru se face cu algoritmul *backpropagation*, strat cu strat dinspre ieșire către intrare. Pentru fiecare strat l se calculează $\boldsymbol{\delta}_{l-1} \overset{not.}{=} \frac{\partial E}{\partial \mathbf{y}_{l-1}}$ și $\frac{\partial E}{\partial \boldsymbol{\theta}_l}$ pe baza $\boldsymbol{\delta}_l$.

Algoritmul 3 Backpropagation - propagarea gradienților prin rețea

funcția BACKPROPAGATE($net, \mathbf{x}, \boldsymbol{\delta}_L$)
  $\boldsymbol{y}_0 \leftarrow \mathbf{x}$
  pentru $l \leftarrow L \dots 1$ **execută**
    $\boldsymbol{\delta}_{l-1} \leftarrow net.layers[l].backward(\mathbf{y}_{l-1}, \boldsymbol{\delta}_l)$ ▷ Intern se acumulează $\frac{\partial E}{\partial \boldsymbol{\theta}_l}$

3.6 Sugestie de implementare

Un mod simplu de a implementa rețeaua este acela de scrie următoarele funcții pentru fiecare tip de strat.

`forward(x)` : Calculează ieșirea $\mathbf{y}$ a stratului curent pe baza intrărilor $\mathbf{x}$ și a parametrilor.

`backward(x, dEdy)` : Calculează derivatele parțiale ale erorii corespunzătoare ultimului exemplu în raport cu parametrii locali. Gradientul se adună valorii existente. (pentru a permite cumularea gradienților într-o secvență de apeluri)

`zeroGradients()` : Șterge gradienții erorii în raport cu parametrii. (pentru a permite apoi cumularea lor într-o secvență de apeluri `backward`)

`updateParameters(lr)` Actualizează parametrii locali folosind *stochastic gradient descent* cu rata de învățare primită. (Formula 7)

4 Tipurile de straturi

4.1 Straturi clasice

Modelul clasic al rețelelor neurale artificiale *feed-forward* este compus din alternări de straturi liniare complet conectate și straturi cu funcții de transfer nonlineare (logistică, tangentă hiperbolică). În problemele de clasificare, în general ultimul strat este unul de tip softmax.

4.1.1 Strat complet conectat

Calculul ieșirilor Un strat complet conectat (`FullyConnected`) proiectează un vector de intrare $\mathbf{x} \in \mathbb{R}^D$ într-un spațiu de ieșire $\mathbb{R}^K$. Formula 9 descrie calculul făcut de acest tip de strat pentru o unitate (un neuron) y_k.

$$y_k = \sum_{i=1}^{D} \theta_{ki} x_i + b_k, \qquad \forall k \in \{1, \dots, K\} \tag{9}$$

Ecuația 9 se poate scrie echivalent în formă matriceală ca în Formula 10.

$$\mathbf{y} = \boldsymbol{\Theta}\mathbf{x} + \mathbf{b} \tag{10}$$

Matricea $\boldsymbol{\Theta} \in \mathbb{R}^{K \times D}$ și vectorul $\mathbf{b} \in \mathbb{R}^K$ reprezintă parametrii stratului.

Inițializarea parametrilor Pentru rezultate bune la antrenarea rețelelor se recomandă inițializeare ponderilor cu metoda Xavier:

$$\Theta_{ij} \sim \mathcal{N}\left(0, \sqrt{\left(\frac{2}{D+K}\right)}\right) \tag{11}$$

Propagarea erorilor În pasul de propagare înapoi a gradienților pe baza $\boldsymbol{\delta}_y = \frac{\partial E}{\partial \mathbf{y}}$ se calculează derivatele parțiale ale erorii E în raport cu intrările $\mathbf{x}$ conform formulei 12 sau, echivalent, în formă matriceală conform formulei 13.

$$\frac{\partial E}{\partial x_i} = \sum_{k=1}^{K} \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_i} = \sum_{k=1}^{K} \delta_{y_k} \theta_{ki} = \boldsymbol{\delta}_y^{\mathrm{T}} \boldsymbol{\theta}_i \tag{12}$$

$$\frac{\partial E}{\partial \mathbf{x}} = \boldsymbol{\Theta}^{\mathrm{T}} \boldsymbol{\delta}_y \tag{13}$$

Similar, gradientul erorii în raport cu parametrii $\boldsymbol{\Theta}$ și $\mathbf{b}$:

$$\frac{\partial E}{\partial \boldsymbol{\Theta}} = \mathbf{x} \boldsymbol{\delta}_y^{\mathrm{T}} \tag{14}$$

$$\frac{\partial E}{\partial \mathbf{b}} = \boldsymbol{\delta}_y \tag{15}$$

4.1.2 Tanh

Un strat de tip `TanH` aplică funcția tangentă hiperbolică individual pe fiecare valoare a intrărilor. Valoarea tangentei hiperbolice într-un punct este dată de Formula 16, iar derivata ei de Formula 17.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{16}$$

$$tanh'(x) = 1 - tanh(x)^2 \tag{17}$$

Calculul ieșirilor

$$y_i = tanh(x_i) \qquad \forall i \tag{18}$$

Propagarea erorilor

$$\boldsymbol{\delta}_x = (1 - \mathbf{y} \odot \mathbf{y}) \odot \boldsymbol{\delta}_y \tag{19}$$

Stratul `TanH` nu are parametri.

4.1.3 SoftMax

Calculul ieșirilor Stratul `SoftMax` este folosit în problemele de clasificare atunci când se dorește ca ieșirea să fie codificată *1-of-k*, iar valorile acesteia să reprezinte o distribuție de probabilitate posterioară peste mulțimea claselor. Dându-se un vector de intrare $\mathbf{x} \in \mathbb{R}^K$, vectorul de ieșire va avea aceeași dimensiune $\mathbf{y} \in \mathbb{R}^K$ și va fi calculat ca în Formula 20.

$$y_k = \frac{e^{x_k}}{\sum_{k'=1}^{N} e^{x_{k'}}} \tag{20}$$

Propagarea erorilor Cum stratul `SoftMax` nu are parametri, în pasul *backward* trebuie calculat doar $\boldsymbol{\delta}_x$ pe baza $\boldsymbol{\delta}_y$. Formula 21 descrie acest calcul. Pentru o demonstrație matematică, citiți Secțiunea A.

$$\delta_{xk} = y_k \left(\delta_{y_k} - Z\right) \tag{21}$$

unde $Z = \sum_{k'} \delta_{y_{k'}} y_{k'}$.

4.1.4 Strat de linearizare

Pentru a putea trece de la straturi de convoluție la straturi complet conectate, *forma* activărilor trebuie schimbată din volum (tensor 3D) în vector. Într-un strat de tipul `Linearize` un volum $\mathbf{x} \in \mathbb{R}^{M \times H \times W}$ se transformă într-un vector $\mathbf{y} \in \mathbb{R}^{MHW}$. În pasul de propagare înapoi a gradienților, se petrece procesul invers: un vector $\boldsymbol{\delta}_y \in \mathbb{R}^{NHW}$ trebuie rearanjat într-un volum $\boldsymbol{\delta}_x \in \mathbb{R}^{N \times H \times W}$. Evident, acest strat nu are parametri și nu efectuează niciun calcul, doar schimbă aspectul unor tensori.

$$y_{m(HW)+iW+j} = x_{m,i,j} \quad 0 \leq m < M, 0 \leq i < H, 0 \leq j < W \tag{22}$$

$$\delta_{x_{m,i,j}} = \delta_{y_{m(HW)+iW+j}}, \quad 0 \leq m < M, 0 \leq i < H, 0 \leq j < W \tag{23}$$

4.2 Rețele convoluționale

Rețelele convoluționale sunt rețele neurale dedicate imaginilor având o conectivitate specială ce urmărește obținerea echivarianței la translație. Rețelele convoluționale prelucrează volume de neuroni alternând straturi de convoluție, de redimensionare (*downsampling*) și `ReLU` (*rectified linear unit*). Spre final, rețelele convoluționale au unul sau două straturi complet conectate.

Consultați și pagina aceasta[4], una dintre cele mai bune prezentări ale rețelelor convoluționale.

4.2.1 Strat de convoluție

Straturile de convoluție `Conv` sunt specifice prelucrărilor de imagini, prelucrând neuroni dispuși în volume (tensori 3D). Volumele conțin hărți de caracteristici (*feature maps*) care reprezintă activări ale aceluiași detector în toate regiunile hărților de intrare. Acest lucru se întâmplă deoarece toți neuronii de pe o hartă au aceiași parametri.

Fiecare hartă descrie un filtru care se conectează pe o suprafață mică, dar pe toată adâncimea volumului de intrare. Un neuron se calculează printr-un produs scalar între parametrii lui (ai hărții căreia îi aparține) și neuronii din volumul de intrare de care este conectat. Ceilalți neuroni de pe aceeași hartă se calculează prin produse scalare între aceiași parametri și alte zone din volumul de intrare.

Un strat de convoluție este parametrizat de dimensiunea filtrului aplicat (***k**ernel size*) și de pasul (***s**tride*) dintre două aplicări ale filtrului.

Să presupunem că un astfel de strat primește la intrare M hărți de dimensiune $H \times W$. De exemplu, pentru o imagine RGB de dimensiune 32×32: $M = 3, H = 32, W = 32$. Mai departe să presupunem că fiecare hartă are un filtru de dimensiune $k = 3$ și îl aplică din 2 în 2 pixeli ($s = 2$). Dacă stratul calculează $N = 10$ hărți la ieșire, atunci dimensiunea volumului de ieșire va fi $N \times H' \times W'$:

[4] `http://cs231n.github.io/convolutional-networks/`

- adâncime: $N = 10$ hărți
- înălțime: $H' = (H - k)/s + 1$
- înălțime: $W' = (W - k)/s + 1$

Calculul realizat într-un strat de convoluție este cel din Formula 24. Fiecare hartă din volumul de ieșire este conectată la toate hărțile din volumul de intrare. De aceea, fiecare valoare din volumul de ieșire va depinde de $M \times k \times k$ valori din volumul de intrare.

$$y_{n,i,j} = \sum_{m=0}^{M-1} \sum_{p=0}^{k-1} \sum_{q=0}^{k-1} \theta_{n,m,p,q} \cdot x_{m,(i \cdot s)+p,(j \cdot s)+q} + b_n \tag{24}$$

Numărul de parametri ai unui strat de convoluție este $N \times (k \times k + 1)$.
Citiți și aici: `http://cs231n.github.io/convolutional-networks/#conv`.

Propagarea erorilor

$$\frac{\partial E}{\partial \theta_{n,m,p,q}} = \sum_{i,j} x_{n,m,i \cdot s+p,j \cdot s+q} \cdot \delta_{y_{n,i,j}} \tag{25}$$

$$\delta_{xm,i,j} = \frac{\partial E}{\partial x_{m,i,j}} = \sum_{0 \leq n < N,p,q} \theta_{n,m,p,q} \cdot \delta_{y_{n,m,(i-p)/s,(j-q)/s}} \tag{26}$$

4.2.2 Max pooling

Calculul ieșirilor Un strat `MaxPooling` reduce dimensiunea unui volum de intrare într-unul cu același număr de hărți, dar de dimensiuni mai mici. Mai precis, fiecare hartă de intrare este împărțită în petice, iar fiecărui petic din hărțile de intrare îi corespunde o singură valoare la ieșire.

Stratul nu are parametri, dar are un hiper-parametru: latura unui petec (*stride*).

$$y_{m,i,j} = \max(x_{m,i \cdot stride,j \cdot stride}, \ldots x_{m,i \cdot (stride+1)-1,j \cdot (stride+1)-1}) \tag{27}$$

Propagarea erorilor În pasul de propagare înapoi a erorilor, gradientul *se transmite* intrării care a avut valoarea maximă într-un petice. Un gradient $\boldsymbol{\delta}_y$ de dimensiune $M \times (H/stride) \times (W/stride)$ este transformat într-un volum $\boldsymbol{\delta}_x$ de dimensiune $M \times H \times W$.

$$\delta_{xm,i,j} = \begin{cases} \delta_{y_{m,\lfloor i/stride \rfloor,\lfloor j/stride \rfloor}} & \text{, daca } \delta_{xm,i,j} == y_{m,\lfloor i/stride \rfloor,\lfloor j/stride \rfloor} \\ 0 & \text{, altfel} \end{cases} \tag{28}$$

4.2.3 ReLU

Calculul ieșirilor Un strat de tip `ReLU` (*rectified linear unit*) primește un tensor $\mathbf{x}$ și produce la ieșire un tensor $\mathbf{y}$ cu aceleași dimensiuni. O valoare de ieșire y_i se calculează conform formulei 29.

$$y_i = relu(x_i) = \max(x_i, 0) \tag{29}$$

Straturile de tip `ReLU` nu au parametri.

Propagarea erorilor

$$\delta_{xi} = \begin{cases} \delta_{y_i} & , \text{daca } y_i > 0 \\ 0 & , \text{altfel} \end{cases} \tag{30}$$

5 Cerințe

Cerința 1. Descărcați seturile de date, și standardizați datele (astfel încât acestea să aibă media 0 și varianța 1). Fiecare imagine va fi prezentată într-un volum $3 \times 32 \times 32$, iar fiecare vector t va fi codificat *1-of-k*.

Cerința 2. Implementați straturile de tip `Linearize`, `FullyConnected`, `TanH` și `SoftMax`.

Cerința 3. Implementați funcțiile pentru rețea și algoritmul *stochastic gradient descent* pentru eroarea din Formula 4.

Cerința 4. Testați diferite arhitecturi ale rețelei pe cele două seturi de date. Variați numărul de straturi, numărul de unități de pe fiecare strat și valoarea ratei de învățare. Faceți grafice în care să arătați evoluția acurateței rețelei pe setul de date de antrenare și pe setul de date de test (datele neimplicate în optimizare). Calculați și matricea de confuzie.

Arhitectură sugerată: `Linearize`, `FullyConnected`(300), `TanH`, `FullyConnected`(100), `SoftMax`.

Cerința 5. Implementați straturile `Conv`, `MaxPooling` și `ReLU`.

Cerința 6. Testați diferite arhitecturi de rețele convoluționale pe cele două seturi de date. Variați numărul de straturi, numărul de hărți de pe un strat de convoluție și rata de învățare. Cerința se consideră rezolvată dacă ați variat fiecare hiperparametru cel puțin o dată păstrând ceilalți hiperparametri constanți. Arhitectura sugerată: Le-Net 5.

Cerința BONUS. Adăugați impuls (*momentum*) algoritmului de învățare și testați impactul pe care îl are pentru arhitecturile testate la rezolvarea cerințelor 4 și 6. O valoare bună pentru rata impulsului μ este 0.9.

$$\Delta\boldsymbol{\theta}^{(t)} = -\eta \frac{\partial E}{\partial \boldsymbol{\theta}^{(t)}} + \mu \cdot \Delta\boldsymbol{\theta}^{(t-1)} \tag{31}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}^{(t)} \tag{32}$$

A Propagarea erorilor printr-un strat `SoftMax`

Fie funcția `SoftMax` ce transformă un vector $\mathbf{x}$ de dimensiune K într-un vector $\mathbf{y}$ de aceeași dimensiune (Formulele 33,34).

$$\mathbf{y} = softmax(\mathbf{x}) \tag{33}$$

$$y_k = \frac{e^{x_k}}{\sum_{k'} e^{x_{k'}}} \tag{34}$$

Cerința	**Punctaj**
Cerința 1 (pregătirea datelor)	**0.5 p.**
Cerința 2 (straturi simple)	**2 p.**
Cerința 3 (antrenare rețea)	**2 p.**
Cerința 4 (testare rețea simplă)	**1.5 p.**
Cerința 5 (rețea convoluțională)	**2.5 p.**
Cerința 6 (testare rețea convoluțională)	**1.5 p.**
Bonus (impuls)	**2 p.**

Tabela 1: Barem

Fiind dat gradientul unei funcții E în raport cu ieșirile $\mathbf{y}$, trebuie dedusă o formulă de calcul pentru gradientul erorii E în raport cu intrările $\mathbf{x}$.

$$\boldsymbol{\delta}_y \overset{not.}{=} \frac{\partial E}{\partial \mathbf{y}} \tag{35}$$

$$\boldsymbol{\delta}_x \overset{not.}{=} \frac{\partial E}{\partial \mathbf{x}} \tag{36}$$

Ecuația 37 exprimă formula de calcul pentru o valoare din vectorul $\boldsymbol{\delta}_x$.

$$\delta_{xk} = \sum_{k'} \frac{\partial E}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial x_k} = \sum_{k'} \delta_{y_{k'}} \frac{\partial y_{k'}}{\partial x_k} \tag{37}$$

$$\frac{\partial y_{k'}}{\partial x_k} = \frac{\mathbb{I}(k == k') e^{x_k} \left(\sum_{k''} e^{x_{k''}}\right) - e^{x_{k'}} e^{x_k}}{\left(\sum_{k''} e^{x_{k''}}\right)^2} = \begin{cases} y_k - y_k y_{k'} & \text{, daca } k == k' \\ -y_k y_{k'} & \text{, daca } k \neq k' \end{cases} \tag{38}$$

Înlocuind Formula 38 în Formula 37:

$$\delta_{xk} = \sum_{k'} \delta_{y_{k'}} \frac{\partial y_{k'}}{\partial x_k} = y_k \left(\delta_{y_k} - \sum_{k'} \delta_{y_{k'}} y_{k'} \right) = y_k \left(\delta_{y_k} - Z \right) \tag{39}$$

unde $Z = \sum_{k'} \delta_{y_{k'}} y_{k'}$.