Rețele neuronale

IA 2023/2024

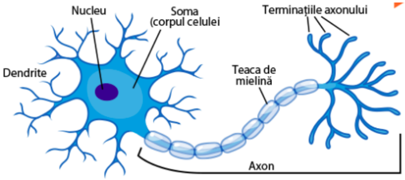
Conținut

Istoric

- McCulloch&Pitts '43 propun primul model matematic al unui neuron artificial
  Nu poate învăța, parametrii se stabilesc analitic
- Minsky '51 primul circuit electronic construit ca o rețea neuronală artificială (subcircuite ce funcționează ca niște neuroni interconectați)
- Rosenblatt '58 dezvolta Perceptronul, prima rețea neuronală funcțională
- Hinton '06 pune bazele *Deep Neural Network*

Rețele neuronale artificiale

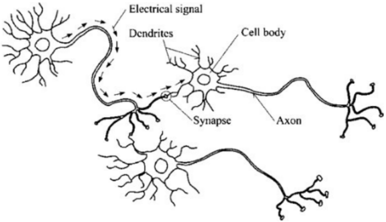
- Sunt inspirate din modul de structurare și funcționare a creierului



Încercarea de a reproduce inteligența (comportamentul unui neuron biologic).

Rețele neuronale artificiale

Un neuron se conectează cu alți neuroni prin intermediul dendritelor. Neuronii comunică între ei prin intermediul sinapselor (excitatorii sau inhibitorii). Neuronul se poate activa și produce un semnal electric care e transmis mai departe prin axon.



Interconectarea neuronilor asigură puterea de calcul.

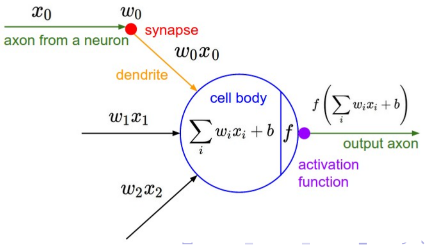
Rețele neuronale artificiale

Unitate funcțională (neuron artificial): un model computațional simplificat al neuronului

- semnale de intrare
- ponderi sinaptice atașate conexiunilor
- prag de activare
- ieșire

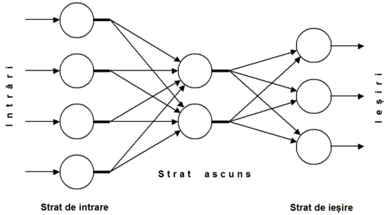
Analogii

RN biologică	RN artificială
corpul celulei	neuron
dendrite	intrări
axon	ieșire
sinapsă	pondere



Rețele neuronale artificiale

- Un ansamblu de unități funcționale (neuroni) interconectate



- Antrenarea presupune determinarea parametrilor rețelei, utilizând date de antrenare

- Sunt sisteme adaptive de tip "cutie neagră" care extrag un model printr-un proces de învățare.

Metode de învățare

- Supervizată (clasificare, regresie)
  - Exemple de antrenare etichetate
  - Scop: estimarea parametrilor care minimizează eroarea (diferența între răspunsurile corecte și cele produse de rețea)
- Nesupervizată (clusterizare, asociere, reducerea dimensionalității)
  - Date de antrenare care nu sunt etichetate
  - Scop: obținerea de informații

Aplicații: Clasificare

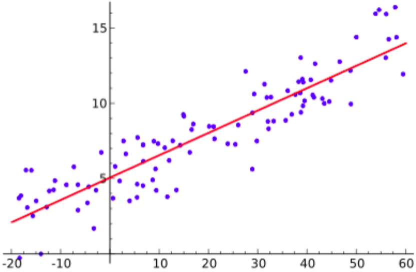
Dată o mulțime de instanțe (atribute, etichete), să se identifice clasa la care aparține o instanță nouă.
(supervizată)

Exemplu: identificarea speciei din care face parte o floare de iris

- atribute: lungime și lățime sepale/petale
- clase: *Iris versicolor*, *Iris setosa*, *Iris virginica*

Aplicații: Regresie

Să se determine relația dintre două sau mai multe variabile, dat un set de date de antrenament (aproximarea unei funcții)

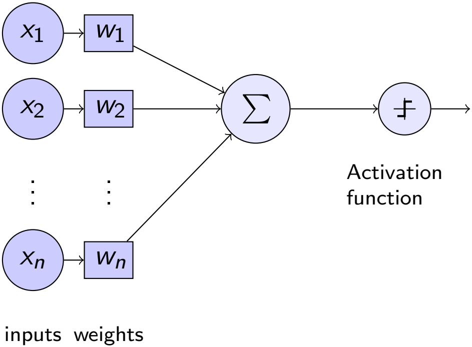


Diferența dintre clasificare și regresie: tipul ieșirii (discret vs. continuu)

Conținut

Perceptronul (Rosenblatl, 1958)

Intrare: un vector de valori reale x_i
Calculează o combinație liniară a acestora.

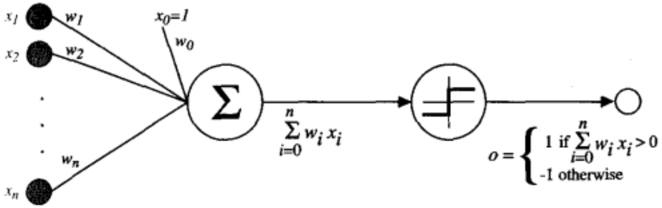


$w_1, \ldots w_n$ ponderi (const. reale) atașate conexiunilor; w_i contribuția intrării x_i la rezultat

Perceptronul

Intrare: un vector de valori reale x_i
Calculează o combinație liniară a acestora.
Returnează 1, dacă rezultatul e mai mare decât un prag ($-w_0$), -1 altfel.

$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{daca } w_0 + w_1 x_1 + \dots w_n x_n > 0 \\ -1 & \text{altfel} \end{cases} \tag{1}$$



Învățarea unui perceptron: alegerea ponderilor $w_0, \dots, w_n$.

Perceptron

Notație simplificată: o intrare constantă $x_0 = 1$.
$\sum_{i=0}^{n} w_i x_i > 0$, sau $\overrightarrow{w} \cdot \overrightarrow{x} > 0$.

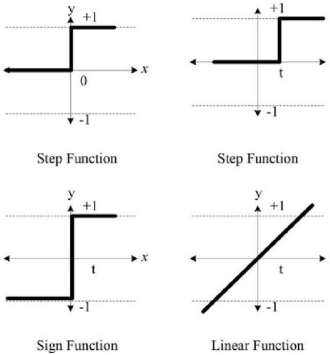
$o(\overrightarrow{x}) = f(\overrightarrow{w} \cdot \overrightarrow{x})$

Funcția de activare treaptă

$$f(y) = \begin{cases} 1 & \text{daca } y \geq 0 \\ 0 & \text{altfel} \end{cases}$$

Funcția de activare semn

$$f(y) = \begin{cases} 1 & \text{daca } y \geq 0 \\ -1 & \text{altfel} \end{cases}$$

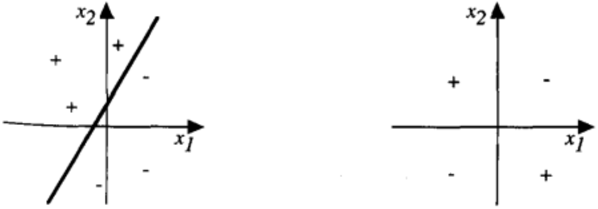


Perceptron: un neuron artificial care utilizează funcția de activare treaptă/semn

Puterea de reprezentare a perceptronilor

- Scopul: să clasificăm intrările $x_1, \ldots x_n$ în 2 clase
- Perceptronul: un hiperplan care împarte spațiul vectorilor de intrare (n-dimensional) în două regiuni

Ecuația hiperplanului: $\overrightarrow{w} \cdot \overrightarrow{x} = 0$



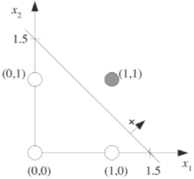
(a) (b)

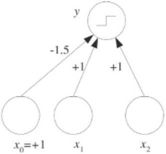
Separabile liniar
(pentru un perceptron cu două intrări)

Puterea de reprezentare a perceptronilor

Un perceptron poate fi utilizat pentru a reprezenta funcții booleene. Pentru a reprezenta funcția AND, setăm ponderile, spre ex. $w_0 = -1.5, w_1 = w_2 = 1$

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1

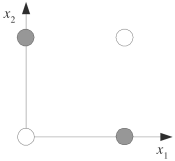




Puterea de reprezentare a perceptronilor

Funcția XOR ($1 \Leftrightarrow x_1 \neq x_2$) nu poate fi reprezentată de un singur perceptron.

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0



Orice funcție booleană poate fi reprezentată de o rețea de unități interconectate.

Conținut

Regula de antrenare a perceptronului

- Învățarea ponderilor: identifică vectorul de ponderi a.i. perceptronul să returneze ieșirea corectă pentru fiecare exemplu de antrenare.
- Generează ponderi aleatoare,

  calculează ieșirea pentru fiecare exemplu de antrenare,
  modifică ponderile atunci când clasifică greșit un exemplu.

  Repetă acest procedeu până când perceptronul clasifică corect exemplele de antrenare.

Regula de antrenare a perceptronului

Ponderile sunt modificate conform *regulii de antrenare a perceptronului*:

$$w_i \leftarrow w_i + \Delta w_i$$

unde

$$\Delta w_i = \eta(t - o)x_i$$

t este ieșirea dorită pentru exemplul de antrenare, o este ieșirea generată de perceptron, η *rata de învățare* (const. pozitivă)

Intuiție (regula de antrenare a perceptronului)

- Dacă exemplul este clasificat corect $t - o = 0; \Delta w_i = 0 \rightarrow$ ponderile nu sunt actualizate
- Dacă perceptronul returnează -1 când ieșirea corectă este $+1$ și $\eta = 0.1, x_i = 0.8$, atunci $\Delta w_i = 0.1(1 - (-1))0.8 = 0.16$
- Dacă perceptronul returnează $+1$ cand ieșirea corectă este -1, atunci ponderea scade

Regula de antrenare a perceptronului

Atunci când exemplele de antrenare sunt separabile liniar și η suficient de mic,

procedura converge (considerând un nr. finit de aplicări a regulii de antrenare a perceptronului)

la un vector de ponderi care clasifică toate exemplele de antrenare.

Conținut

Regula delta

Regula de antrenare a perceptronului poate eșua dacă exemplele nu sunt separabile liniar.

Regula delta: utilizează *Gradient descent* pentru a căuta în spațiul vectorilor de ponderi.

Considerăm o unitate liniară pentru care ieșirea este $o(\overrightarrow{x}) = \overrightarrow{w} \cdot \overrightarrow{x}$.

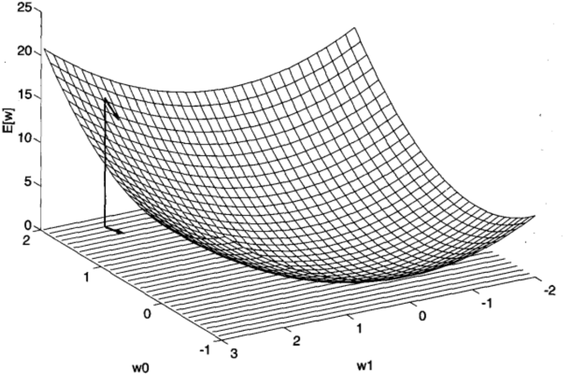
Eroarea de antrenare pentru un vector de ponderi w:

$$E(\overrightarrow{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

unde D mulțimea datelor de antrenare, t_d ieșirea dorită pentru exemplul d, o_d ieșirea unității liniare pentru d.

Vizualizarea spațiului de ipoteze

Suprafața erorii are forma parabolica, cu un minim global.



Gradient descent: modifică în mod repetat vectorul de ponderi. La fiecare pas, vectorul este modificat în direcția care produce cea mai abruptă coborâre. Acest proces continuă până la atingerea erorii minime globale.

Gradient descent

Gradientul specifică direcția care produce cea mai abruptă ascensiune în E.

$$\nabla E(\overrightarrow{w}) = [\frac{\delta E}{\delta w_0}, \frac{\delta E}{\delta w_1}, \ldots, \frac{\delta E}{\delta w_n}]$$

Regula de antrenare pentru *Gradient descent*: $\overrightarrow{w} \leftarrow \overrightarrow{w} + \Delta \overrightarrow{w}$, unde $\Delta \overrightarrow{w} = -\eta \nabla E(\overrightarrow{w})$, η este *rata de învățare* (const. pozitivă).

$$w_i = w_i + \Delta w_i, \quad \Delta w_i = -\eta \frac{\delta E}{\delta w_i}$$

Gradient descent

$$
\begin{aligned}
\frac{\delta E}{\delta w_i} &= \frac{\delta}{\delta w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}\frac{\delta}{\delta w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\delta}{\delta w_i}(t_d - o_d) \\
&= \sum_{d\in D}(t_d - o_d)\frac{\delta}{\delta w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\delta E}{\delta w_i} &= \sum_{d\in D}(t_d - o_d)(-x_{id})
\end{aligned}
$$

x_{id} componenta x_i a exemplului de antrenare d.

Actualizarea ponderii cu $\Delta w_i = \eta\sum_{d\in D}(t_d - o_d)x_{id}$.

Gradient descent

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i \tag{T4.1}$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i \tag{T4.2}$$

Stochastic gradient descent

Problemele algoritmului *Gradient descent*:

- convergență lentă
- existența mai multor minime locale

Stochastic gradient descent: actualizarea ponderilor incremental, calculând eroarea pentru fiecare exemplu individual

$$\Delta w_i = \eta(t - o)x_i$$

unde t valoarea dorită, o ieșirea reală, x_i a i-a intrare pentru exemplul de antrenare

Ecuația T4.1 este înlocuită cu $w_i \leftarrow w_i + \eta(t - o)x_i$.

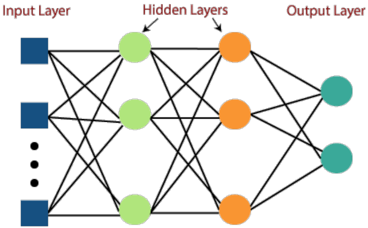
Regula de antrenare $\Delta w_i = \eta(t - o)x_i$ se mai numește regula delta/ regula LMS (*least-mean-square*)/regula Adaline.

Conținut

Rețele neuronale multi-strat

O rețea neuronală cu propagare înainte (*feed-forward*) cu

- un strat de intrare
- unul sau mai multe straturi ascunse
- un strat de ieșire

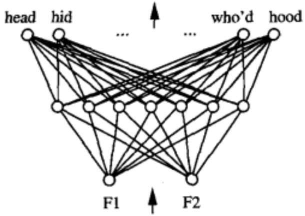


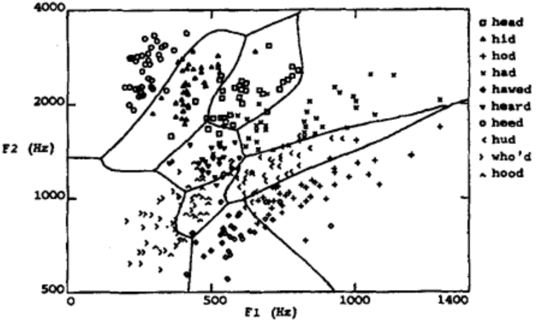
- Semnalele de intrare sunt propagate înainte prin straturile rețelei
- Calculele se realizează în neuronii din straturile ascunse și din stratul de ieșire

Rețele neuronale multi-strat

Pot exprima suprafețe de decizie neliniare.

Exemplu: Rețea antrenată să recunoască între 10 vocale ("h_d").



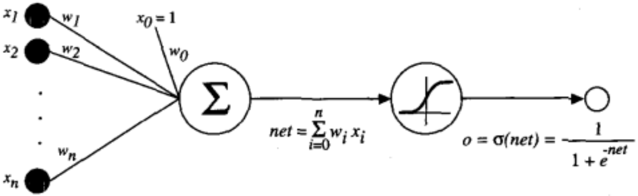


Semnalul vocal este reprezentat de doi parametri numerici, obtinuți din analiza spectrala a sunetului. Punctele din graficul din dreapta sunt exemplele de testare.

Proprietatea de aproximare universală

- O rețea neuronală cu un strat ascuns, cu un nr. posibil infinit de neuroni, poate aproxima orice funcție reală continuă
- Un strat suplimentar poate însă reduce foarte mult nr. de neuroni necesari în straturile ascunse

Unitate sigmoid

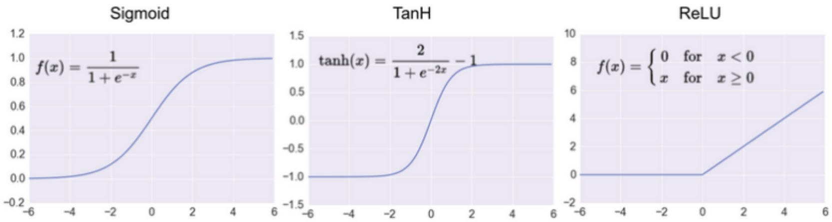


$$o = \sigma(\overrightarrow{w} \cdot \overrightarrow{x}), \qquad \text{unde } \sigma(y) = \frac{1}{1 + e^{-y}}$$

σ funcția sigmoidă; derivata $\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))$

Funcții de activare neliniară

- Funcția sigmoidă (logistică)
  $f(x) = \frac{1}{1+e^{-x}}, \quad f'(x) = f(x)(1 - f(x))$
- Funcția sigmoidă bipolară (tangenta hiperbolică)
  $f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}, \quad f'(x) = 1 - f(x)^2$
- Funcția ReLU (Rectified Linear Unit)
  $f(x) = \begin{cases} 0 & \text{daca } x < 0 \\ x & \text{daca } x \geq 0 \end{cases}, \quad f'(x) = \begin{cases} 0 & \text{daca } x < 0 \\ 1 & \text{daca } x \geq 0 \end{cases}$



Perceptron multi-strat: proprietăți

- Un perceptron cu un singur strat are aceleași limitări chiar dacă folosește o funcție de activare neliniară
- Un perceptron multi-strat cu funcții de activare liniare este echivalent cu un perceptron cu un singur strat
  - o combinație liniară de funcții liniare este tot o funcție liniară
    ex: f(x)=2x+1, g(y)=y-3, g(f(x))=(2x+1)-3=2x-2

Conținut

Algoritmul *Backpropagation*

- Rumelhart, Hinton& Williams, '86
- Învață ponderile într-o rețea multi-strat. Folosește *Gradient descent* pentru a minimiza eroarea pătratică între ieșirea rețelei și valorile dorite.
- Deoarece avem rețele cu mai multe unități de ieșire, redefinim E

$$E(\overrightarrow{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

unde *outputs* mulțimea de unități de ieșire, t_{kd} și o_{kd} valoarea dorită, respectiv, de ieșire asociată cu unitatea de ieșire k și exemplul de antrenare d.

Algoritmul *Backpropagation*

Are două faze:

- Rețeaua primește vectorul de intrare și propagă semnalul înainte, strat cu strat, până se generează ieșirea
- Semnalul de eroare este propagat înapoi, de la stratul de ieșire către stratul de intrare, ajustându-se ponderile rețelei

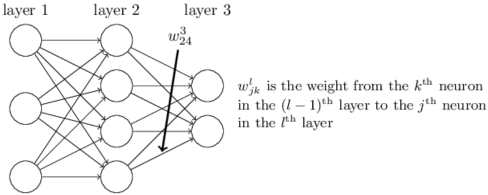
Backpropagation

Pașii algoritmului:

- Inițializarea: alege numărul de intrări, unități ascunse și de ieșire; inițializează ponderile și pragurile cu valori aleatorii mici
  - în general, pot fi valori din intervalul [-0.1, 0.1]

- Activarea
  - se activează rețeaua prin aplicarea vectorului de antrenare $\overrightarrow{x}$
  - se calculează ieșirile neuronilor din stratul ascuns
  - se calculează ieșirile neuronilor din stratul de ieșire $o = \sigma(\overrightarrow{w} \cdot \overrightarrow{x})$

Backpropagation



- Ieșirile neuronilor din stratul ascuns

$$o_h = \sigma(\sum_{i=0}^{n} w_{hi} x_i)$$

- Ieșirile neuronilor din stratul de ieșire

$$o_k = \sigma(\sum_{i=0}^{m} w_{ki} o_i)$$

Backpropagation

Actualizează fiecare pondere w_{ji} proporțional cu rata de învățare η, valoarea de intrare x_{ji} și eroarea δ_j.

- Pentru neuronii de ieșire
  - se calculează gradienții de eroare ai neuronilor din stratul de ieșire
    Pentru unitatea de ieșire k,
    $$\delta_k = (t_k - o_k)o_k(1 - o_k)$$
- Pentru neuronii din stratul ascuns
  - se calculează gradienții de eroare ai neuronilor din stratul ascuns
    Pentru unitatea ascunsă h, se însumează erorile δ_k pentru fiecare unitate de ieșire influențată de h, ponderate cu w_{kh} (ponderea de la stratul ascuns h la stratul de ieșire k):
    $$\delta_h = o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

Actualizarea ponderilor

Pentru fiecare exemplu de antrenare d: $w_{ji} = w_{ji} + \Delta w_{ji}, \Delta w_{ji} = -\eta \frac{\delta E_d}{\delta w_{ji}}$, unde E_d este eroarea pentru exemplul de antrenare d

$$E_d(\overrightarrow{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

- Ponderile unei unități de ieșire

$$\Delta w_{ji} = \eta \delta_j x_{ji}, \qquad \delta_j = (t_j - o_j) o_j (1 - o_j)$$

- Ponderile unui neuron ascuns

$$\Delta w_{ji} = \eta \delta_j x_{ji}, \qquad \delta_j = o_j (1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

Backpropagation

Pentru rețele *feed-forward* cu număr arbitrar de straturi,

$$\delta_r = o_r(1 - o_r) \sum_{s \in layer \;\; m+1} w_{sr}\delta_s$$

δ_r pentru unitatea r din stratul m este calculată din valorile δ de la următorul strat $m + 1$

Derivarea

- x_{ji} = the ith input to unit j
- w_{ji} = the weight associated with the ith input to unit j
- $net_j = \sum_i w_{ji} x_{ji}$ (the weighted sum of inputs for unit j)
- o_j = the output computed by unit j
- t_j = the target output for unit j
- σ = the sigmoid function
- $outputs$ = the set of units in the final layer of the network
- $Downstream(j)$ = the set of units whose immediate inputs include the output of unit j

Utilizăm regula de înlănțuire:

$$\begin{aligned} \frac{\delta E_d}{\delta w_{ji}} &= \frac{\delta E_d}{\delta net_j} \frac{\delta net_j}{\delta w_{ji}} \\ &= \frac{\delta E_d}{\delta net_j} x_{ji} \end{aligned} \tag{2}$$

Derivarea

Case 1: Training Rule for Output Unit Weights. Just as w_{ji} can influence the rest of the network only through net_j, net_j can influence the network only through o_j. Therefore, we can invoke the chain rule again to write

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j} \tag{4.23}$$

To begin, consider just the first term in Equation (4.23)

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$$

The derivatives $\frac{\partial}{\partial o_j}(t_k - o_k)^2$ will be zero for all output units k except when $k = j$. We therefore drop the summation over output units and simply set $k = j$.

$$\begin{aligned}\frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2 \\ &= \frac{1}{2}2(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j} \\ &= -(t_j - o_j) \qquad (4.24)\end{aligned}$$

Next consider the second term in Equation (4.23). Since $o_j = \sigma(net_j)$, the derivative $\frac{\partial o_j}{\partial net_j}$ is just the derivative of the sigmoid function, which we have already noted is equal to $\sigma(net_j)(1 - \sigma(net_j))$. Therefore,

$$\begin{aligned}\frac{\partial o_j}{\partial net_j} &= \frac{\partial \sigma(net_j)}{\partial net_j} \\ &= o_j(1 - o_j) \qquad (4.25)\end{aligned}$$

Substituting expressions (4.24) and (4.25) into (4.23), we obtain

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\; o_j(1 - o_j) \tag{4.26}$$

Derivarea

Case 2: Training Rule for Hidden Unit Weights. In the case where j is an internal, or hidden unit in the network, the derivation of the training rule for w_{ji} must take into account the indirect ways in which w_{ji} can influence the network outputs and hence E_d. For this reason, we will find it useful to refer to the set of all units immediately downstream of unit j in the network (i.e., all units whose direct inputs include the output of unit j). We denote this set of units by $Downstream(j)$. Notice that net_j can influence the network outputs (and therefore E_d) only through the units in $Downstream(j)$. Therefore, we can write

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \; w_{kj} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \; w_{kj} \; o_j(1-o_j)
\end{aligned} \tag{4.28}$$

Rearranging terms and using δ_j to denote $-\frac{\partial E_d}{\partial net_j}$, we have

$$\delta_j = o_j(1-o_j) \sum_{k \in Downstream(j)} \delta_k \; w_{kj}$$

and

$$\Delta w_{ji} = \eta \; \delta_j \; x_{ji}$$

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between −.05 and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in $training\_examples$, Do

    Propagate the input forward through the network:

    1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

    Propagate the errors backward through the network:

    2. For each network output unit k, calculate its error term δ_k

    $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

    3. For each hidden unit h, calculate its error term δ_h

    $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

    4. Update each network weight w_{ji}

    $$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

    where

    $$\Delta w_{ji} = \eta\, \delta_j\, x_{ji} \tag{T4.5}$$

Backpropagation

- Se iterează peste toate exemplele (vectori) de antrenare (o epocă)
- Antrenarea rețelei continuă până când eroarea medie pătratică ajunge sub un prag acceptabil sau până când se atinge un nr. maxim de epoci de antrenare

Exemplu: din *Artificial Intelligence. A Guide to Intelligent Systems*.

Backpropagation

- Convergența: algoritmul *Backpropagation* *converge către un minim local*
- Invățare incrementală vs. învățare pe lot (*batch learning*)
  - *batch learning*: ponderile se actualizează o singură dată, după prezentarea tuturor vectorilor din grup

    Avantaj: rezultatele antrenării nu mai depind de ordinea în care sunt prezentați vectorii de antrenare

Varianta cu "moment" a algoritmului *Backpropagation*

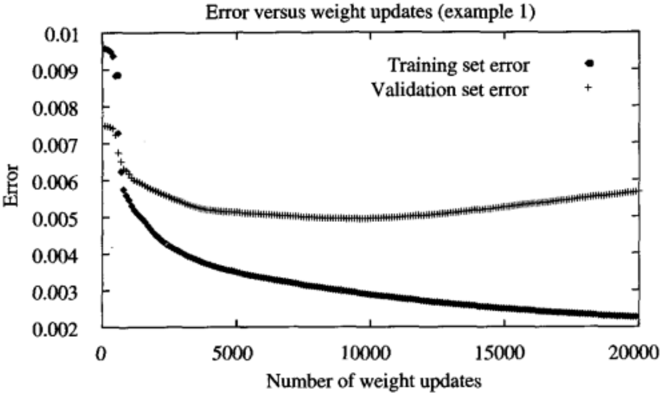
- Ajustarea ponderilor de la epoca curentă se calculează pe baza gradientului precum și a ajustărilor de la epoca anterioară

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1)$$

unde $\Delta w_{ji}(n)$ ajustarea ponderii la epoca n, $0 \leq \alpha < 1$ const. *momentum* (inerție)

Why Momentum Really Works, `https://distill.pub/2017/momentum/`

Overfitting



Soluții: weight decay (include o penalitate în funcția de eroare), k-fold cross-validation

Proiectarea rețelelor neuronale: Etape

- Arhitectură: număr de nivele și de unități pe fiecare nivel, topologie (mod de interconectare), funcții de activare
  Arhitecturi: unidirecționale vs. recurente
- Antrenare: determinarea valorilor ponderilor
- Validare: testarea modelului pe date de test

https://playground.tensorflow.org/

Bibliografie

- T. M. Mitchell, *Machine Learning*, Ch. 4 Artificial Neural Networks, McGraw-Hill Science, 1997
- S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, Ch. 18.7 Artificial Neural Networks, Prentice Hall, 1995
- M. Neqnevitsky. *Artificial Intelligence. A Guide to Intelligent Systems*, Ch. 6. Multilayer neural networks, 2005