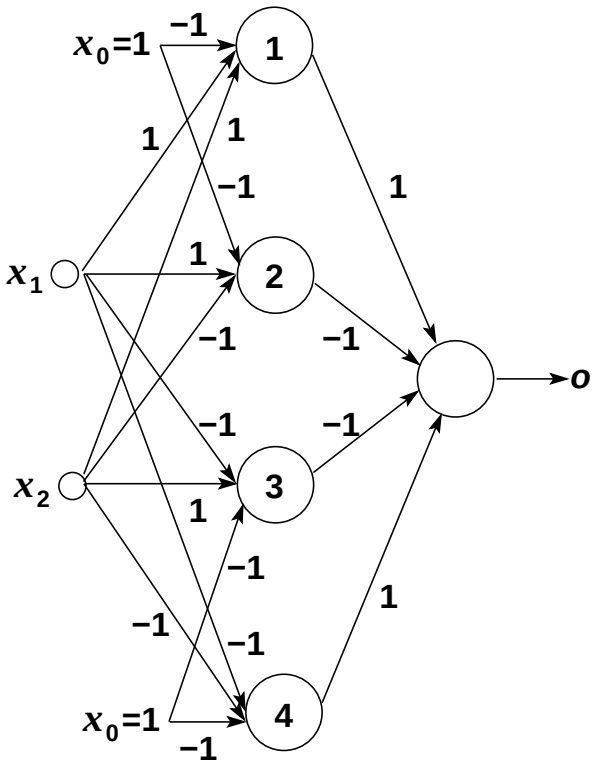
Artificial Neural Networks

(abbrev. ANNs, or simply NNs)

Calculation of the output of a neural network
made of threshold perceptrons

CMU, 2010 fall, Aarti Singh, HW5, pr. 4.1.1

Considerăm reţeaua neuronală din figura alăturată. Toate unităţile acestei reţele neuronale sunt de tip prag, adică folosesc pentru activare funcţia $sign$ **definită prin** $sign(z) = 1$ **dacă** $z \geq 0$ **şi** -1 **în rest. Pentru unitatea de pe nivelul de ieşire (lăsată nenumerotată), ponderea corespunzătoare termenului liber (**$x_0 = 1$**) este** 0**.**



a. Scrieţi funcţia matematică calculată de fiecare dintre unităţile reţelei, în raport cu intrările x_1 **şi** x_2**. Veţi nota cu** o_i **(unde** $i = 1, \ldots, 4$**) ieşirile unităţilor de pe nivelul ascuns şi cu** o **ieşirea reţelei, adică valoarea produsă de către unitatea de pe nivelul de ieşire.**

b. Calculaţi output-ul reţelei atunci când intrările x_1 **şi** x_2 **iau valori în mulţimea** $\{-1, 1\}$**.**

Răspuns

a.

$$
\begin{aligned}
o_1(x_1, x_2) &= sign(x_1 + x_2 - 1), \\
o_2(x_1, x_2) &= sign(x_1 - x_2 - 1), \\
o_3(x_1, x_2) &= sign(-x_1 + x_2 - 1), \\
o_4(x_1, x_2) &= sign(-x_1 - x_2 - 1), \\
o(x_1, x_2) &= sign(o_1(x_1, x_2) - o_2(x_1, x_2) - o_3(x_1, x_2) + o_4(x_1, x_2)).
\end{aligned}
$$

b.

x_1	x_2	o_1	o_2	o_3	o_4	o
1	1	1	−1	−1	−1	1
1	−1	−1	1	−1	−1	−1
−1	1	−1	−1	1	−1	−1
−1	−1	−1	−1	−1	1	1

c. Indicaţi funcţiile boolene reprezentate de către uităţile de pe nivelul ascuns (1, 2, 3 şi 4) atunci când $x_1, x_2 \in \{-1, 1\}$. Procedaţi similar pentru ieşirea o.

Răspuns:

Din tabelul obţinut la punctul precedent este imediat că, atunci când $x_1, x_2 \in \{-1, 1\}$, ieşirile calculate de către unităţile 1, 2, 3 şi 4 corespund conceptelor/funcţiilor $x_1 \wedge x_2$, $x_1 \wedge \neg x_2$, $\neg x_1 \wedge x_2$ şi respectiv $\neg x_1 \wedge \neg x_2$. Ieşirea reţelei, o, corespunde conceptului $\neg(x_1 \text{ XOR } x_2)$.

***Observaţie*: Se poate remarca faptul că în această reţea un neuron (şi doar unul!) de pe nivelul ascuns este activat la fiecare combinaţie de valori (din cele 4 posibile) pentru $x_1, x_2 \in \{-1, 1\}$.**

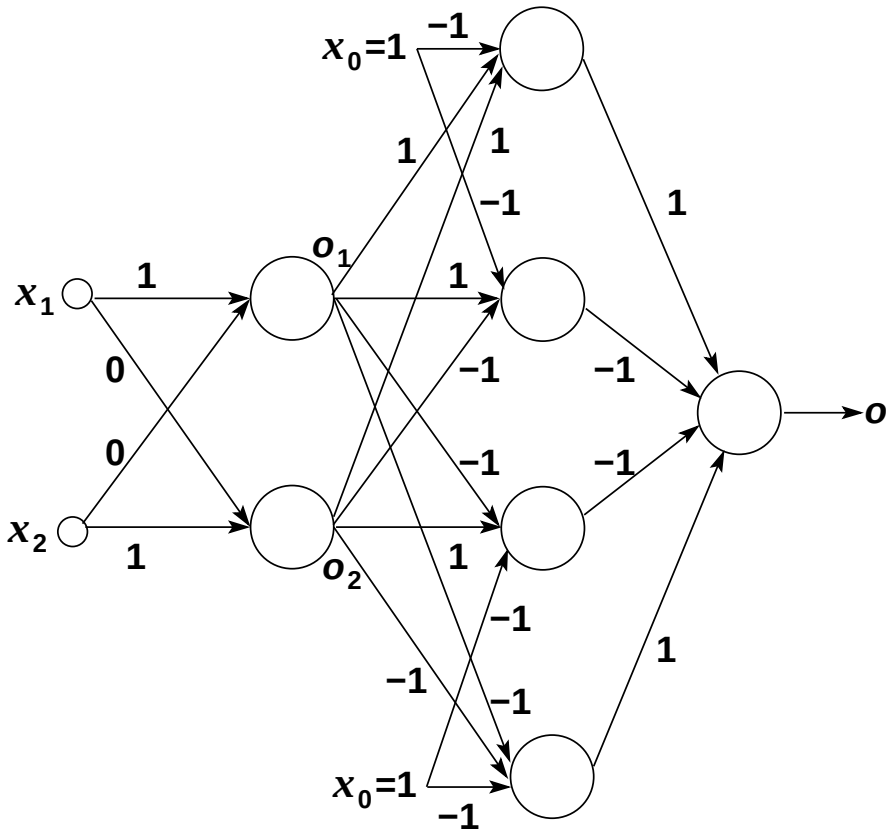
d. Specificaţi cum anume ar putea fi modificată reţeaua dată astfel încât noua variantă să calculeze funcţia de variabile reale (nu boolene ca mai înainte!) x_1 şi x_2, a cărei relaţie de definiţie este: $f(x_1, x_2) = 1$ dacă $x_1, x_2 \geq 0$ sau $x_1, x_2 < 0$, şi -1 în caz contrar.

Răspuns:

Este imediat că funcţia indicată în enunţ se poate scrie sub forma

$$f(x_1, x_2) = \begin{cases} 1 & \text{dacă } sign(x_1) \cdot sign(x_2) \geq 0 \\ -1 & \text{dacă } sign(x_1) \cdot sign(x_2) < 0. \end{cases}$$

Se observă imediat că aceasta coincide cu funcţia $\neg(sign(x_1)$ XOR $sign(x_2))$. Prin urmare, este suficient să adăugăm la reţeaua dată în enunţ un nivel ascuns suplimentar, format din două unităţi cu funcţie de activare de tip prag, care să transforme intrările x_1 şi x_2 în $sign(x_1)$ şi respectiv $sign(x_2)$. Vom obţine ca rezultat reţeaua din slide-ul următor.


$x_0=1$
−1
1
1
−1
1
o_1
x_1
1
1
0
−1
−1
0
−1
−1
x_2
1
1
o_2
−1
1
−1
−1
$x_0=1$
−1
o

Expressivity of neural networks:
boolean functions

CMU, 2010 fall, Aarti Singh, HW5, pr. 4.3

Consider the set of binary functions over d Boolean (binary) variables: $F = \{f : \{0,1\}^d \rightarrow \{0,1\}\}$. Show that any function in F can be represented by a neural network with a single hidden layer.

***Note*: You can work with values other than $\{0,1\}$, such as $\{1,-1\}$, as long as the function is still binary.**

Solution

Let S denote the set of all possible d-bit binary vectors. For any function $f \in F$, define $T_f = \{b \in \{0,1\}^d | f(b) = 1\}$. We construct a neural network with one hidden layer to represent f as follows:

The neural network has $|T_f|$ hidden units, each of which corresponds to a distinct binary vector in T_f. The weights from the input layer to the i-th hidden unit are determined by the values of the corresponding binary vector b_i: the j-th weight is 1 if b_{ij} is 1 and -1 otherwise. The activation functions in the hidden units are threshold functions:

$$h_i(v) = \begin{cases} 1 & \text{if } v \geq \sum_{j=1}^{d} b_{ij}, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that h_i outputs 1 if and only if the input is b_i. The weights from all hidden units to the output layer are 1, and the activation function in the output unit is the identity function.

To show that the above neural network represents f, we first note that for every b in T_f exactly one hidden unit, the one corresponding to b, outputs 1, and therefore the neural net outputs 1. For any $b \notin T_f$, all hidden units output 0, and the neural network outputs 0. This completes of the proof.

Exemplification:

CMU, 2011 spring, Roni Rosenfeld, HW4, pr. 1.c

A neural network with only one hidden layer that implements $(A \vee \neg B)$ XOR $(\neg C \vee D)$**.**

Solution (in Romanian)

Explicitând definiţia funcţiei logice XOR şi apoi aplicând regulile lui DeMorgan, expresia booleană din enunţ devine:

$$\begin{aligned}(A \vee \neg B) \text{ XOR } (\neg C \vee D) &= [(A \vee \neg B) \wedge \neg(\neg C \vee D)] \vee [\neg(A \vee \neg B) \wedge (\neg C \vee D)] \\ &= [(A \vee \neg B) \wedge (C \wedge \neg D)] \vee [(\neg A \wedge B) \wedge (\neg C \vee D)]\end{aligned}$$

Întrucât operatorii logici $\vee$ şi $\wedge$ sunt distributivi unul faţă de celălalt, rezultă că:

$$(A \vee \neg B) \text{ XOR } (\neg C \vee D) = (A \wedge C \wedge \neg D) \vee (\neg B \wedge C \wedge \neg D) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge D)$$

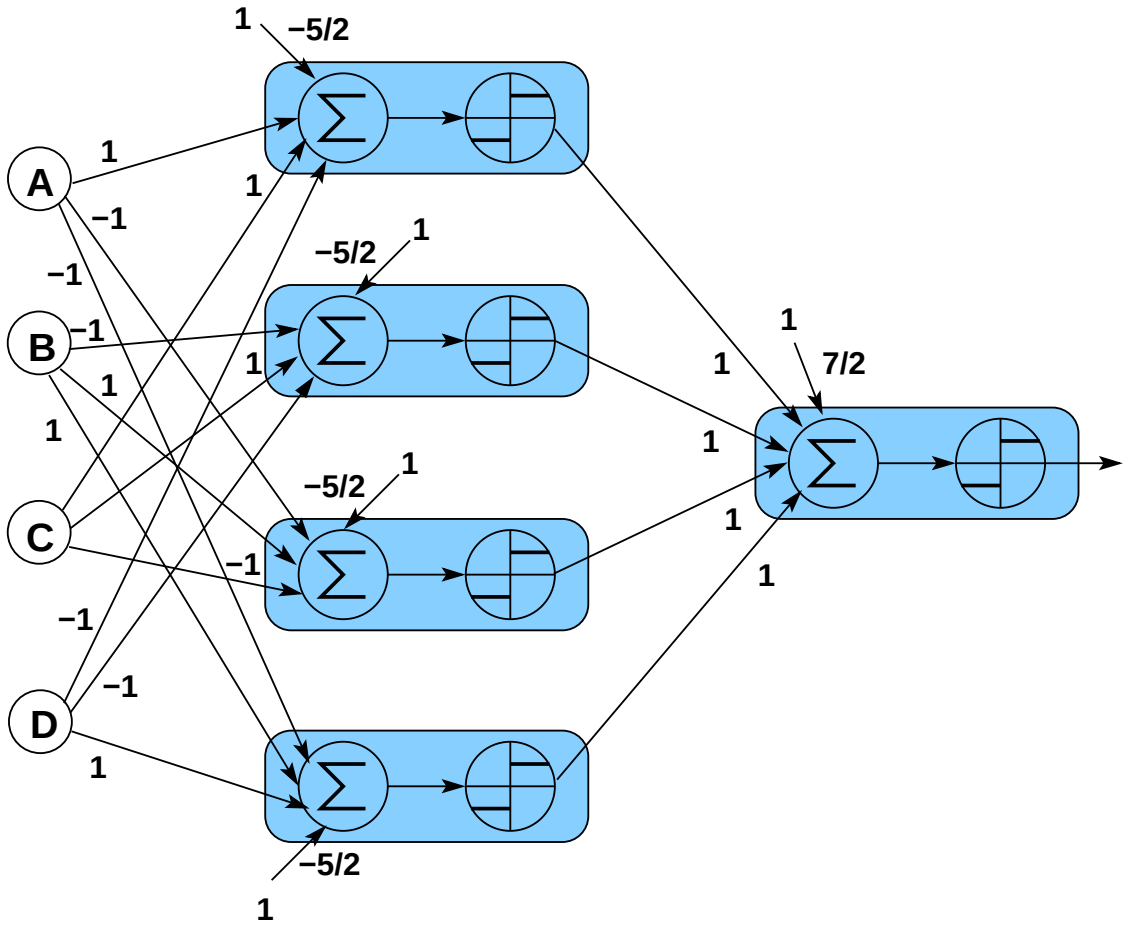
Fiecare din cele patru paranteze din partea dreaptă a egalităţii de mai sus poate fi reprezentată cu ajutorul unui perceptron-prag cu patru intrări, şi anume câte o intrare pentru fiecare din cele trei variabile boolene din paranteză, plus o intrare pentru termenul liber.

De *exemplu*, conjuncţiei $A \wedge C \wedge \neg D$ îi putem asocia inegalitatea $x + z - t > 5/2$, deci ponderile intrărilor perceptronului corespunzător vor fi $-5/2$, 1, 1 şi -1.)

Cei patru perceptroni vor fi plasaţi pe primul nivel ascuns al reţelei pe care o construim.

Apoi, ieşirile acestor patru perceptroni vor constitui intrări pentru un alt perceptron-prag, care să reprezinte disjuncţia a patru variabile boolene. (Acestui perceptron, situat pe nivelul de ieşire, îi putem asocia, de exemplu, inegalitatea $x + y + z + t > -7/2$**.)**

Reţeaua neuronală rezultată este cea din figura alăturată.



A *negative* expressivity result
concerning networks of threshold perceptrons

CMU, 2010 fall, Aarti Singh, HW5, pr. 4.1.2

Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined as $f(x_1, x_2) = 1$ for $x_1, x_2 \geq 0$ or $x_1, x_2 < 0$, and -1 otherwise. It was shwown (see CMU, 2010 fall, Aarti Singh, HW5, pr. 4.1.1) that it can be represented using a network of threshold perceptrons with two hidden layers.

Show that no neural network made of threshold units, with only one hidden layer, can represent f. For simplicity, you shall assume that the number of hidden units in a hidden layer is finite, but it can be arbitrarily large.

Hint

When there is only one hidden layer, the separating border corresponding to each hidden unit is a line in the 2-d plane such that the hidden unit outputs $+1$ on one side of the line and -1 on the other.

Consider a circular neighborhood (V) of the origin such that if a separating line determined by a hidden unit that crosses the neighborhood V, then it passes through the origin. (Since the number of hidden units is finite, we can always find such a neighborhood.)

Can any neural network made only of threshold units, with one hidden layer, represent f in the neighborhood V?

Solution

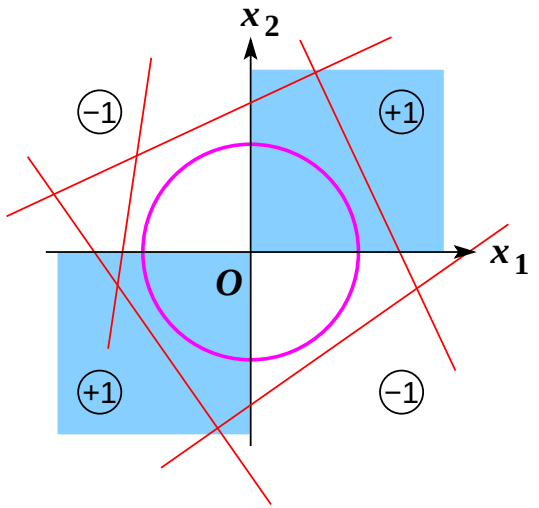
Given any neural network with one hidden layer of finite hidden units, we will consider a circular neighborhood of the origin

$$N_\varepsilon = \{(x_1, x_2) | \sqrt{x_1^2 + x_2^2} \leq \varepsilon\},$$

with ε being [less then] the minimum of the distances from the origin to the separating lines (hidden units) which do not pass through the origin.

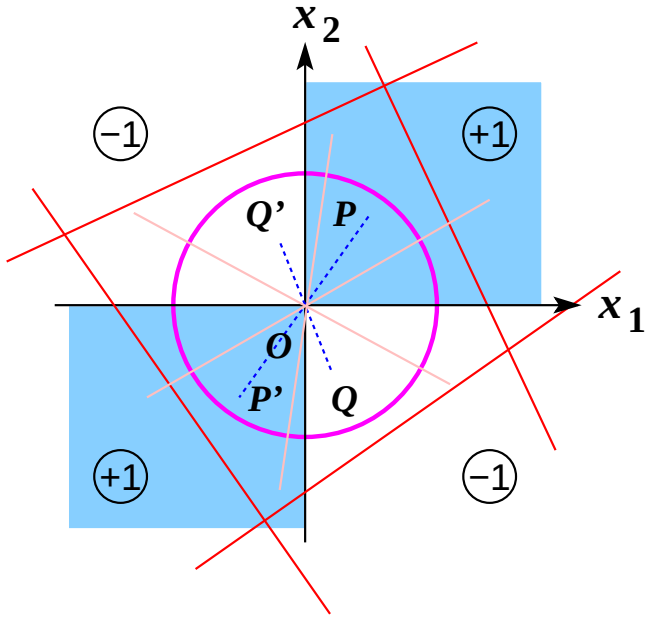
We will consider two cases:

Case 1: No line crosses the neighborhood. In this case, the neural network must output the same value in this neighborhood, but f outputs 1 or -1. Therefore, neural networks in this case cannot represent f.



Case 2

Some lines cross the neighborhood. By construction, these lines must all pass through the origin. Denote the set of the hidden units corresponding to these lines as O. Let us consider a point $P = (p_1, p_2)$ situated in this neighborhood and in the first quadrant, that does not fall on any line (see the figure). We can always find such a point because the number of the hidden units, hence the number of lines, is finite. By symmetry, the point $P' = (-p_1, -p_2)$, also in the neighborhood, does not fall on any line either, and $f(P) = f(P') = 1$. Let $y_h(P) \in \{1, -1\}$ denote the output of the hidden unit h on P.



It is easy to see the following:

$$y_h(P) = -y_h(P') \; \forall h \in O, \text{ and } y_h(P) = y_h(P') \; \forall h \notin O.$$

Let w_h denote the weight from the hidden unit h to the output unit. We thus have

$$\begin{aligned} \sum_h w_h y_h(P) &= \sum_{h \in O} w_h y_h(P) + \sum_{h \notin O} w_h y_h(P) = -\sum_{h \in O} w_h y_h(P') + \sum_{h \notin O} w_h y_h(P') \\ &= -\sum_h w_h y_h(P') + 2 \sum_{h \notin O} w_h y_h(P'). \end{aligned} \tag{1}$$

Similarly, we can pick Q and $Q' = -Q$ in the fourth and the second quadrantal parts of the neighborhood, and have that

$$\sum_h w_h y_h(Q) = -\sum_h w_h y_h(Q') + 2\sum_{h\notin O} w_h y_h(Q'). \tag{2}$$

Since $f(P) = f(P') = 1$ and $f(Q) = f(Q') = -1$, in order to represent f, the free weight w_0 (corresponding to $x_0 = 1$) for the output unit of the neural net must satisfy the inequalities

$$\sum_h w_h y_h(P) \geq -w_0 \ , \ \sum_h w_h y_h(P') \geq -w_0 \overset{(1)}{\Rightarrow} \sum_{h\notin O} w_h y_h(P') \geq -w_0,$$

$$\sum_h w_h y_h(Q) < -w_0 \ , \ \sum_h w_h y_h(Q') < -w_0 \overset{(2)}{\Rightarrow} \sum_{h\notin O} w_h y_h(Q') < -w_0,$$

which cannot happen because $\sum_{h\notin O} w_h y_h(P') = \sum_{h\notin O} w_h y_h(Q')$, since $y_h(P') = y_h(Q')$ for every $h \notin O$. This completes the proof.

Expressivity of neural networks:

Any Lipschitz continuous function defined on a bounded interval from $\mathbb{R}$ can be approximated by a neural network with a single hidden layer

CMU, 2011 fall, Tom Mitchell, Aarti Singh, HW5, pr. 2.3

Let $f(x)$ be any function whose domain is $[C, D)$, for real values $C < D$. Suppose that the function is Lipschitz continuous, that is,

$$\forall x, x' \in [C, D), |f(x') - f(x)| \leq L|x' - x|,$$

for some constant $L \geq 0$.

Construct a neural network with one hidden layer that approximates this function within $\varepsilon > 0$, that is, $\forall x \in [C, D), |f(x) - \boldsymbol{out}(x)| \leq \varepsilon$, where $out(x)$ is the output of your neural network given the input x.

***Hint*: You may use as building blocks the network(s) designed at CMU, 2007 fall, Carlos Guestrin, HW2, pr. 4.**

***Note*: Your network should use only the activation functions g_I and g_S (the identity and respectively the step function) mentioned at above referred problem.**

You need to specify the number K of hidden units, the activation function for each unit, and a formula for calculating each weight $w_0, w_k, w_0^{(k)}$, and $w_1^{(k)}$, for each $k \in \{1, 2, \ldots, K\}$. These weights may be specified in terms of C, D, L and ε, as well as the values of $f(x)$ evaluated at a finite number of x values of your choosing (you need to explicitly specify which x values you use). You do not need to explicitly write the $out(x)$ function.

Why does your network attain the given accuracy?

Răspuns

Este imediat că din ipoteza $|f(x)-f(x')| \leq L|x-x'|$**, în cazul în care** $|x-x'| \leq \frac{\varepsilon}{L}$**, rezultă**

$$|f(x) - f(x')| \leq \varepsilon \tag{3}$$

Aşadar, este de dorit să „acoperim" intervalul $[C, D)$ **cu intervale de lungime** $\frac{2\varepsilon}{L}$ **(sau mai mică):**

$$[C, C + \frac{2\varepsilon}{L}),\ [C + \frac{2\varepsilon}{L}, C + 2\frac{2\varepsilon}{L}),\ \ldots,\ [C + (K-1)\frac{2\varepsilon}{L}, D))$$

şi să luăm punctele x' **de forma** $\xi_i = (\alpha_i + \beta_i)/2$**, pentru cu** $i = 1, \ldots, K$**, unde**

$$K \overset{not.}{=} \left\lceil (D-C)\frac{L}{2\varepsilon} \right\rceil,$$

$$\alpha_1 = C,\ \beta_1 = \alpha_2 = C + \frac{2\varepsilon}{L}, \ldots, \beta_{K-1} = \alpha_K = C + (K-1)\frac{2\varepsilon}{L},\ \beta_K = D.$$

Aşadar, prin ξ_i **am notat mijlocul intervalului** i **de mai sus,** $[\alpha_i, \beta_i)$**.**

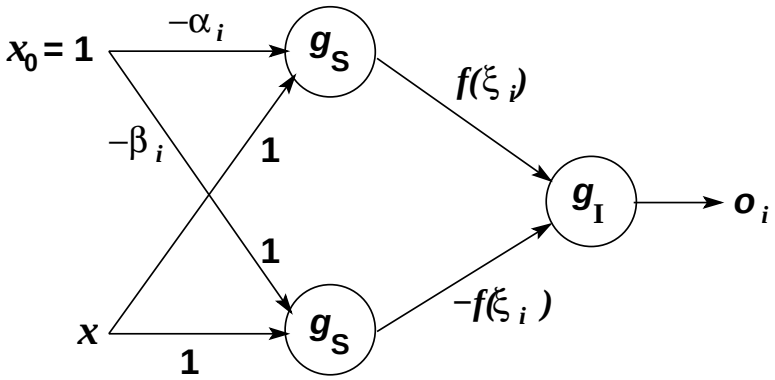
În continuare, vom arăta că putem construi o reţea neuronală cu un singur nivel ascuns şi care foloseşte doar unităţi de tip liniar sau de tip prag, astfel încât

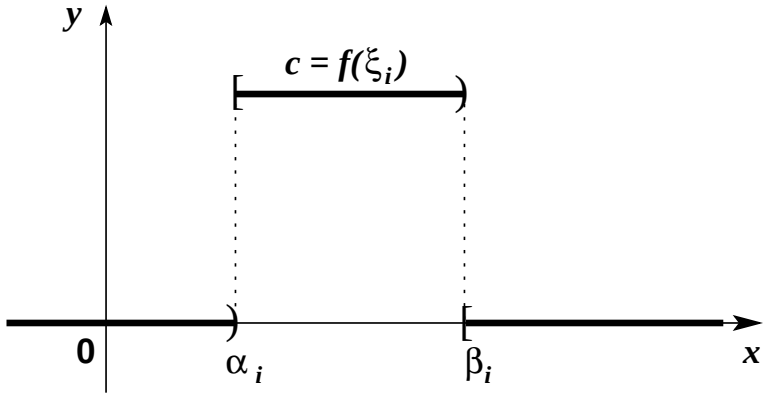
$$\textbf{pentru } \forall x \in [C, D), \textbf{ dacă } x \in [\alpha_i, \beta_i), \textbf{ atunci } \textbf{\textit{out}}(x) \stackrel{def.}{=} f(\xi_i). \tag{4}$$

În consecinţă, dacă în relaţia (3) vom înlocui x' cu ξ_i, va rezulta imediat că $|f(x) - \textbf{\textit{out}}(x)| \leq \varepsilon$.

Revenind acum la proprietatea (4), vom arăta că există o reţea având K unităţi pe nivelul ascuns (şi o singură unitate de ieşire), astfel încât, dacă $x \in [\alpha_i, \beta_i)$, atunci unitatea i de pe nivelul ascuns se activează (producând valoarea $f(\xi_i)$, iar toate celelalte unităţi de pe nivelul ascuns rămân neactivate (adică produc output 0). Prin urmare, rezultatul va fi exact cel dorit.

Conform problemei CMU, 2007 fall, Carlos Guestrin, HW2, pr. 4.b, există o reţea neuronală care produce valoarea $c = f(\xi_i)$ pentru orice input $x \in [\alpha_i, \beta_i)$ şi 0 în rest, şi care foloseşte funcţii de activare g_S pe nivelul ascuns şi g_I pe nivelul de ieşire:



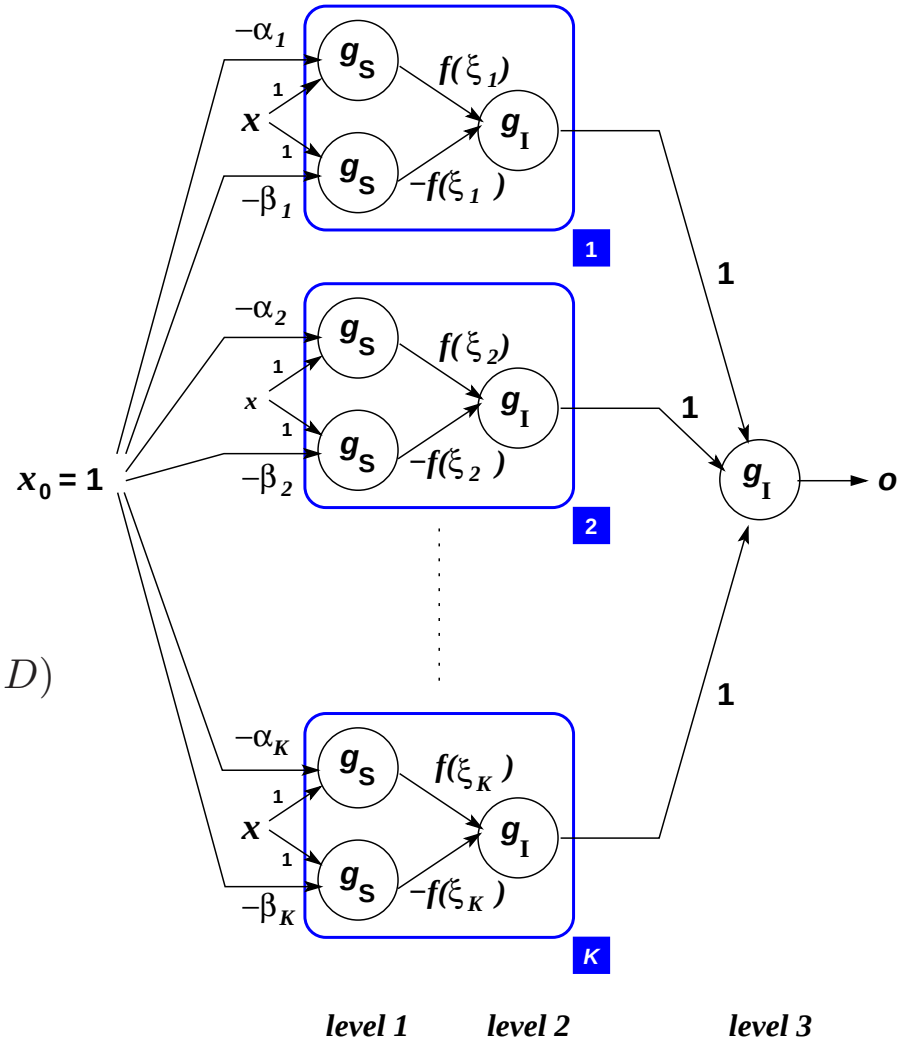


„Ansamblând“ K astfel de reţele, vom obţine reţeaua din figura alăturată.

Output-ul acestei reţele este exact cel dorit (deci satisface condiţia din enunţ):

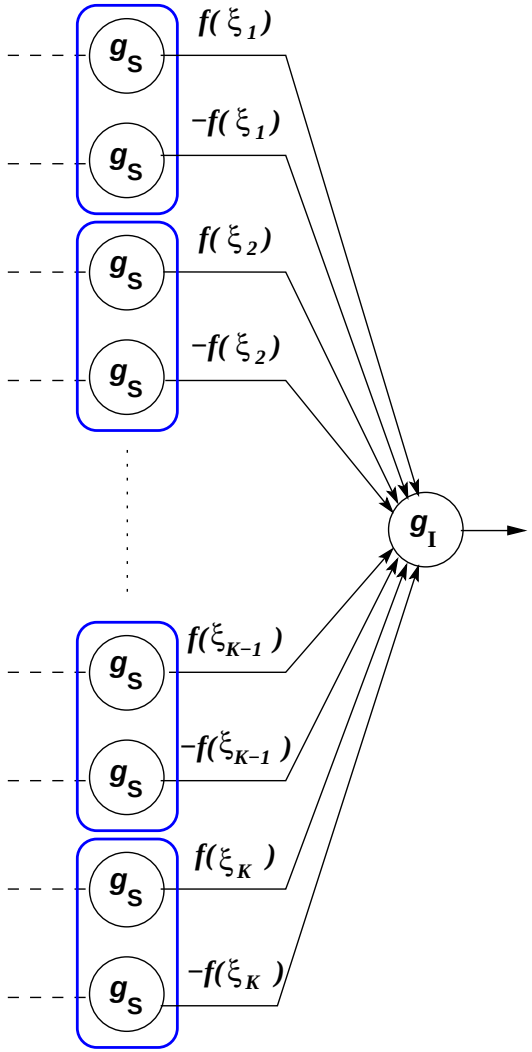
$$\textbf{\textit{out}}(x) = \begin{cases} f(\xi_1) & \textbf{pt.} \quad x \in [\alpha_1 = C, \beta_1) \\ f(\xi_2) & \textbf{pt.} \quad x \in [\alpha_2 = \beta_1, \beta_2) \\ \dots & \dots \quad \dots \\ f(\xi_K) & \textbf{pt.} \quad x \in [\alpha_K = \beta_{K-1}, \beta_K = D) \end{cases}$$

$$\Rightarrow |f(x) - \textbf{\textit{out}}(x)| \leq \varepsilon, \ \forall x \in [C, D).$$

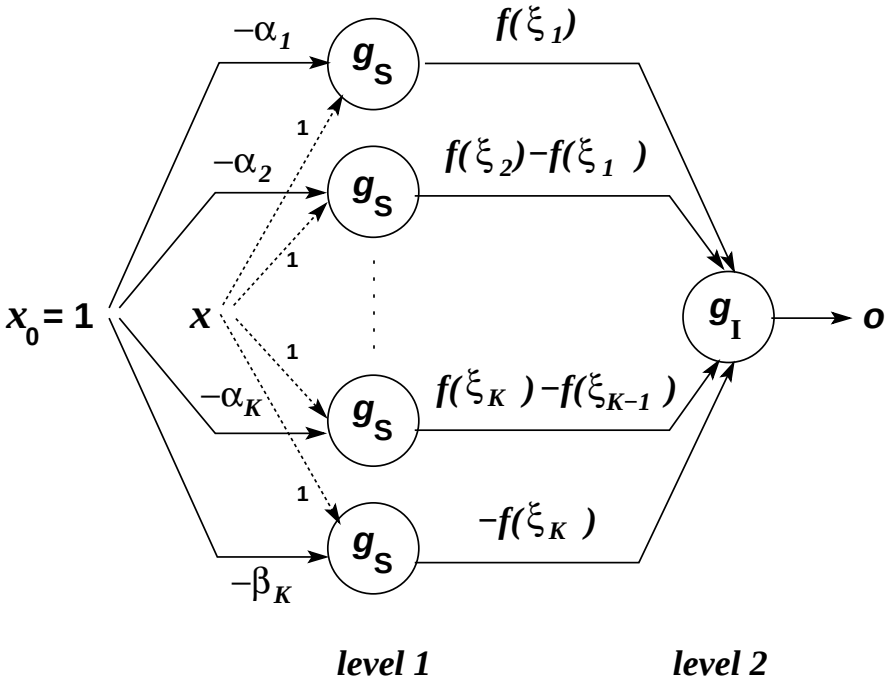


Neajunsul **este că reţeaua aceasta are două niveluri ascunse, în loc de unul singur, aşa cum se cere în enunţ.**

Totuşi, el poate fi ,,corectat“ imediat, comasând nivelurile 2 şi 3 — formate doar din unităţi liniare — într-o singură unitate liniară (care va constitui nivelul de ieşire al noii reţele), ca în figura alăturată.



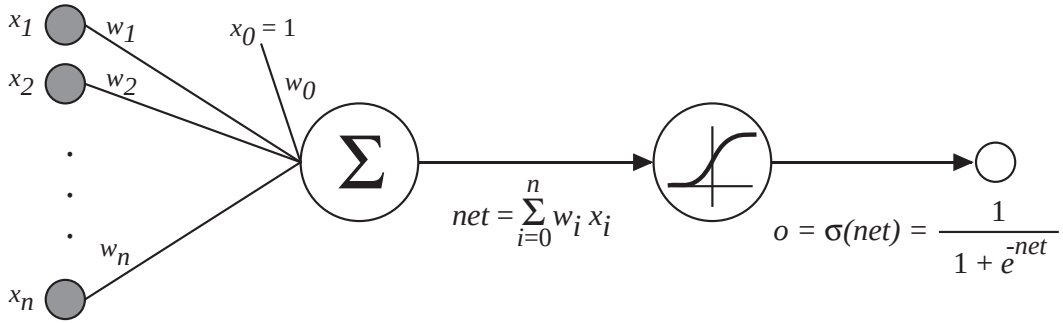
De asemenea, datorită faptului că $\beta_1 = \alpha_2, \beta_2 = \alpha_3, \ldots, \beta_{K-1} = \alpha_K$, nivelul ascuns (al acestei noi reţele), format din cele $2K$ unităţi de tip prag, poate fi echivalat cu un altul, compus din doar $K + 1$ unităţi de tip prag. La final, obţinem reţeaua din figura alăturată.



The sigmoidal / logistic perceptron:

Exemplification: setting up its weights so as to represent a given boolean expression

CMU, 2003 fall, T. Mitchell, A. Moore, midterm, pr. 6.a



Consider a single sigmoid threshold unit with three inputs, x_1, x_2, and x_3.

$$y = \sigma(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3) \textbf{ where } \sigma(z) = \frac{1}{1 + e^{-z}}.$$

We input values of either 0 or 1 for each of these inputs. Assign values to weights w_0, w_1, w_2 and w_3 so that the output of the sigmoid unit is greater than 0.5 if an only if $(x_1$ AND $x_2)$ OR x_3 is true.

Solution (in Romanian)

Vom scrie mai întâi tabela de valori a funcţiei / expresiei boolene date. Vom adăuga în această tabelă o coloană care va conţine forma [particulară a] sumei ponderate $w_0 + w_1x_1 + w_2x_2 + w_3x_3$ — adică, *argumentul* pe care-l va lua funcţia sigmoidală (σ) — pentru fiecare combinaţie de valori ale variabilelor x_1, x_2, x_3. În plus, la fiecare linie vom preciza şi *condiţia* care trebuie să fie satisfăcută de către suma ponderată de pe linia respectivă, în aşa fel încât perceptronul să realizeze codificarea cerută în enunţ.

x_1	x_2	x_3	$(x_1 \wedge x_2) \vee x_3$	$w_0 + w_1x_1 + w_2x_2 + w_3x_3$	$\vdots\ 0$
0	0	0	0	w_0	< 0
0	0	1	1	$w_0 + w_3$	> 0
0	1	0	0	$w_0 + w_2$	< 0
0	1	1	1	$w_0 + w_2 + w_3$	> 0
1	0	0	0	$w_0 + w_1$	< 0
1	0	1	1	$w_0 + w_1 + w_3$	> 0
1	1	0	1	$w_0 + w_1 + w_2$	> 0
1	1	1	1	$w_0 + w_1 + w_2 + w_3$	> 0

(5) **Aşadar, rezumând, pentru ca unitatea sigmoidală să codifice expresia booleană dată, trebuie ca ponderile w_i să satisfacă sistemul de inecuaţii scrise în ultima coloană a tabelului alăturat.**

În vederea găsirii unei soluţii pentru acest sistem, putem observa că în expresia $(x_1 \wedge x_2) \vee x_3$ variabilele logice x_1 şi x_2 joacă rol simetric în raport cu operatorul $\wedge$. Deci este natural să considerăm că ponderile alocate lui x_1 şi x_2 [ca input-uri] în perceptronul sigmoidal pot fi egale. Introducând deci restricţia $w_1 = w_2$, sistemul (5) va deveni cel scris în dreapta.

$$\begin{cases} w_0 < 0 \\ w_0 + w_3 > 0 \\ w_0 + w_1 < 0 \\ w_0 + w_1 + w_3 > 0 \\ w_0 + 2w_1 > 0 \\ w_0 + 2w_1 + w_3 > 0 \end{cases} \tag{6}$$

Mai departe, analizând din nou expresia $(x_1 \wedge x_2) \vee x_3$, este natural să gândim că, în raport cu operatorul $\vee$, ponderea variabilei x_3 ar trebui să fie aceeaşi cu ponderile ,,cumulate“ ale variabilelor x_1 şi x_2. Prin urmare, ,,injectând“ şi restricţia $w_3 = w_1 + w_2 = 2w_1$, sistemul de inecuaţii (6) va deveni cel scris în dreapta.

$$\begin{cases} w_0 < 0 \\ w_0 + 2w_1 > 0 \\ w_0 + w_1 < 0 \\ w_0 + 3w_1 > 0 \\ w_0 + 4w_1 > 0 \end{cases} \tag{7}$$

Din forma sistemului (7), apare natural să explorăm ce anume se întâmplă dacă restricţionăm spaţiul de soluţii inpunând condiţia $w_1 > 0$. În această ipoteză, sistemul (7) va avea aceeaşi soluţie ca şi inecuaţia dublă

$$w_0 + w_1 < 0 < w_0 + 2w_1,$$

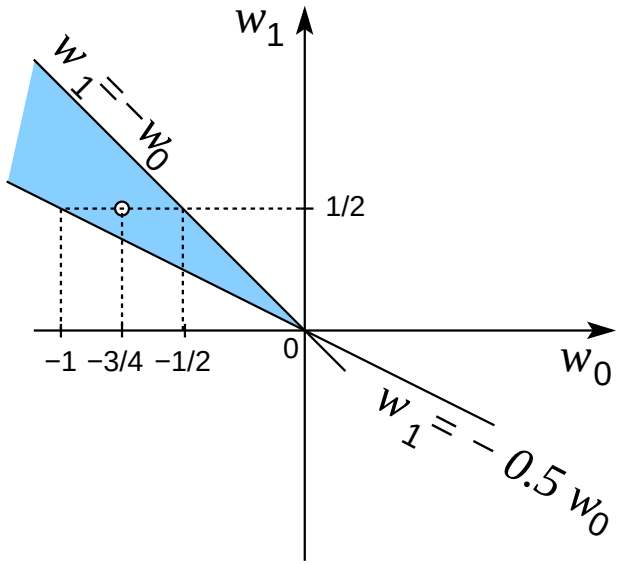
care este echivalentă cu

$$w_1 < -w_0 < 2w_1. \tag{8}$$

Pentru această ultimă inecuaţie dublă este foarte uşor să indicăm soluţii. Iată una dintre ele: $w_1 = \frac{1}{2}$ şi $w_0 = -\frac{3}{4}$. Prin urmare, $w_2 = \frac{1}{2}$ şi $w_3 = 1$.

Observaţie (1)

De fapt, dacă lucrăm într-un reper de coordonate w_0Ow_1, orice punct (w_0, w_1) din cadranul al doilea [şi] care este situat între dreptele de ecuaţii $w_1 = -w_0$ şi respectiv $w_1 = -\frac{1}{2}w_0$ este o soluţie a inecuaţiei duble (8), deci şi a sistemului de inecuaţii (5), la care, aşa cum am văzut, au fost adăugate restricţiile $w_1 = w_2 = \frac{1}{2}w_3$.



(Invers, adică a preciza care este întreg spaţiul de soluţii al sistemului (5) este mult în afara obiectivului acestui exerciţiu.)

Observaţie (2)

Rostul acestui [tip de] exerciţiu este să arate că pentru a găsi valori corespunzătoare ponderilor unui perceptron în aşa fel încât el să reprezinte/codifice o anumită funcţie, putem apela la [rezolvarea de] sisteme de restricţii / inecuaţii.

Mai general, aşa cum precizează şi Tom Mitchell în Machine Learning la pag. 95, putem folosi metode de programare liniară sau ne-liniară.

Pe de altă parte, acest exerciţiu pune în evidenţă în mod indirect faptul că ar fi de dorit ca (alternativ) să dispunem de o procedură de calcul generală, cât mai simplă, prin care să atingem obiectivul menţionat anterior.

O astfel de procedură — bazată pe metoda gradientului descendent — va face obiectul problemelor următoare. În raport cu programarea linininiară sau cea ne-liniară, această procedură are avantajul că se scalează în mod elegant/convenabil la nivel de reţea neuronală.

Rosenblatt's *Perceptron* Algorithm:

some simple properties

CMU, 2015 spring, Tom Mitchell, Nina Balcan, HW6, pr 3.ab

CMU, 2017 spring, Barnabas Poczos, HW3, pr. 1.bc

Consider running the *Perceptron* algorithm (see the nearby pseudo-code) on some sequence of examples S. (Remember that an example is a data point and its label.)

```
initialize w̄ ← 0̄
for i = 1, ..., n do
    if y_i (w̄ · x̄_i) ≤ 0 then
        w̄ ← w̄ + y_i x̄_i
    end if
end for
```

Let S' be the same set of examples as S, but presented in a different order.

a. Does the *Perceptron* algorithm necessarily make the same number of mistakes on S as it does on S'?
If so, why? If not, show such an S and S' where the *Perceptron* algorithm makes a different number of mistakes on S' than it does on S.

Solution (in Romanian)

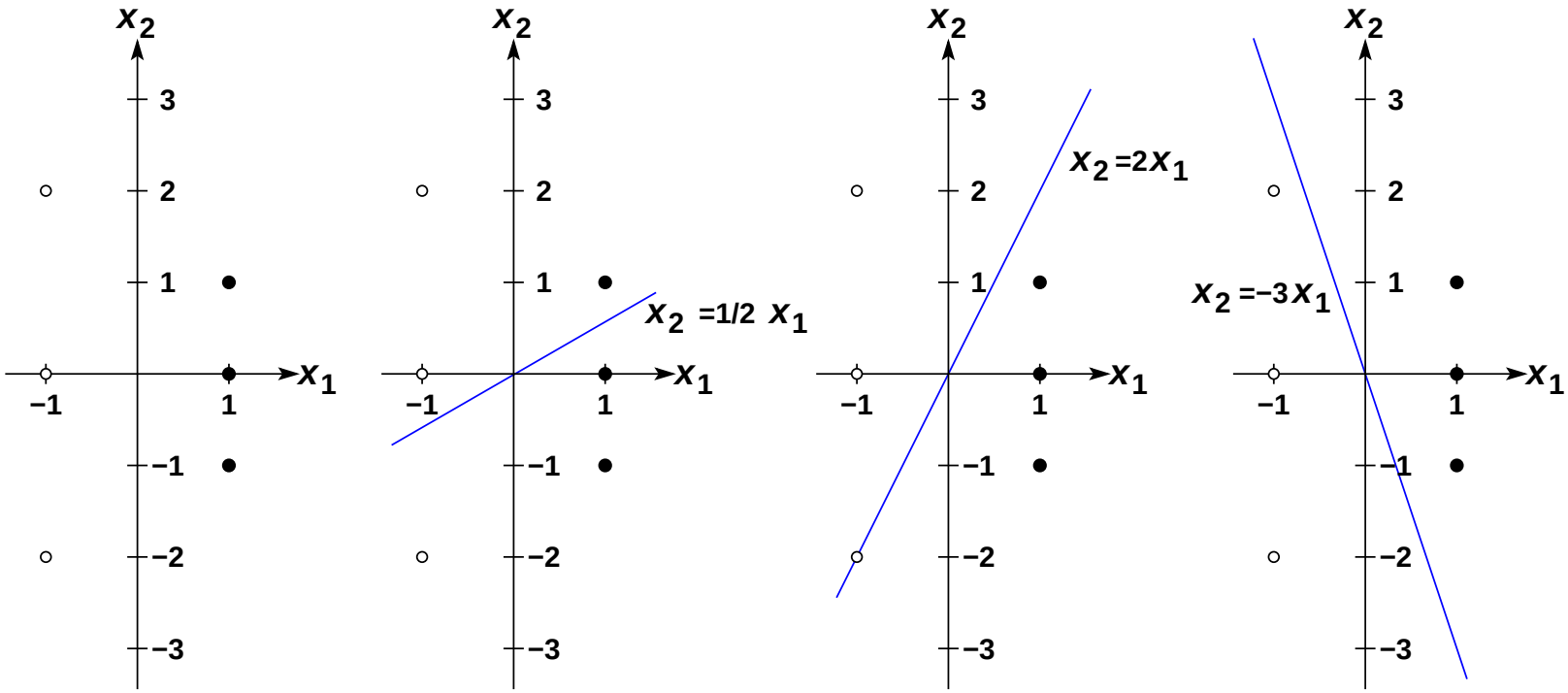
a. Considerăm următorul set de exemple (S), în ordinea indicată:

Exemplul	1	2	3	4	5	6
Instanţa (x_1, x_2)	$(-1, 2)$	$(1, 0)$	$(1, 1)$	$(-1, 0)$	$(-1, -2)$	$(1, -1)$
Eticheta y	-1	$+1$	$+1$	-1	-1	$+1$

Sintetizăm execuţia algoritmului *Perceptron* pe aceste date, fără a folosi termenul "bias" (w_0), alcătuind tabelul următor:

Iteraţia(i)	$\bar{x}_i$	y_i	$y_i \bar{w} \cdot \bar{x}_i$	$\bar{w}$	greşeală	ec. separatorului
iniţializare	–	–	$(0, 0)$	–	–	
1	$(-1, 2)$	-1	$0 \leq 0$	$(1, -2)$	da	$x_2 = 0.5x_1$
2	$(1, 0)$	$+1$	$1 > 0$	$(1, -2)$	nu	$x_2 = 0.5x_1$
3	$(1, 1)$	$+1$	$-1 \leq 0$	$(2, -1)$	da	$x_2 = 2x_1$
4	$(-1, 0)$	-1	$2 > 0$	$(2, -1)$	nu	$x_2 = 2x_1$
5	$(-1, -2)$	-1	$0 \leq 0$	$(3, 1)$	da	$x_2 = -3x_1$
6	$(1, -1)$	$+1$	$2 > 0$	$(3, 1)$	nu	$x_2 = -3x_1$

Aşadar, algoritmul a comis trei greşeli. Cei trei separatori obţinuţi în cursul aplicării algoritmului *Perceptron* pe şirul de exemple S sunt prezentaţi în figura de mai jos.



Este imediat că modul în care se construieşte separatorul — a cărui expresie analitică este o combinaţie liniară de instanţe greşit catalogate — este dependent de exemplele prezentate. Întrebarea care a fost pusă în enunţ este dacă pentru două succesiuni/ordini diferite, rezultatele finale (i.e., expresiile separatorilor obţinuţi) coincid (întotdeuna) sau nu. În continuare vom arăta că în mod obişnuit rezultatele finale (i.e., poziţiile finale ale separatorilor obţinuţi) *nu* coincid.

Considerăm S' secvenţa de exemple obţinută din S prin inversarea [ordinii] exemplelor $\bar{x}_1$ şi $\bar{x}_2$. Este imediat că la prima iteraţie perceptronul greşeşte şi apoi calculează $\bar{w} = (1, 0)$, ceea ce conduce la ecuaţia separatorului $x_1 = 0$, adică axa Ox_1. Evident, aceasta dreaptă este un separator perfect pentru întregul set de exemple, deci algoritmul nu va mai comite până la final nicio [altă] greşeală. Concluzia este că, pe un acelaşi set de exemple de antrenament, algoritmul *Perceptron* poate comite un număr diferit de greşeli (şi, de asemenea, un alt separator) dacă exemplele îi sunt prezentate în două seccesiuni diferite.

Observaţie importantă

Deşi în ambele cazuri de mai sus (S şi S') algoritmul *Perceptron* găseşte un separator liniar pentru datele de intrare, acest fapt nu este garantat în gazul general, chiar dacă datele sunt liniar separabile. Vezi de exemplu problema CMU, 2015 spring, Alex Smola, midterm, pr. 4. (Motivele sunt: rata de învăţare (implicită) prea mare şi/sau ne-parcurgerea setului de date decât o singură dată.)

b. The *Perceptron* algorithm classifies data points by their position relative to a hyper-plane. The weight vector, w, learned by Perceptron will be normal to this hyperplane? (True or False?)

Răspuns:

Cunoaştem din geometria analitică faptul că orice doi vectori $\bar{w}$ şi $\bar{x}$ pentru care are loc relaţia $\bar{w} \cdot \bar{x} = 0$ sunt ortogonali (adică, perpendiculari unul pe celălalt).

În cazul perceptronilor (de tip liniar, prag sau sigmoidal), ecuaţia separatorului este tocmai de forma $\bar{w} \cdot \bar{x} = 0$, unde $\bar{x}$ este un punct arbitrar de pe dreapta [sau, în general, hiper-planul] separator. Considerând $\bar{x}_1$ şi $\bar{x}_2$ două astfel de puncte, diferenţa lor, $\bar{x}_1 - \bar{x}_2$, va fi un vector care are direcţia dreptei-separator. Din relaţiile $\bar{w} \cdot \bar{x}_1 = 0$ şi $\bar{w} \cdot \bar{x}_2 = 0$ rezultă $\bar{w} \cdot (\bar{x}_1 - \bar{x}_2) = 0$. Prin urmare, vectorul de ponderi $\bar{w}$ este perpendicular pe direcţia dreptei-separator.

Proprietatea aceasta se poate verifica şi în mod direct/particular pe datele de la punctele precedente. De exemplu, la punctul a, pe şirul de exemple S, la iteraţia 1 avem $\bar{w} = (1, -2)$ şi se poate observa direct pe grafic că acest vector este perpendicular pe direcţia dreptei $y = \frac{1}{2}x$.

Aşadar, răspunsul este *Adevărat*.

Rosenblatt' *Perceptron* Algorithm:

Convergence / Mistake bounds

MIT, 2009 fall, Tommy Jaakkola, lecture notes 2

Presupunem că folosim perceptronul de tip prag în varianta Rosenblatt şi vrem să învăţăm un concept, folosind instanţele de antrenament $x_1, \ldots, x_n, \ldots \in \mathbb{R}^d$ împreună cu etichetele corespunzătoare $y_1, \ldots, y_n, \ldots \in \{-1, 1\}$.
Demonstraţi că în cazul în care sunt îndeplinite condiţiile *i-iv* de mai jos, algoritmul de actualizare a ponderilor perceptronului termină într-un număr finit de paşi. Formal, exprimăm acest fapt astfel: $\exists m \in \mathbb{N}$ astfel încât $y_t w^{(m)} \cdot x_t \geq 0$ pentru orice $t \in \{1, \ldots, n, \ldots\}$, unde $w^{(m)}$ este vectorul de ponderi obţinut de perceptron la iteraţia m.
Iată acum *condiţiile* menţionate mai sus:

i. **Instanţele $x_1, \ldots, x_n, \ldots$ sunt separabile liniar prin originea sistemului de coordonate, cu o margine finită $\gamma > 0$. Din punct de vedere formal, aceasta înseamnă că există $w^* \in \mathbb{R}^d$ astfel încât $y_t w^* \cdot x_t \geq \gamma$ pentru $t = 1, \ldots, n, \ldots$.**

ii. **Toate instanţele $x_1, \ldots, x_n, \ldots$ sunt conţinute într-o sferă din $\mathbb{R}^d$ cu centrul în origine, adică $\exists R > 0$ astfel încât $||x_t|| \stackrel{def.}{=} \sqrt{x_t \cdot x_t} \leq R$ pentru orice t.**

iii. **Învăţarea se face în manieră *incrementală*, folosind următoarea regulă de actualizare a ponderilor:**

$$w^{(k+1)} = w^{(k)} + y_{t_k} x_{t_k} \text{ pentru un } t_k \in \{1, \ldots, n, \ldots\} \text{ a.î. } y_{t_k} w^{(k)} \cdot x_{t_k} \leq 0, \tag{9}$$

ceea ce înseamnă că instanţa x_{t_k} este clasificată eronat de către perceptron la iteraţia k.

iv. **Startarea procesului de învăţare se face cu $w^{(0)} = 0 \in \mathbb{R}^d$.**

***Indicaţie*: Arătaţi că la fiecare iteraţie (k) a algoritmului, sunt satisfăcute următoarele proprietăţi:**

a. $w^* \cdot w^{(k)} \geq k\gamma$**;**

b. $||w^{(k)}||^2 \leq kR^2$**;**

c. $k \leq \left(\frac{||w^*||}{\gamma} R\right)^2$**.**

Ultima inegalitate indică [o margine superioară pentru] numărul maxim de iteraţii executate de către perceptron.

***Observaţia 1*: Notând cu θ_t unghiul format de vectorii x_t şi w^* în $\mathbb{R}^d$ şi ţinând cont de faptul că**

$$\boldsymbol{cos}\,\theta_t = \boldsymbol{cos}\,(x_t, w^*) = \frac{x_t \cdot w^*}{||x_t||\,||w^*||},$$

deci

$$y_t w^* \cdot x_t = y_t\,||w^*||\,||x_t||\,\boldsymbol{cos}\,\theta_t,$$

ceea ce, coroborat cu condiţia i, implică $y_t\,||x_t||\,\boldsymbol{cos}\,\theta_t\,||w^*|| \geq \gamma$, adică $y_t\,||x_t||\,\boldsymbol{cos}\,\theta_t \geq \frac{\gamma}{||w^*||}$. Din punct de vedere geometric, aceasta înseamnă că în spaţiul $\mathbb{R}^d$ distanţa de la orice instanţă x_t la vectorul w^* (care trece prin originea sistemului de coordonate) este mai mare sau egală cu $\frac{\gamma}{||w^*||}$.

***Observaţia 2*: Separatorul $w^{(k)}$ este o combinaţie liniară de instanţele x_i, întrucât regula de actualizare este $w^{(k+1)} = w^{(k)} + y_i x_i$. Observaţia aceasta este valabilă şi la antrenarea perceptronului-prag clasic, bazat pe regula delta.**

Solution (in Romanian)

Ideea de bază

Algoritmul de actualizare a ponderilor perceptronului determină schimbarea poziţiei separatorului la fiecare iteraţie (k) la care avem de a face cu un exemplu clasificat greşit.

Intuitiv, ne aşteptăm ca poziţia separatorului la iteraţia $k+1$ (adică hiperplanul de ecuaţie $w^{(k+1)} \cdot x = 0$) să se apropie de poziţia separatorului w^* care defineşte conceptul de învăţat.

În consecinţă, cosinusul unghiului dintre $w^{(k)}$ şi w^* ar trebui să crească de la o iteraţie la alta.

Pentru a dovedi/verifica în mod riguros aceasta, vom ţine cont că, prin definiţie,

$$\cos(w^{(k)}, w^*) = \frac{w^{(k)} \cdot w^*}{||w^{(k)}||\ ||w^*||}.$$

Mai întâi vom compara valorile produsului scalar de la numărătorul fracţiei de mai sus, la două iteraţii succesive. Folosind regula (9), putem scrie:

$$w^{(k+1)} \cdot w^* = (w^{(k)} + y_{t_k} x_{t_k}) \cdot w^* = w^{(k)} \cdot w^* + y_{t_k} x_{t_k} \cdot w^*.$$

Întrucât $y_{t_k} x_{t_k} \cdot w^* \geq \gamma$ **(vezi condiţia** i **din enunţ), rezultă că valoarea produsului scalar** $w^{(k)} \cdot w^*$ **creşte la fiecare iteraţie cu o cantitate cel puţin egală cu** γ**. Cum** $w^{(0)} = 0$ **conform restricţiei *iv*, este imediat că** $w^{(k)} \cdot w^* \geq k\gamma$ **la iteraţia** k**. Aşadar, numărătorul fracţiei care defineşte** $\cos(w^{(k)}, w^*)$ **creşte cel puţin liniar în raport cu** k**. Cu aceasta, tocmai am demonstrat relaţia** a**.**

A analiza evoluţia numitorului fracţiei care defineşte $\cos(w^{(k)}, w^*)$ revine la a compara $||w^{(k+1)}||$ cu $||w^{(k)}||$, întrucât $||w^*||$ este constant în raport cu k.

$$\begin{aligned}
||w^{(k+1)}||^2 &\overset{def.}{=} w^{(k+1)} \cdot w^{(k+1)} \overset{(9)}{=} (w^{(k)} + y_{t_k} x_{t_k}) \cdot (w^{(k)} + y_{t_k} x_{t_k}) \\
&= w^{(k)} \cdot w^{(k)} + 2 y_{t_k} w^{(k)} \cdot x_{t_k} + y_{t_k}^2 x_{t_k} \cdot x_{t_k} \\
&= ||w^{(k)}||^2 + 2 y_{t_k} w^{(k)} \cdot x_{t_k} + y_{t_k}^2 ||x_{t_k}||^2 \\
&= ||w^{(k)}||^2 + 2 y_{t_k} w^{(k)} \cdot x_{t_k} + ||x_{t_k}||^2 \\
&\leq ||w^{(k)}||^2 + R^2.
\end{aligned}$$

Inegalitatea de mai sus derivă din faptul că $y_{t_k} w^{(k)} \cdot x_{t_k} \leq 0$ (i.e., exemplul t_k este clasificat greşit la iteraţia k, conform condiţiei iii din enunţ) şi din ipoteza că orice instanţă de antrenament este conţinută în sfera de rază R şi având centrul în originea spaţiului $\mathbb{R}^d$, conform condiţiei ii. Cum $w^{(0)} = 0$, este imediat că $||w^{(k)}||^2 \leq k\,R^2$, deci $||w^{(k)}|| \leq \sqrt{k}\,R$ la iteraţia k. Cu aceasta, am demonstrat relaţia b.

Acum, combinând rezultatele a şi b, obţinem următoarea inegalitate:

$$\cos(w^{(k)}, w^*) = \frac{w^{(k)} \cdot w^*}{||w^{(k)}||\ ||w^*||} \geq \frac{k\gamma}{\sqrt{k}\, R\, ||w^*||} = \sqrt{k}\frac{\gamma}{R\, ||w^*||}.$$

Ştim că valoarea funcţiei $\cos$ **este întotdeauna cel mult egală cu** 1**. În consecinţă,**

$$1 \geq \cos(w^{(k)}, w^*) \geq \sqrt{k}\frac{\gamma}{R\, ||w^*||} \Rightarrow k \leq \left(R\frac{||w^*||}{\gamma}\right)^2,$$

deci am demonstrat relaţia c**.**

Aceasta înseamnă că în condiţiile specificate în enunţ, antrenarea perceptronului-prag se va face în cel mult $\left\lfloor \left(R\frac{||w^*||}{\gamma}\right)^2 \right\rfloor$ **iteraţii, unde perechea de simboluri** $\lfloor\ \rfloor$ **desemnează funcţia parte întreagă inferioară.**

***Observaţia 3*:** **Marginea superioară calculată mai sus este remarcabilă, întrucât ea nu depinde nici de instanţele** x_i **şi nici de dimensiunea** (d) **a spaţiului din care sunt selectate aceste instanţe.**

Observaţia 4

Restricţia referitoare la separabilitatea prin origine a instanţelor de antrenament (vezi condiţia *i*, prima parte) — şi anume, componenta „liberă" w_0 asociată separatorului w^* trebuie să fie 0 — nu este de fapt limitativă. Cazul general al separabilităţii liniare în $\mathbb{R}^d$, adică $y_t(w_0^* + w^* \cdot x_t) \geq 0$ pentru orice $t \in \{1, 2, \ldots, n \ldots\}$, poate fi redus la cazul particular al separabilităţii prin origine în $\mathbb{R}^{d+1}$ dacă pe de o parte se consideră $w' = (w^0, w^1, \ldots, w^d)$, cu $w^* = (w^1, \ldots, w^d)$, iar pe de altă parte se mapează toate instanţele de antrenament $x_t = (x_t^1, \ldots, x_t^d)$ din $\mathbb{R}^d$ în $\mathbb{R}^{d+1}$ astfel: $x'_k = (x_t^0, x_t^1, \ldots, x_t^d)$, cu $x_t^0 = 1$ pentru orice t. Cu această mapare, rezultă $y_t w' \cdot x'_t \geq 0$ pentru orice t (respectiv $y_t w' \cdot x'_t \geq \gamma$ când se lucrează cu margine finită, ca în enunţul acestei probleme).

Nici restricţia *iv* ($w^{(0)} = 0$) nu este cu adevărat limitativă; în general algoritmii de antrenare a unităţilor / reţelelor neuronale iniţializează ponderile w la valori mici (în modul).

Similar, este imediat că restricţia potrivit căreia sfera în care sunt conţinute instanţele $x_1, \ldots, x_n, \ldots$ trebuie să aibă centrul în originea lui $\mathbb{R}^d$ (vezi condiţia *ii*, partea a doua) poate fi eliminată fără ca rezultatul de convergenţă să fie afectat.

Observaţia 5

Deducerea marginii superioare de la punctul c de mai sus în cazul în care se foloseşte o rată de învăţare oarecare $\eta > 0$ se face extinzând în mod natural demonstraţia de mai sus (vezi CMU, 2013 spring, A. Smola, B. Poczos, HW2, pr. 2.a).

În concluzie, mai rămân de examinat două condiţii: separabilitatea liniară cu margine $\gamma > 0$ şi conţinerea instanţelor de antrenament într-o sferă de rază finită. Se poate demonstra că ambele condiţii sunt esenţiale pentru convergenţa perceptronului Rosenblatt în regim de învăţare online (vezi problema CMU, 2008 fall, Eric Xing, midterm exam, pr. 3.2).

Exemplifying

The gradient descent method:

finding the minimum of a real function of second degree

University of Utah, 2008 fall, Hal Daumé III, HW4, pr. 1

Suppose we are trying to find the minimum of the function $f(x) = 3x^2 - 2x + 1$, for uni-variate (scalar) x.

First, verify that this function is convex (and therefore it has a *global* minimum).

Second, find the minimum of this function using calculus.

Finally, perform (by hand – show your work) three steps of gradient descent with $\eta = 0.1$ and the initial point $x_0 = 1$. How close does it get to the true solution?

Solution (in Romanian)

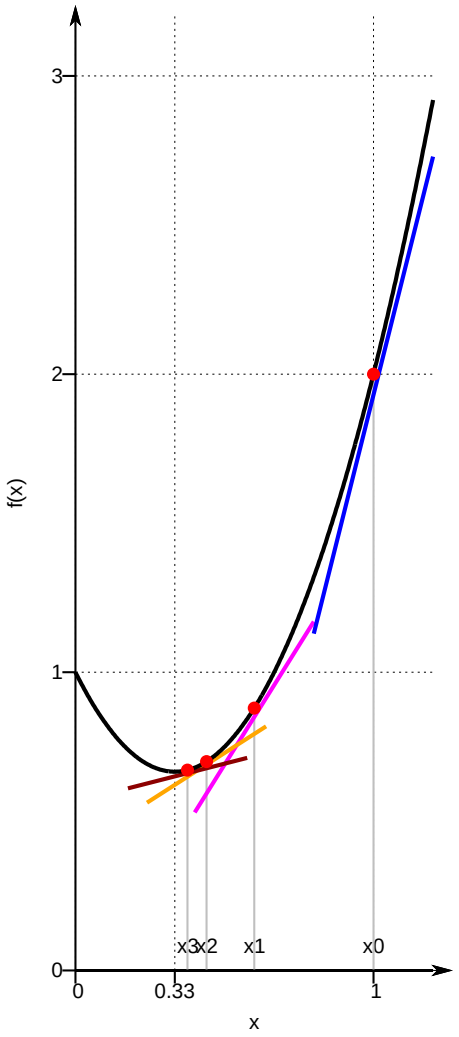
Pentru a studia convexitatea funcţiei $f(x)$ se calculează derivata a doua:

$$f''(x) = 6 > 0, \forall x \in \mathbb{R}.$$

Rezultă că funcţia f este convexă pe întreg domeniul ei de definiţie, deci are un (singur) punct de minim.
Pentru a calcula minimul funcţiei $f(x) = 3x^2 - 2x + 1$ utilizăm derivata de ordinul întâi:

$$f'(x) = 6x - 2.$$

Punctul de minim este dat de soluţia ecuaţiei $f'(x) = 0$, şi anume: $x = \frac{1}{3} \approx 0.33$.



Observaţii

Se constată uşor că

1. apropierea de punctul de optim este [mai] rapidă atâta timp cât valoarea primei derivate (i.e., panta tangentei la graficul funcţiei) este mare în valoare absolută;

2. dacă rata de învăţare η a fost fixată la o valoare prea mare, atunci este posibil ca la un moment dat să depăşim punctul de optim şi apoi să ,,pendulăm“ în jurul lui. Acest *punct slab* al metodei gradientului poate fi contracarat reducând în mod dinamic mărimea lui η.

Altminteri metoda gradientului are *avantajul* de a fi o *tehnică de optimizare* foarte simplă din punct de vedere conceptual şi uşor de implementat.

Alte două *puncte slabe* ale metodei gradientului descendent sunt:
imposibilitatea de a garanta găsirea optimul global şi
numărul mare de iteraţii care trebuie executate pe unele seturi de date reale.

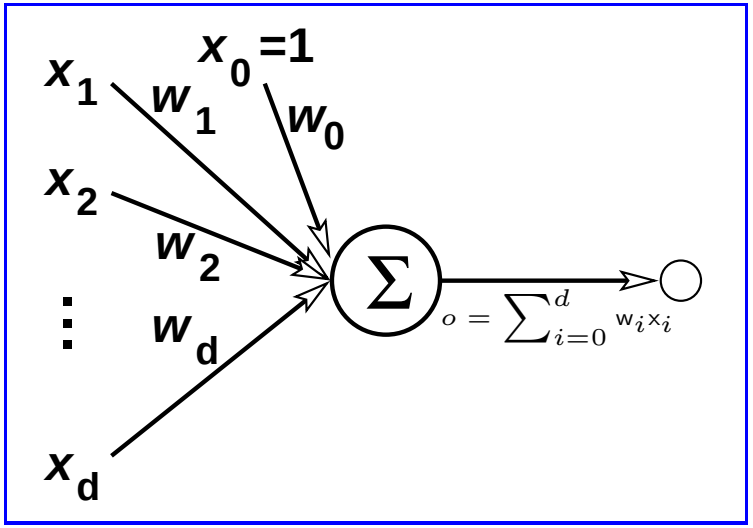
Theoretical foundations for the perceptron training algorithm

Liviu Ciortuz, 2017

following Tom Mitchell, *Machine Learning* book, 1997, p. 89-93

În acest exerciţiu vom elabora/reda [mai întâi] partea de fundamentare teoretică pentru antrenarea perceptronului liniar, date fiind instanţele $(\bar{x}_1, t_1), \ldots, (\bar{x}_n, t_n)$, cu $\bar{x}_i \in \mathbb{R}^d$ şi $t_i \in \mathbb{R}$ pentru $i = 1, \ldots, n$.

Aşadar, vom considera perceptronul liniar având intrările $x_0 = 1, x_1, \ldots, x_d$ şi ponderile $w_0, w_1, \ldots, w_d$.



a. În linie cu *metoda gradientului descendent*, deduceţi care este forma *regulii de actualizare* a vectorului de ponderi $\bar{w} \in \mathbb{R}^{d+1}$ (sau, echivalent, forma regulilor de actualizare a ponderilor w_i, cu $i = 0, \ldots, d$) pentru găsirea acelei valori care minimizează semi-suma pătratelor erorilor comise de perceptron,[a] adică

$$E(\bar{w}) \stackrel{def.}{=} \frac{1}{2} \sum_{i=1}^{n} (t_i - o_i)^2.$$

În această ultimă expresie, o_i este output-ul perceptronului liniar pentru intrarea $\bar{x}_i \stackrel{renot.}{=} (x_0 = 1, x_{i,1}, \ldots, x_{i,d})$, adică $o_i = w_0 + \sum_{j=1}^{d} w_j x_{i,j}$.

Vă reamintim că la aplicarea metodei gradientului descendent pentru găsirea unui punct de minim al unei funcţii derivabile $f : \mathbb{R}^{d+1} \to \mathbb{R}$, regula de actualizare a parametrilor $\bar{w}$ ai funcţiei f are forma $\bar{w} \leftarrow \bar{w} + \Delta\bar{w}$, cu $\Delta\bar{w} \stackrel{def.}{=} -\eta \nabla_{\bar{w}} f(\bar{w})$, unde $\eta > 0$ este un număr real mic, numit *rata de învăţare*, iar $\nabla_{\bar{w}} f(\bar{w})$ este vectorul *gradient*, adică $\left(\frac{\partial}{\partial w_0} f(\bar{w}), \frac{\partial}{\partial w_1} f(\bar{w}), \ldots \right)$.

---

[a] Conform problemei CMU, 2001 fall, T. Mitchell, A. Moore, final exam, pr. 10.2, am putea scrie — în ipoteza că datele au fost generate probabilist, cu o componentă „zgomot“ urmând o distribuţie gaussiană uni-variată de medie 0, aşa cum s-a precizat acolo —,

$$\bar{w}_{\mathrm{ML}} = \arg\min_{\bar{w}} E(\bar{w}) = \arg\min_{\bar{w}} \frac{1}{2} \sum_{i=1}^{n} (t_i - \bar{w} \cdot \bar{x}_i)^2.$$

Answer

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{j=1}^{n}(t_j - o_j)^2 = \frac{1}{2}\sum_{j=1}^{n}\frac{\partial}{\partial w_i}(t_j - o_j)^2 \\
&= \frac{1}{2}\sum_{j=1}^{n} 2(t_j - o_j)\frac{\partial}{\partial w_i}(t_j - o_j) = \sum_{j=1}^{n}(t_j - o_j)\frac{\partial}{\partial w_i}(t_j - \bar{w}\cdot\bar{x}_j) \\
&= \sum_{d}(t_j - o_j)(-x_{j,i})
\end{aligned}$$

Therefore,

$$\Delta w_i = \eta\sum_{j=1}^{n}(t_j - o_j)x_{j,i}$$

b. Dacă în locul perceptronului liniar vom considera perceptronul-prag, respectiv perceptronul sigmoidal, prin ce va diferi forma regulii de actualizare a ponderilor faţă de rezultatul de la punctul a? (Pentru perceptronul-prag, se va presupune că $t_i \in \{-1, +1\}$, în vreme ce pentru perceptronul sigmoidal vom considera $t_i \in \{0, 1\}$, pentru $i = 1, \ldots, n$.)

Answer:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{j=1}^{n}(t_j - o_j)^2 = \frac{1}{2}\sum_{j=1}^{n}\frac{\partial}{\partial w_i}(t_j - o_j)^2 = \frac{1}{2}\sum_{j=1}^{n}2(t_j - o_j)\frac{\partial}{\partial w_i}(t_j - o_j) \\
&= \sum_{j=1}^{n}(t_j - o_j)\frac{\partial}{\partial w_i}(t_j - \underbrace{\sigma(\bar{w}\cdot\bar{x}_j)}_{o_j}) = -\sum_{j=1}^{n}(t_j - o_j)\frac{\partial}{\partial w_i}\sigma(\bar{w}\cdot\bar{x}_j) \\
&= -\sum_{j=1}^{n}(t_j - o_j)\sigma(\bar{w}\cdot\bar{x}_j)(1 - \sigma(\bar{w}\cdot\bar{x}_j))\frac{\partial}{\partial w_i}\bar{w}\cdot\bar{x}_j = -\sum_{j=1}^{n}(t_j - o_j)o_j(1 - o_j)\frac{\partial}{\partial w_i}\bar{w}\cdot\bar{x}_j \\
&= -\sum_{j=1}^{n}(t_j - o_j)o_j(1 - o_j)x_{j,i}
\end{aligned}$$

***Note*: Here above we used the fact that**

$$\sigma'(z) = \left(\frac{1}{1+e^{-z}}\right)' = -\frac{(1+e^{-z})'}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2} = \sigma(z)(1-\sigma(z)), \forall z \in \mathbb{R}$$

c. Elaboraţi algoritmul de antrenare a perceptronului liniar (şi apoi, dacă doriţi, şi a celui sigmoidal, dar veţi preciza doar diferenţele!). La *iniţializare*, ponderile w_i vor primi ca valori numere reale mici. În ce priveşte *condiţia de oprire* a algoritmului, se poate opta pentru una din următoarele variante (eventual combinate):

– toate instanţele de antrenament sunt corect clasificate (în ipoteza că datele $\bar{x}_i$, cu $i = 1, \ldots, n$, sunt liniar separabile);

– efectuarea unui număr prestabilit de iteraţii;

– verificarea condiţiei $|E(\bar{w}^{(t+1)}) - E(\bar{w}^{(t)})| < \varepsilon$, unde pragul numeric $\varepsilon > 0$ este stabilit iniţial, iar $E(\bar{w}^{(t)})$ este valoarea *funcţiei de eroare* E (i.e., ceea ce mai sus am numit semi-suma pătratelor erorilor) pe setul de date de antrenament, la finalul iteraţiei t.

The gradient descent algorithm for the linear unit

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

where each *training example* is a pair of the form $\langle \bar{x}, t \rangle$, with

$\bar{x}$ **– the vector of input values**
t **– 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**

  initialize each Δw_i to zero;
  for each $\langle \bar{x}, t \rangle$ in $training\_examples$,

    input the instance $\bar{x}$ to the unit and compute the output o;
    for each linear unit weight w_i,

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i;$$

  for each linear unit weight w_i,

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

d. Ce cunoaşteţi (de la curs) despre convergenţa algoritmului de la punctul c? Dar despre convergenţa perceptronului-prag?

Answer:

[Hertz et al., 1991] has proven that the the gradient descent-based training algorithm [centered on the weight updating rule] for the linear unit is guaranteed to converge to a hypothesis that minimizes the sum of squared errors

- **given a sufficiently small learning rate η,**
- **even when the training data contains noise,**
- **even when the training data is not separable by H.**

Prior to that, it was proven by [Minsky & Papert, 1969] that [the same algorithm, applied to] the threshold perceptron converges

- **if the training data is linearly separable,**
- **and η is sufficiently small.**

Using another *loss-function*

in conjunction with the linear perceptron

and the gradient descent method:

The log-sigmoidal function;

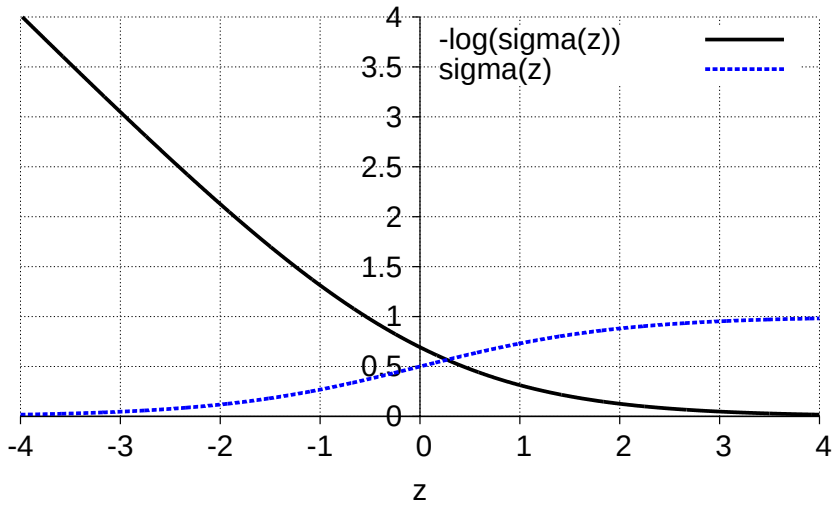
deduction of the *update rules*

University of Utah, 2008 fall, Hal Daumé III, HW4, pr. 2-4

The 0/1 loss function is hard to optimize using gradient descent because it is poorly behaved (discontinuous, non-differentiable, etc.). A common "smooth" version of 0/1 loss is the *sigmoid* loss, which makes use of our favorite function, the sigmoid: $\sigma(z) = 1/(1+\exp(-z))$.

In this exercise we will work with another loss function, the log-sigmoid:

$$\ell(y, f(x)) = -\log(\sigma(yf(x))).$$



a. First, verify that this is a reasonable loss function: when $f(x)$ is correct, the loss is low, and when $f(x)$ is incorrect, the loss is high. You may assume that $f(x)$ produces *real values*, where positive values mean class $+1$ and negative values mean class -1.

b. Consider optimizing a linear function under log-sigmoid loss, so that we have $f(\bar{w}) = \sum_n -\log(\sigma(y_n(\bar{w} \cdot \bar{x}_n + b)))$. Compute the gradient of this function with respect to $\bar{w}$ and with respect to b so that we might construct a gradient descent algorithm.

c. Show that the loss function from the previous two questions is convex in both $\bar{w}$ and b.

d. Finally, state the *update rules* for both w_i and b which serve for training a linear perceptron, based on the gradient descent method in conjunction with the log-sigmoid loss function.

Solution (in Romanian)

a. Funcţia l se poate scrie sub forma:

$$l(y, h(x)) = -\ln \sigma(y\, h(x)) = -\ln \frac{1}{1+e^{-yh(x)}} = \ln(1+e^{-yh(x)})$$

Din proprietăţile logaritmului ştim că $\ln 1 = 0$, $\ln a < 0$ dacă $0 < a < 1$, şi $\ln a > 0$ dacă $a > 1$.

Dacă ipoteza $h(x)$ este corectă, adică are acelaşi semn cu y, atunci $e^{-y\, h(x)}$ este o valoare mică (şi cu atât mai mică cu cât $|h(x)|$ este mai mare), deci şi valoarea funcţiei $l(y, h(x))$ este mică.

Dacă ipoteza $h(x)$ este incorectă, adică are semn contrar semnului lui y, atunci $e^{-y\, h(x)}$ este o valoare mare (şi cu atât mai mare cu cât $|h(x)|$ este mai mare), deci şi valoarea funcţiei $l(y, h(x))$ este mare.

De *exemplu*, dacă $y = 1$ şi $h(x) = -1$ (sau $y = -1$ şi $h(x) = 1$) atunci pierderea/costul este $-\ln \frac{1}{1+e} \approx 1.31$, iar dacă $y = h(x) = 1$ (sau, ambele, -1) atunci costul este $-\ln \frac{1}{1+e^{-1}} = -\ln \frac{e}{1+e} \approx 0.31$.

b. Calculăm derivatele parţiale ale funcţiei $f(b, \bar{w}) = \sum_n -\ln \sigma(y_n(\bar{w} \cdot \bar{x}_n + b))$ în raport cu w_i şi respectiv b. Vom folosi *proprietatea* $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

$$\begin{aligned}
\frac{\partial f}{\partial w_i} &= \frac{\partial}{\partial w_i} \sum_n -\ln \sigma(y_n(\bar{w} \cdot \bar{x}_n + b)) = -\sum_n \frac{\partial}{\partial w_i} \ln \sigma(y_n(\bar{w} \cdot \bar{x}_n + b)) \\
&= -\sum_n \frac{1}{\sigma(y_n(\bar{w} \cdot \bar{x}_n + b))} \frac{\partial}{\partial w_i} \sigma(y_n(\bar{w} \cdot \bar{x}_n + b)) \\
&= -\sum_n \frac{1}{\sigma(y_n(\bar{w} \cdot \bar{x}_n + b))} \sigma(y_n(\bar{w} \cdot \bar{x}_n + b))(1 - \sigma(y_n(\bar{w} \cdot \bar{x}_n + b)))y_n x_{n,i} \\
&= -\sum_n (1 - \sigma(y_n(\bar{w} \cdot \bar{x}_n + b)))y_n x_{n,i}.
\end{aligned}$$

Similar, vom obţine $\frac{\partial f}{\partial b} = -\sum_n (1 - \sigma(y_n(\bar{w} \cdot \bar{x}_n + b)))y_n$.

c. Funcţia $f(b, \bar{w})$ este convexă în raport cu variabila w_i dacă $\frac{\partial^2 f}{\partial w_i^2} \geq 0$. Similar, $f(b, \bar{w})$ este convexă în raport cu b dacă $\frac{\partial^2 f}{\partial b^2} \geq 0$. Aşadar, trebuie să calculăm aceste derivate parţiale de ordinul 2. Pentru aceasta, vom folosi derivatele parţiale de ordinul întâi care au fost calculate la punctul precedent.

$$\begin{aligned}
\frac{\partial^2 f}{\partial w_i^2} &= \frac{\partial}{\partial w_i}\left(\frac{\partial f}{\partial w_i}\right) = \frac{\partial}{\partial w_i}\left(-\sum_n (1-\sigma(y_n(\bar{w}\cdot\bar{x}_n+b)))\ y_n x_{n,i}\right) \\
&= \sum_n y_n x_{n,i}\ \sigma(y_n(\bar{w}\cdot\bar{x}_n+b))\ (1-\sigma(y_n(\bar{w}\cdot\bar{x}_n+b)))\ \frac{\partial}{\partial w_i}(y_n(\bar{w}\cdot\bar{x}_n+b))) \\
&= \sum_n y_n x_{n,i}\ \sigma(y_n(\bar{w}\cdot\bar{x}_n+b))\ (1-\sigma(y_n(\bar{w}\cdot\bar{x}_n+b)))\ y_n x_{n,i} \\
&= \sum_n \sigma(y_n(\bar{w}\cdot\bar{x}_n+b))\ (1-\sigma(y_n(\bar{w}\cdot\bar{x}_n+b)))\ y_n^2 x_{n,i}^2
\end{aligned}$$

Cum $\sigma(x) \in (0,1)$ pentru orice $x \in \mathbb{R}$, rezultă că şi $1-\sigma(x) \in (0,1)\,\forall x$. Aşadar, $\sigma(x) > 0$, $1-\sigma(x) > 0$ şi, evident, $y_n^2 \geq 0$, deci $\frac{\partial^2 f}{\partial b^2} \geq 0$, adică f este convexă în w_i, pentru orice i.

Analog, calculăm derivata parţială de ordin secund a lui f în raport cu b:

$$\begin{aligned}
\frac{\partial^2 f}{\partial b^2} &= \frac{\partial}{\partial b}\left(\frac{\partial f}{\partial b}\right) = \frac{\partial}{\partial b}\left(-\sum_n (1-\sigma(y_n(\bar{w}\cdot\bar{x}_n+b)))\ y_n\right) \\
&= \sum_n y_n \frac{\partial}{\partial b}\sigma(y_n(\bar{w}\cdot\bar{x}_n+b)) \\
&= \sum_n y_n\ \sigma(y_n(\bar{w}\cdot\bar{x}_n+b))\ (1-\sigma(y_n(\bar{w}\cdot\bar{x}_n+b)))\ \frac{\partial}{\partial b}(y_n(\bar{w}\cdot\bar{x}_n+b))) \\
&= \sum_n y_n\ \sigma(y_n(\bar{w}\cdot\bar{x}_n+b))\ (1-\sigma(y_n(\bar{w}\cdot\bar{x}_n+b)))\ y_n \\
&= \sum_n \sigma(y_n(\bar{w}\cdot\bar{x}_n+b))\ (1-\sigma(y_n(\bar{w}\cdot\bar{x}_n+b)))\ y_n^2 \geq 0
\end{aligned}$$

Aşadar, f este convexă şi în raport cu variabila b.

***Observaţie*: La curs, b a fost asimilat cu w_0, iar $x_{n,0} = 1$ prin definiţie. Dacă s-ar fi procedat la fel şi aici, atunci nu am fi avut de calculat decât o derivată la punctul a şi una la punctul b.**

d. În *concluzie*, valoarea optimă a funcţiei f este un minim, iar algoritmul de învăţare automată bazat pe metoda gradientului descendent va găsi (la limită, eventual) acest minim. Antrenarea acestui perceptron se face similar cu antreanarea unităţii liniare care a fost prezentată la curs, folosind reguli de actualizare de forma:

$$w_i \leftarrow w_i + \Delta w_i \text{ şi } b \leftarrow w_i + \Delta b.$$

În cazul de faţă, Δw_i şi Δb sunt definite de expresiile

$$\Delta w_i = -\eta \frac{\partial f}{\partial w_i} = \eta \sum_n (1 - \sigma(y_n(\bar{w} \cdot \bar{x}_n + b)))y_n \bar{x}_{n,i}$$

şi respectiv

$$\Delta b = -\eta \frac{\partial f}{\partial b} = \eta \sum_n (1 - \sigma(y_n(\bar{w} \cdot \bar{x}_{n,i} + b)))y_n$$

Algoritmul de retro-propagare:

deducerea regulilor de actualizare a ponderilor

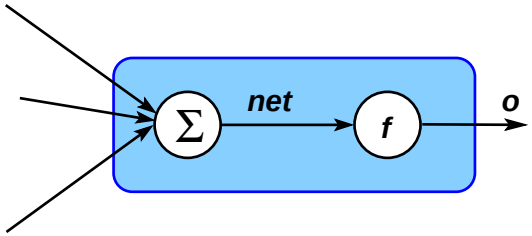
pentru o reţea feed-forward cu 2 niveluri,

folosind o funcţie de activare oarecare (derivabilă) f

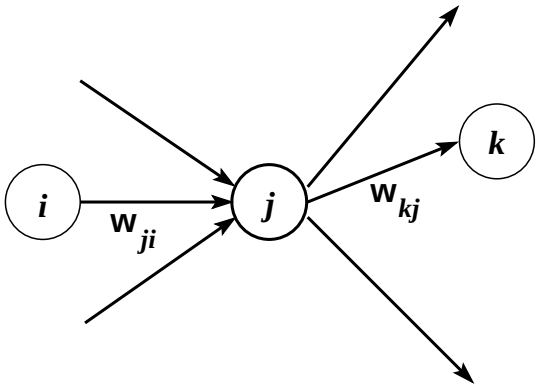
CMU, 2008 fall, Eric Xing, HW2, pr. 2.2

La acest exerciţiu veţi deriva regula de actualizare pentru algoritmul de retro-propagare (varianta stochastică) pentru o reţea neuronală artificială de tip "feed-forward", cu două niveluri de unităţi sigmoidale. Se consideră D unităţi de intrare, H unităţi ascunse şi K unităţi de ieşire.

Funcţia de eroare cu care se lucrează este $E(\bar{w}) = \frac{1}{2}\sum_k (t_k - o_k)^2$, unde $o_k = f(net_k) = \frac{1}{1+e^{c\, net_k}}$, iar c este o constantă.



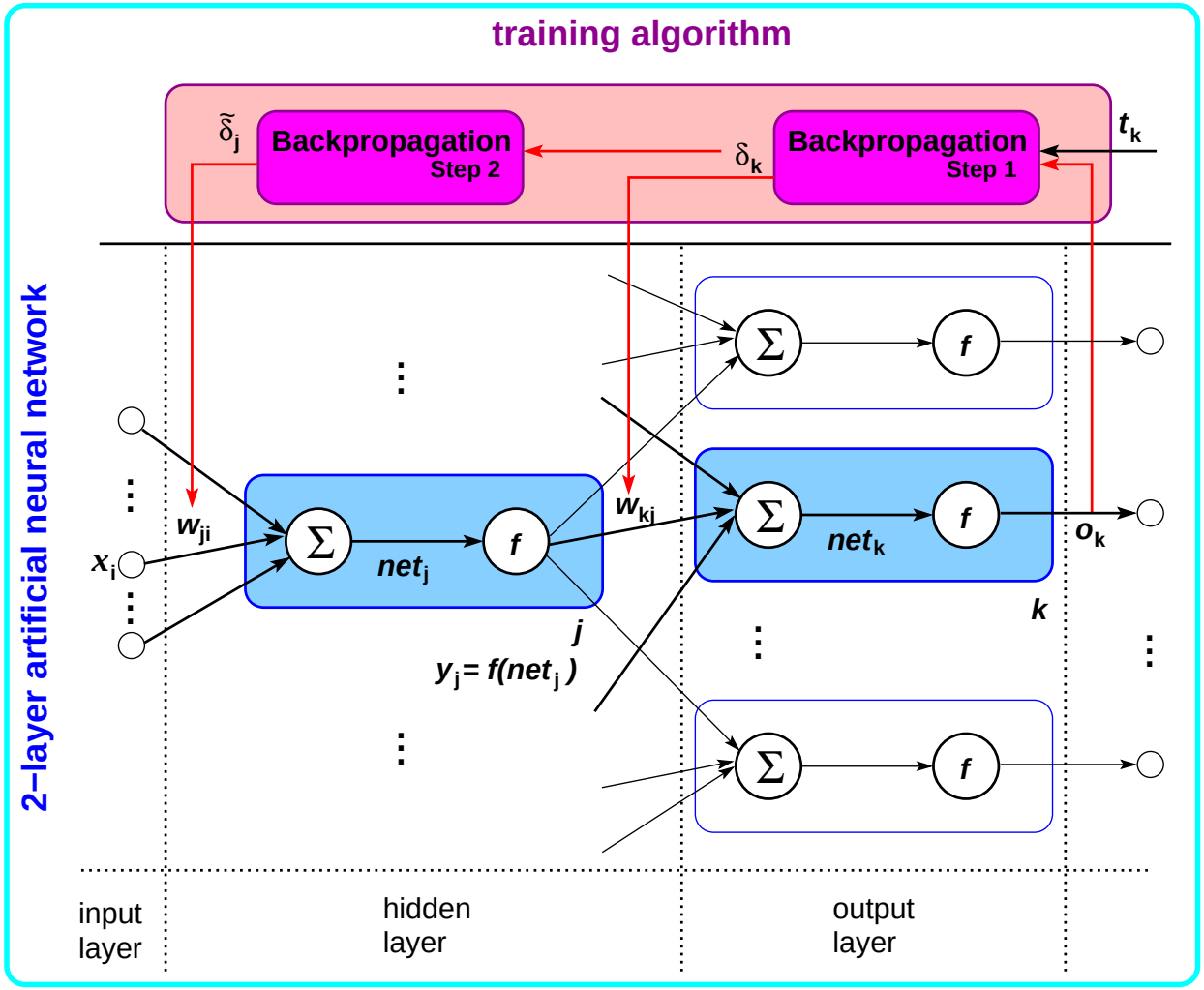
a. Fie w_{kj} ponderea conexiunii de la unitatea ascunsă j către unitatea de ieşire k. Arătaţi că regula de actualizare pentru w_{kj} este de forma $w_{kj} \leftarrow w_{kj} + \eta\delta_k y_j$, unde y_j este ieşirea unităţii ascunse j, iar $\delta_k = f'(net_k)(t_k - o_k)$, cu $net_k = \sum_{j'} w_{kj'}\, y_{j'}$.



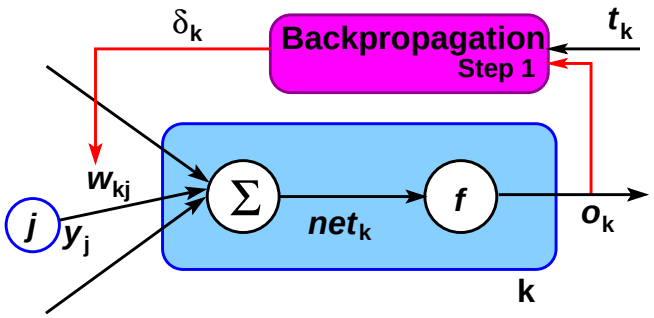
b. Arătaţi că regulile de actualizare pentru ponderile care corespund conexiunilor input-to-hidden sunt de forma $w_{ji} \leftarrow w_{ji} + \eta\tilde{\delta}_j x_i$, unde x_i este intrarea i, iar $\tilde{\delta}_j = f'(net_j)[\sum_{k=1}^{K} w_{kj}\delta_k]$, cu $net_j = \sum_{i'} w_{ji'}\, x_{i'}$.

Răspuns

***Observaţie*: Vom face rezolvarea problemei în raport cu o funcţie oarecare f derivabilă, lăsată nespecificată. Forma particulară dată în enunţ pentru funcţia f nu are nicio consecinţă particulară asupra raţionamentului dezvoltat în continuare.**


training algorithm
Backpropagation
Step 2
Backpropagation
Step 1
$\tilde{\delta}_j$
δ_k
t_k
2–layer artificial neural network
w_{ji}
x_i
net_j
f
w_{kj}
net_k
o_k
k
j
$y_j = f(net_j)$
input
layer
hidden
layer
output
layer

a. Conform metodei gradientului descendent, $w_{kj} \leftarrow w_{kj} - \eta \dfrac{\partial E}{\partial w_{kj}}$, **unde** η **este o constantă pozitivă (rata de învăţare).**



Intuitiv, ponderea w_{kj} **influenţează valoarea funcţiei de eroare** E **(doar) prin intermediul lui** net_k**. (S-a notat cu** net_k **suma** $\sum_{j'} w_{kj'} y_{j'}$**.) Aşadar, este necesară aplicarea formulei pentru derivarea unei compuneri de funcţii derivabile:**

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \left(\frac{\partial}{\partial net_k}\frac{1}{2}\sum_{k'=1}^{K}(t_{k'} - o_{k'})^2\right)\left(\frac{\partial}{\partial w_{kj}}\sum_{j'=0}^{n_k} w_{kj'} y_{j'}\right) = \left(\frac{\partial}{\partial net_k}\frac{1}{2}(t_k - f(net_k))^2\right) y_j$$

$$= y_j\left(\frac{1}{2}2(t_k - f(net_k))\frac{\partial}{\partial net_k}(t_k - f(net_k))\right) = -y_j(t_k - f(net_k))f'(net_k)$$

unde n_k **este numărul de intrări ale unităţii** k**. În particular,** $n_k = H$ **dacă se consideră toate conexiunile posibile (hidden-to-output).**

Dacă notăm $\delta_k = (t_k - f(net_k))f'(net_k)$**, atunci avem** $\dfrac{\partial E}{\partial w_{kj}} = -\delta_k y_j$**, şi**

$$w_{kj} \leftarrow w_{kj} - \eta\frac{\partial E}{\partial w_{kj}} = w_{kj} + \eta\delta_k y_j$$

b. Ca şi la punctul precedent, $w_{ji} \leftarrow w_{ji} - \eta \dfrac{\partial E}{\partial w_{ji}}$**. Ponderea** w_{ji} **influenţează valoarea funcţiei de eroare** E **(doar) prin intermediul lui** net_j**. Folosim din nou formula pentru derivarea unei compuneri de funcţii derivabile şi ţinem cont că** $net_j = \sum_{i'} w_{ji'} x_{i'}$**:**

$$\begin{aligned}
\frac{\partial E}{\partial w_{ji}} &= \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \left(\frac{\partial}{\partial net_j}\frac{1}{2}\sum_{k=1}^{K}(t_k - o_k)^2\right)\left(\frac{\partial}{\partial w_{ji}}\sum_{i'=0}^{D} w_{ji'}x_{i'}\right) \\
&= \left(\frac{1}{2}\sum_{k=1}^{K} 2(t_k - o_k)\frac{\partial}{\partial net_j}(t_k - o_k)\right)x_i = -x_i\sum_{k=1}^{K}(t_k - o_k)\frac{\partial o_k}{\partial net_j}
\end{aligned}$$

Pe de altă parte, valoarea net_j **influenţează valoarea** o_k **(doar) prin intermediul lui** net_k**. Aşadar,**

$$\begin{aligned}
\frac{\partial o_k}{\partial net_j} &= \frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial net_j} = f'(net_k)\frac{\partial}{\partial net_j}\sum_{j'=0}^{n_k} w_{kj'}y_{j'} \\
&= f'(net_k)\frac{\partial}{\partial net_j}\sum_{j'=0}^{n_k} w_{kj'}f(net_{j'}) = f'(net_k)f'(net_j)w_{kj}
\end{aligned}$$

Înlocuind acest rezultat în egalitatea precedentă, vom avea:

$$\begin{aligned}\frac{\partial E}{\partial w_{ji}} &= -x_i \sum_{k=1}^{K} (t_k - o_k) f'(net_k) f'(net_j) w_{kj} \\ &= -x_i f'(net_j) \sum_{k=1}^{K} (t_k - o_k) f'(net_k) w_{kj} = -x_i f'(net_j) \sum_{k=1}^{K} \delta_k w_{kj}\end{aligned}$$

Folosind notaţia $\widetilde{\delta_j} = f'(net_j) \sum_{k=1}^{K} \delta_k w_{kj}$**, egalitatea de mai sus devine** $\frac{\partial E}{\partial w_{ji}} = -x_i \widetilde{\delta_j}$**, iar regula de actualizare a ponderii se transformă în:**

$$w_{ji} \leftarrow w_{ji} + \eta \widetilde{\delta_j} x_i$$

Observaţie

În mod similar cu demonstraţia din cartea *Machine Learning* de Tom Mitchell, pag. 101-103, demonstraţia de aici poate fi extinsă în mod facil la reţele neuronale feed-forward cu un număr oarecare de niveluri ascunse.

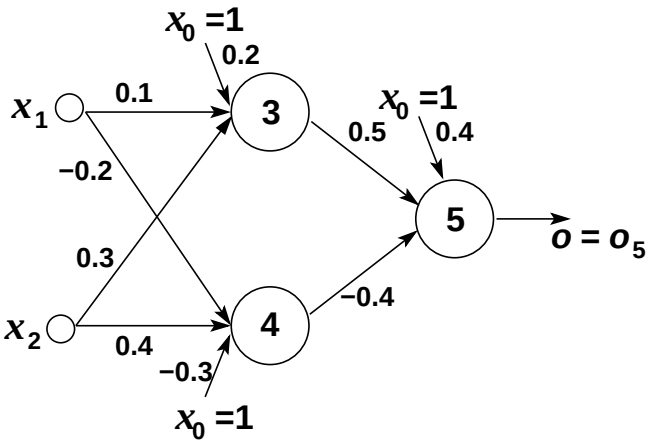
De asemenea, mai este posibilă *încă o generalizare*: funcţia de activare f poate fi diferită de la o unitate la alta (sigmoidală, generalizat-sigmoidală (cu o anumită constantă c fixată), liniară etc). În demonstraţie va fi suficient să ataşăm simbolului f indicele unităţii neuronale respective.

Aplicarea algoritmului de retro-propagare pe o reţea feed-forward de unităţi sigmoidale cu 2 niveluri

prelucrare de Liviu Ciortuz, după un exemplu din

,,Apprentissage artificiel“, A. Cornuéjols, L. Miclet, 2010, pag. 332, 341

Reţeaua neuronală din figura alăturată este formată din unităţi sigmoidale.



a. Care este rezultatul produs de această reţea pentru intrarea $x \overset{not.}{=} (x_1, x_2) = (1, 1)$**?**

b. Executaţi manual prima iteraţie a algoritmului de retro-propagare pe reţeaua dată. Presupunem că în cazul intrării $x = (1, 1)$ **ieşirea produsă de reţea ar trebui să fie** $t = 0$**. Luând rata de învăţare** $\eta = 1$**, precizaţi care vor fi**

- **valorile ponderilor** $w_{30}, w_{31}, w_{32}, w_{40}, w_{41}, w_{42}, w_{50}, w_{51}, w_{52}$, **după aplicarea acestei prime iteraţii în antrenarea reţelei;**[a]
- **noul output produs de reţea (după actualizarea ponderilor) pe aceeaşi intrare** $x = (1, 1)$**.**

***Sugestie*: Pentru a vedea detaliile algoritmului de retro-propagare a erorii, consultaţi cartea *Machine Learning* de Tom Mitchell, pag. 98.**

c. Comparaţi rezultatele de la punctele a **şi** b**. Ce constataţi?**

[a] Semnificaţia pentru w_{ji} este cea din cartea *Machine Learning*a lui Tom Mitchell, la pag. 98: w_{ji} este ponderea de pe arcul/legătura de la unitatea (sau intrarea) i către unitatea j.

Solution (in Romanian)

a.

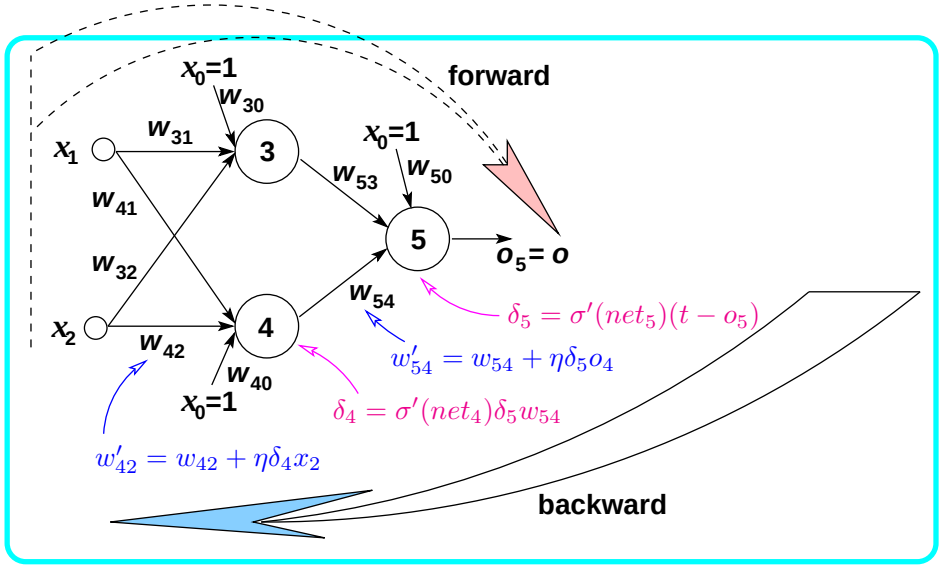
i	$net_i = \sum_j w_{ij} x_j$	$o_i = \sigma(net_i)$
3	$0.2 + 0.1 + 0.3 = 0.6$	$1/(1 + e^{-0.6}) \simeq 0.646$
4	$-0.3 - 0.2 + 0.4 = -0.1$	$1/(1 + e^{0.1}) \simeq 0.475$
5	$0.4 + 0.5 \cdot 0.646 - 0.4 \cdot 0.475 = 0.533$	$1/(1 + e^{-0.533}) \simeq 0.630$

Aşadar, output-ul produs de reţea este 0.63**.**

b.

***Precizare*:**

Ca să facilităm / sintetizăm înţelegerea modului în care se aplică acest algoritm, în schema alăturată am arătat cum anume vor fi calculate noile valori ale ponderilor w, precum şi cantităţile δ implicate în execuţia unei iteraţii a algoritmului. Ilustrarea se rezumă la parcurgerea (în ordinea output-to-input) unuia dintre drumurile din această reţea, şi anume drumul 5, 4, 2. Extensia la celelalte căi este facilă.



În această imagine (dar numai aici) am notat cu w' noile valori pentru ponderi (calculate în ultimul pas al iteraţiei), pentru a nu fi confundate cu vechile valori, care intervin în calculul cantităţilor δ.

Conform algoritmului de retro-propagare,

$$\begin{aligned}\delta_5 &\overset{def.}{=} -\frac{\partial E}{\partial net_5} = \sigma'(net_5)(t-o) = \sigma(net_5)(1-\sigma(net_5))(t-o)\\ &= o(1-o)(t-o) = 0.63(1-0.63)(0-0.63) = -0.147\end{aligned}$$

Output-ul neuronului 3 constituie unul din input-urile neuronului 5.
În notaţia folosită de Tom Mitchell, vom scrie $Downstream(3) = \{5\}$**.**
Aşadar,

$$\delta_3 \overset{def.}{=} -\frac{\partial E}{\partial net_3} = o_3 \cdot (1-o_3) \cdot \delta_5 \cdot w_{53} = 0.646 \cdot (1-0.646) \cdot (-0.147) \cdot 0.5 \simeq -0.017$$

În mod similar, $Downstream(4) = \{5\}$**, şi**

$$\delta_4 \overset{def.}{=} -\frac{\partial E}{\partial net_4} = o_4 \cdot (1-o_4) \cdot \delta_5 \cdot w_{54} = 0.475 \cdot (1-0.475) \cdot (-0.147) \cdot (-0.4) \simeq 0.015$$

Întrucât am terminat de calculat toate cantităţile de tip δ_j, vom trece acum la actualizarea ponderilor reţelei. Mai întâi vom calcula noile valori pentru ponderile de pe conexiunile hidden-to-output, deci vom avea:

$$w_{5j} \leftarrow w_{5j} + \Delta w_{5j} \textbf{ cu } \Delta w_{5j} = \eta \delta_5 o_j = \delta_5 o_j \textbf{ pentru } j \in \{0, 3, 4\}.$$

Aşadar,

$$\begin{cases} \Delta w_{50} = -0.147 \cdot 1 = -0.147 \\ \Delta w_{53} = -0.147 \cdot 0.646 \simeq -0.095 \\ \Delta w_{54} = -0.147 \cdot 0.475 \simeq -0.070 \end{cases} \Rightarrow \begin{cases} w_{50} \leftarrow 0.4 - 0.147 \simeq 0.253 \\ w_{53} \leftarrow 0.5 - 0.1 = 0.405 \\ w_{54} \leftarrow -0.4 - 0.070 = -0.470 \end{cases}$$

Apoi vom actualiza ponderile care corespund conexiunilor de tip input-to-hidden.

Ştim că $\Delta w_{3i} = \eta \delta_3 x_i$ pentru $i \in \{0, 1, 2\}$, aşadar input-ul $x_0 = x_1 = x_2 = 1$ va implica $\Delta w_{30} = \Delta w_{31} = \Delta w_{32} = 1 \cdot (-0.017) \cdot 1 = -0.017$.

În consecinţă,

$$\begin{cases} w_{30} \leftarrow 0.2 - 0.017 = 0.183 \\ w_{31} \leftarrow 0.1 - 0.017 = 0.083 \\ w_{32} \leftarrow 0.3 - 0.017 = 0.283 \end{cases}$$

Similar, fiindcă $\Delta w_{4i} = \eta \delta_4 x_i$ pentru $i \in \{0, 1, 2\}$, rezultă că $\Delta w_{40} = \Delta w_{41} = \Delta w_{42} = 1 \cdot 0.015 \cdot 1 = 0.015$ şi, prin urmare:

$$\begin{cases} w_{40} \leftarrow -0.3 + 0.015 = -0.285 \\ w_{41} \leftarrow -0.2 + 0.015 = -0.185 \\ w_{42} \leftarrow 0.4 + 0.015 = 0.415 \end{cases}$$

Acum putem calcula noua valoare produsă de reţeaua neuronală:

i	$net_i = \sum_j w_{ij} x_j$	$o_i = \sigma(net_i)$
3	$0.183 + 0.083 + 0.283 = 0.549$	$1/(1 + e^{-0.549}) \simeq 0.634$
4	$-0.285 - 0.185 + 0.415 = -0.055$	$1/(1 + e^{0.055}) \simeq 0.486$
5	$0.253 + 0.405 \cdot 0.634 - 0.47 \cdot 0.486 = 0.281$	$1/(1 + e^{-0.281}) \simeq 0.569$

c. Valoarea produsă de reţea după ce a fost aplicată o iteraţie din algoritmul de retro-propagare (0.569) **este mai aproape de valoarea-target** (0) **decât fusese output-ul produs de reţea înainte de aplicarea acestei prime iteraţii** (0.63)**. Este de aşteptat ca rezultatul să se îmbunătăţească la execuţia următoarelor iteraţii ale algoritmului.**

Regularizarea perceptronilor / reţelelor

pentru prevenirea overfitting-ului,

prin extinderea cu încă un termen

a funcţiei de cost

Tom Mitchell, *Machine Learning* book, 1997, pr. 4.10

Consider the following error function (which is an alternative to the sum of squares function):

$$E(\bar{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in Outputs} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

Derive the gradient descent update rule for this definition of E.
Show that it can be implemented by multiplying each weight by some constant before performing the standard gradient descent update ($w \leftarrow w + \Delta w$).

***Convenţie de notaţie* (in Romanian):**

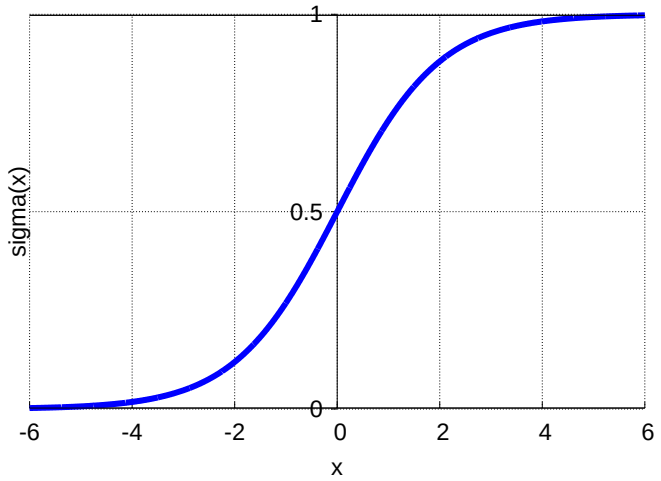
Am folosit mai sus obişnuita [de acum] scriere a variabilelor w, şi anume w_{ji} este ponderea care corespunde conexiunii dinspre unitatea i către unitatea j.

Notaţiile t_{kd} şi o_{kd} corespund valorii-target şi respectiv output-ului unităţii liniare k de pe nivelul de ieşire al reţelei, corespunzător input-ului x_d.

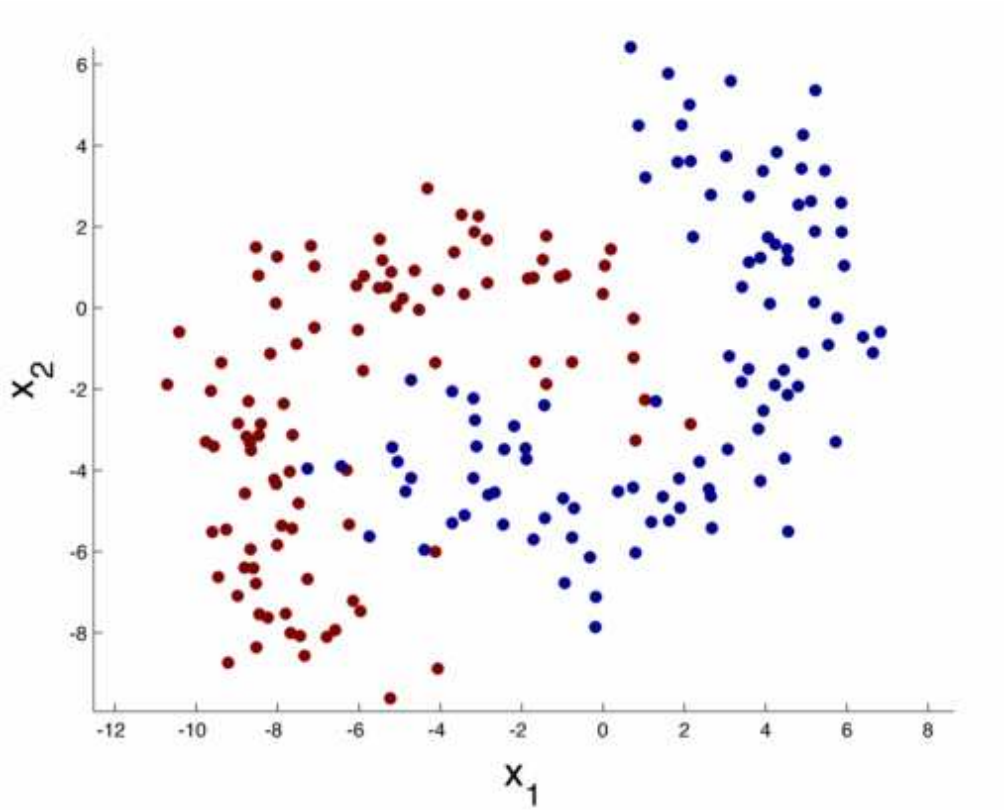
Remark (in Romanian)

Suma $\sum_i w_i^2$ se poate scrie ca $||w||^2$, unde $||w||$ notează norma vectorului w. Intuitiv, minimizarea funcţiei obiectiv $E(\bar{w})$ va implica menţinerea lui $||w||^2$ la o valoare destul de redusă.

Efectul practic, în cazul folosirii de unităţi sigmoidale este următorul: graniţa (suprafaţa) de decizie calculată de către reţea pentru ponderi w mici (în modul) este aproape liniară — a se vedea graficul funcţiei σ în jurul originii —, ceea ce o împiedică să se „muleze“ în jurul neregularităţilor din datele de antrenament.

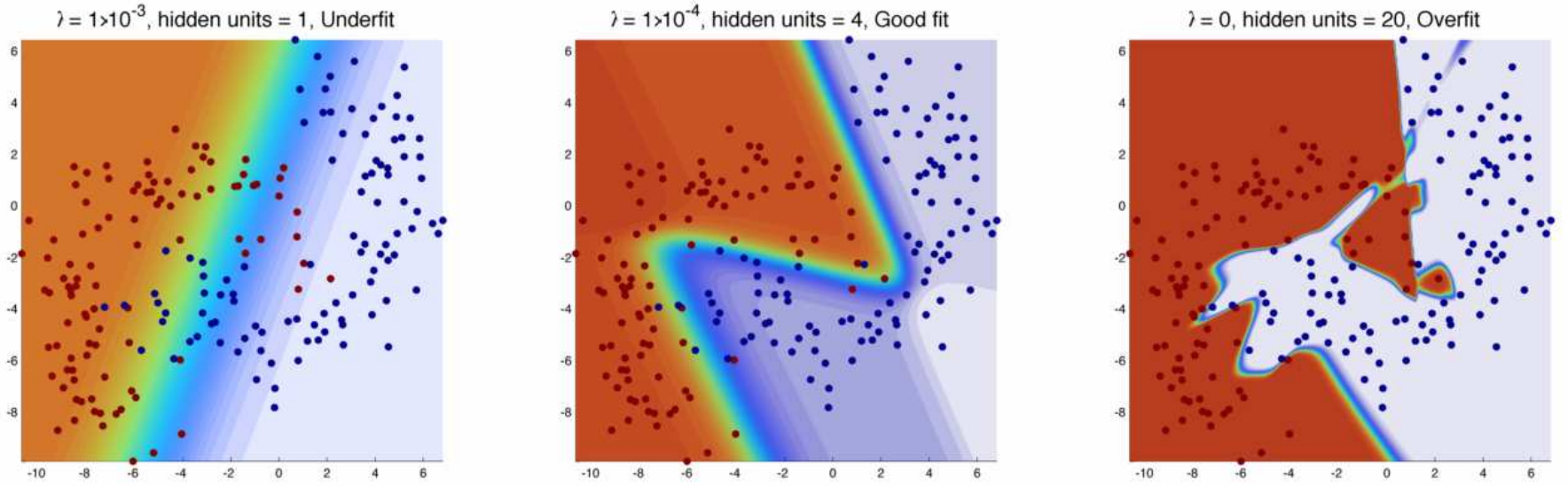


Exemplification: The Banana Dataset



Cf. CMU, 2014 fall, W. Cohen, Z. Bar-Joseph, HW3, pr. 2.

Exemplification (cont'd)



Cf. CMU, 2014 fall, W. Cohen, Z. Bar-Joseph, HW3, pr. 2.

Remark (in Romanian)

Pentru simplitate, vom considera că reţea noastră este de tip feed-forward şi că are [doar] două niveluri, iar unităţile sunt liniare.

Din acelaşi motiv, vom lucra cu varianta stochastică/incrementală a algoritmului de retro-propagare, adică vom considera o singură instanţă de antrenament pe ciclu/„epocă“ de antrenare.

Demonstraţia următoare se poate extinde imediat la cazul unităţilor de diverse tipuri (de exemplu sigmoidal etc.), şi la varianta “batch” a algoritmului de retro-propagare.

Solution (in Romanian)

În condiţiile stabilite mai înainte, putem scrie:

$$E(\bar{w}) = \frac{1}{2} \sum_{k \in Outputs} (t_k - o_k)^2 + \gamma ||\bar{w}||^2,$$

unde t_k şi o_k corespund valorii-target şi respectiv output-ului unităţii liniare k de pe nivelul de ieşire al reţelei, pentru instanţa de intrare x considerată.

În cele ce urmează, w_{kj} va corespunde unei conexiuni hidden-to-output (unde j indică o unitate de pe nivelul ascuns, iar k o unitate de pe nivelul de ieşire), iar w_{ji} va corespunde unei conexiuni input-to-hidden (i – input, j – hidden).

Prin urmare, pentru ponderile de pe conexiunile hidden-to-output, vom avea:

$$\begin{aligned}
\frac{\partial E}{\partial w_{kj}} &= \frac{\partial}{\partial w_{kj}}\left[\frac{1}{2}(t_k - o_k)^2 + \gamma w_{kj}^2\right] + \sum_{k' \neq k} \frac{\partial}{\partial w_{kj}}\left[\frac{1}{2}(t_{k'} - o_{k'})^2 + \gamma w_{k'j}^2\right] \\
&= -\frac{1}{2} 2(t_k - o_k) \frac{\partial}{\partial w_{kj}} o_k + 2\gamma w_{kj} \\
&= -(t_k - o_k) \frac{\partial}{\partial w_{kj}} \sum_{j'} w_{kj'} y_{j'} + 2\gamma w_{kj} = -(t_k - o_k) y_j + 2\gamma w_{kj}
\end{aligned}$$

Aşadar,

$$w_{kj} \leftarrow w_{kj} - \eta \frac{\partial E}{\partial w_{kj}} = w_{kj} + \eta[(t_k - o_k) y_j - 2\gamma w_{kj}] = (1 - 2\eta\gamma) w_{kj} + \eta(t_k - o_k) y_j$$

Tratăm acum cazul ponderilor de pe conexiunile input-to-hidden.
Notăm $Downstream(j)$ mulţimea tuturor indicilor k cu proprietatea că există conexiune de la unitatea j către unitatea k.

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left\{ \left[\sum_{k \in Downstream(j)} \frac{1}{2}(t_k - o_k)^2 \right] + \gamma w_{ji}^2 \right\} = 2\gamma w_{ji} - \sum_{k \in Downstream(j)} (t_k - o_k) \frac{\partial}{\partial w_{ji}} o_k$$

$$= 2\gamma w_{ji} - \sum_{k \in Downstream(j)} \left[(t_k - o_k) \frac{\partial}{\partial w_{ji}} \left(\sum_{j'} w_{kj'} y_{j'} \right) \right]$$

$$= 2\gamma w_{ji} - \sum_{k \in Downstream(j)} \left\{ (t_k - o_k) \frac{\partial}{\partial w_{ji}} \left[\sum_{j'} w_{kj'} \left(\sum_{i'} w_{j'i'} x_{i'} \right) \right] \right\}$$

$$= 2\gamma w_{ji} - \sum_{k \in Downstream(j)} \left[(t_k - o_k) \frac{\partial}{\partial w_{ji}} \left(\sum_{j'} \sum_{i'} w_{kj'} \, w_{j'i'} \, x_{i'} \right) \right]$$

$$= 2\gamma w_{ji} - x_i \left(\sum_{k \in Downstream(j)} (t_k - o_k) w_{kj} \right)$$

Aşadar,

$$
\begin{aligned}
w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E}{\partial w_{ji}} &= w_{ji} + \eta \left[x_i \left(\sum_{k \in Downstream(j)} w_{kj}(t_k - o_k) \right) - 2\gamma w_{ji} \right] \\
&= (1 - 2\eta\gamma) w_{ji} + \eta x_i \left(\sum_{k \in Downstream(j)} w_{kj}(t_k - o_k) \right)
\end{aligned}
$$

Concluzie

Analizând regulile de actualizare a ponderilor deduse mai sus, se observă că în ambele cazuri (input-to-hidden şi hidden-to-output) ele sunt de forma $w \leftarrow (1 - 2\eta\gamma)w + \Delta w$**.**

Deosebirea faţă de cazul clasic ($w \leftarrow w + \Delta w$**), este evidentă. Întrucât algoritmul de retro-propagare iniţializează ponderile** w **cu valori apropiate de** 0 **— iar în practică se folosesc valori mici pentru constantele** η **şi** γ **—, lucrând cu varianta funcţiei de eroare propusă în această problemă rezultă valori ale ponderilor** w **mai aproape de** 0 **(decât în varianta clasică).**

[Networks of] sigmoidal units: The cross-entropy error function

CMU, 2011 fall, Eric Xing, HW1, pr. 3.3

Fie datele de antrenament $D = (X, t) = \{(x_i, t_i)\}, i = 1, 2, \ldots, N$. De exemplu, x_i (element dintr-un spaţiu $\mathbb{R}^d$) poate fi imaginea unei feţe, în vreme ce t_i este o etichetă binară care are valoarea 0 dacă faţa respectivă este a unui bărbat, respectiv 1 în cazul unei femei.

Vom considera o unitate neuronală de tip sigmoidal.[a] Notând cu y valoarea reală produsă de această unitate, rezultă că $y \in (0, 1)$. Este natural să interpretăm acest output ca fiind probabilitatea ca eticheta booleană t să fie 1. Formal, putem scrie $y \stackrel{def.}{=} P(t = 1 \mid x; w)$.

În consecinţă, este natural să căutăm valori convenabile pentru ponderile w folosind principiul verosimilităţii maxime (MLE),[b] sub forma următoare:

$$w_{MLE} = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{N} P(t_i \mid x_i; w).$$

---

[a] Mai general, se poate considera o reţea formată din unităţi sigmoidale. (De fapt, aşa a fost formulată problema originală de la CMU.) Într-un astfel de caz este însă suficient să presupunem că tipul unităţii / unităţilor de pe nivelul de ieşire din reţea este cel sigmoidal.

[b] De fapt, în cazul de faţă vom lucra cu verosimilitate condiţională.

a. Arătaţi că a maximiza expresia $\prod_{i=1}^{N} P(t_i \mid x_i; w)$ revine la a minimiza expresia următoare, despre care putem spune, după o observare atentă, că este o cross-entropie:

$$E(w) = -\sum_{i=1}^{N} [t_i \ln y_i + (1 - t_i) \ln(1 - y_i)],$$

unde y_i este output-ul produs de reţeaua neuronală pentru exemplul x_i.

b. Să presupunem că este posibil ca etichtele datelor de antrenament să fi fost puse eronat, şi anume cu probabilitatea ε. Considerând că datele de antrenament sunt distribuite în mod identic şi independent unele de altele, scrieţi expresia funcţiei de eroare / cost $E^*(w, \varepsilon)$ care corespunde log-verosimilităţii cu semn schimbat (engl., the negative log likelihood).

c. Verificaţi că funcţia de eroare $E(w)$ se obţine din $E^*(w, \varepsilon)$ pentru cazul particular $\varepsilon = 0$.

d. Explicaţi de ce anume funcţia de eroare $E^*(w, \varepsilon)$ va face ca modelul obţinut să fie [mai] robust în raport cu date incorect etichetate, spre deosebire de funcţia uzuală $E(w)$.

a. Conform enunţului, $y_i \overset{def.}{=} P(t_i = 1|x_i; w)$**. Aşadar, putem scrie**

$$P(t_i|x_i; w) = \begin{cases} y_i & \textbf{dacă} \quad t_i = 1, \\ 1 - y_i & \textbf{dacă} \quad t_i = 0. \end{cases}$$

Se verifică imediat că, din punct de vedere matematic, relaţia de mai sus se poate exprima în mod echivalent sub forma

$$P(t_i|x_i; w) = (y_i)^{t_i}(1 - y_i)^{1-t_i}.$$

Această expresie, în ciuda faptului că este mai puţin intuitivă, este mult mai convenabilă pentru calculele pe care trebuie să le facem în vederea aplicării principiului verosimilităţii maxime:

$$\begin{aligned} w_{MLE} &\overset{def.}{=} \operatorname*{argmax}_w P(t_1, \ldots, t_n|x_1, \ldots, x_n; w) \overset{i.i.d.}{=} \operatorname*{argmax}_w \prod_{i=1}^{N} P(t_i \mid x_i; w) \\ &= \operatorname*{argmax}_w \sum_{i=1}^{N} \ln P(t_i \mid x_i; w) = \operatorname*{argmax}_w \sum_{i=1}^{N} \ln (y_i)^{t_i}(1 - y_i)^{1-t_i} \\ &= \operatorname*{argmax}_w \sum_{i=1}^{N} (t_i \ln y_i + (1 - t_i) \ln(1 - y_i)) = \operatorname*{argmax}_w (-E(w)) = \operatorname*{argmin}_w E(w). \end{aligned}$$

b. În cazul în care etichetarea datelor de antrenament s-a făcut în mod eronat, cu probabilitatea ε, rezultă imediat că

$$x_i \textbf{ a fost etichetat cu } \begin{cases} 1 & \textbf{cu probabilitatea } 1-\varepsilon\textbf{, dacă } t_i = 1, \\ 0 & \textbf{cu probabilitatea } \varepsilon\textbf{, dacă } t_i = 1, \\ 0 & \textbf{cu probabilitatea } 1-\varepsilon\textbf{, dacă } t_i = 0, \\ 1 & \textbf{cu probabilitatea } \varepsilon\textbf{, dacă } t_i = 0. \end{cases}$$

Prin urmare, putem scrie în manieră condensată

$$P(t_i|x_i;w) = \begin{cases} y_i(1-\varepsilon) + (1-y_i)\varepsilon & \textbf{dacă} \quad t_i = 1, \\ (1-y_i)(1-\varepsilon) + y_i\varepsilon & \textbf{dacă} \quad t_i = 0. \end{cases}$$

Ba chiar şi mai mult, această expresie este echivalentă cu următoarea:

$$P(t_i|x_i;w) = [y_i(1-\varepsilon) + (1-y_i)\varepsilon]^{t_i}\,[(1-y_i)(1-\varepsilon) + y_i\varepsilon]^{1-t_i}.$$

Făcând un calcul similar cu cel de la punctul a, va rezulta:[a]

$$E^*(w,\varepsilon) \quad = \quad -\sum_{i=1}^{N}\{t_i \ln[(y_i(1-\varepsilon) + (1-y_i)\varepsilon] + (1-t_i)\ln[(1-y_i)(1-\varepsilon) + y_i\varepsilon]\}.$$

[a] Se verifică imediat că $[y_i(1-\varepsilon) + (1-y_i)\varepsilon] + [(1-y_i)(1-\varepsilon) + y_i\varepsilon] = 1$.

c. Înlocuind parametrul ε cu valoarea 0 în expresia pe care tocmai am obţinut-o mai sus, vom avea:

$$\begin{aligned} E^*(w,0) &= -\sum_{i=1}^{N}\{[t_i \ln y_i + (1-y_i)\times 0] + (1-t_i)\ln[(1-y_i) + y_i \times 0]\} \\ &= -\sum_{i=1}^{N}[t_i \ln y_i + (1-t_i)\ln(1-y_i)] = E(w) \end{aligned}$$

d. Vom considera ca *exemplu* cazul extrem (însă instructiv) în care toate etichetele au fost puse incorect. Dacă am folosi funcţia de eroare $E(w)$, atunci modelul produs de algoritmul de retro-propagare ar fi complet eronat. Dacă în schimb vom folosi funcţia de eroare $E^*(w,\varepsilon)$ cu $\varepsilon = 1$, atunci algoritmul de retro-propagare va încerca să înveţe un model care clasifică datele exact invers faţă de etichetele originale, ceea ce corespunde obiectivului nostru.

Important Remarks (in Romanian)

1.
Analizând cu atenţie demonstraţia pentru deducerea regulilor de actualizarea ponderilor w pentru algoritmul de retro-propagare atunci când funcţia obiectiv este prezenta cross-entropie (vezi CMU, 2008 fall, E. Xing, HW2, pr. 2.2), şi
facând comparaţia cu demonstraţia din cartea lui Tom Mitchell (alternativ, vezi CMU, 2008(?) spring, HW2, pr. 2.2-4) — cazul utilizării semi-sumei pătratelor erorilor ca funcţie obiectiv —,
se contată că singura diferenţă [între cele două seturi de reguli de actualizare] apare în expresia lui δ_j pentru j indice de unitate de pe nivelul ascuns:
în acel caz, în componenta lui δ_j este inclus factorul $f'(net_j)$, însă aici (folosind σ, funcţia sigmoidală) acest factor lipseşte.
În rest, totul este la fel!

2.
Ar fi interesant de văzut care este forma regulilor de actualizare atunci cînd în locul prezentului criteriu de optimizat se foloseste $E(w, \varepsilon)$ definit [în acest set de slide-uri] la punctele b şi c.

Deep neural networks:

gradient vanishing / explosion

CMU, 2015 fall, Eric Xing, Zib Bar-Joseph, HW3, pr. 1.3

In this problem we will study the difficulty of back-propagation in training deep neural networks. For simplicity, we consider the simplest deep neural network: one with just a single neuron in each layer, where the output of the neuron in the jth layer is $z_j \overset{not.}{=} f(net_j) = f(w_j z_{j-1} + b_j)$. Here f is some activation function whose derivative on x is $f'(x)$. Let m be the number of layers in the neural network, L the training loss function.

a. Derive the derivative of L w.r.t. b_1 (the bias of the neuron in the first layer).

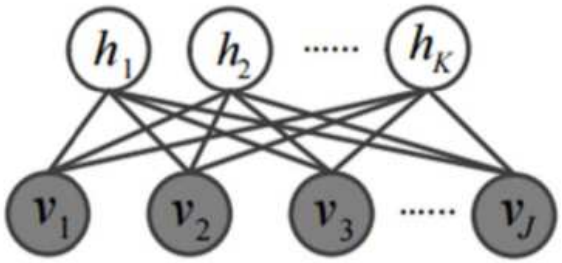
b. Assume the activation function is the usual sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$. The weights $\bar{w}$ are initialized to be $|w_j| < 1$ $(j = 1, \ldots, m)$.

Explain why the above gradient $(\partial L/\partial b_1)$ tends to vanish $(\to 0)$ when m is large.

Even if $|w|$ is large, the above gradient would also tend to vanish, rather than explode $(\to \infty)$. Explain why.

c. One of the approaches to (partially) address the gradient vanishing/explosion problem is to use the *rectified linear* (ReL) activation function instead of the sigmoid. The ReL activation function is $\sigma(x) = \max\{0, x\}$. Explain why ReL can alleviate the gradient vanishing problem as faced by sigmoid.

d. A second approach to (partially) address the gradient vanishing/explosion problem is *layer-wise pre-training.* *Restricted Boltzmann machine* (RBM) is one of the widely-used models for layer-wise pre-training.
The nearby figure shows an example of RBM which includes K hidden units $\bar{h}$, and J input units $\bar{v}$.



Let us define the *joint distribution* over $\bar{v}$ and $\bar{h}$ as having the following general form:

$$P(\bar{v}, \bar{h}) = \frac{1}{Z} \exp\left(\sum_i \theta_i \phi_i(\bar{v}, \bar{h})\right),$$

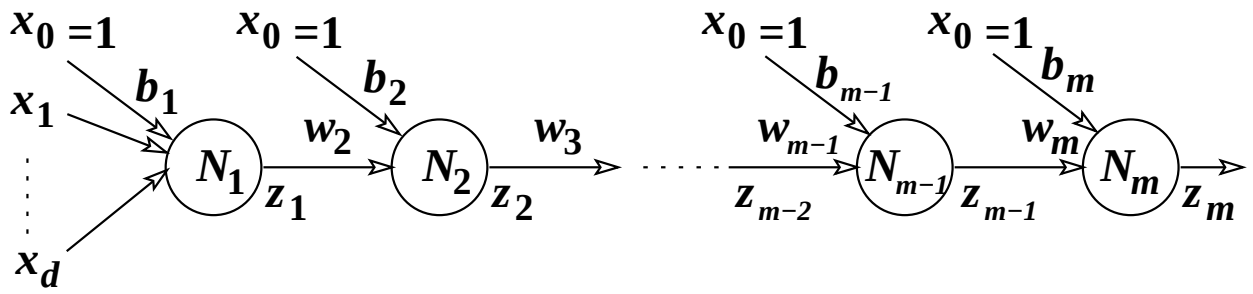
where $Z = \sum_{\bar{v}', \bar{h}'} \exp\left(\sum_i \theta_i \phi_i(\bar{v}', \bar{h}')\right)$ is the normalization term; $\phi_i(\bar{v}, \bar{h})$ are some features; θ_i are the parameters corresponding to the weights in the RBM. Consider the simplest learning algorithm, gradient descent.

Show that

$$\frac{\partial \log P(\bar{v})}{\partial \theta_i} = \sum_{\bar{h}} \phi_i(\bar{v}, \bar{h}) P(\bar{h}|\bar{v}) - \sum_{\bar{v}', \bar{h}'} \phi_i(\bar{v}', \bar{h}') P(\bar{v}', \bar{h}').$$

Răspuns

a. Ilustrăm reţeaua neuronală profundă definită în enunţ:



$$
\begin{aligned}
\Rightarrow z_1 &= f(net_1) = f(b_1 + w_{11}x_1 + \ldots + w_{1d}x_d) \\
z_2 &= f(net_2) = f(b_2 + w_2 z_1) \\
&\ldots \\
o = z_m &= f(net_m) = f(b_m + w_m z_{m-1})
\end{aligned}
$$

Funcţia de pierdere (engl., loss function) L se va exprima astfel:

$$
\begin{aligned}
&L(w_{11},\ldots,w_{1d},w_2,\ldots,w_m,b_1,\ldots,b_m)\\
&\overset{not.}{=} L(f(\underbrace{net_m}_{b_m+w_m z_{m-1}}))\\
&= L(f(b_m+w_m f(\underbrace{net_{m-1}}_{b_{m-1}+w_{m-1}z_{m-2}})))\\
&\quad \ldots\\
&= L(f(b_m+w_m f(b_{m-1}+w_{m-1}f(b_{m-2}+\ldots+w_3 f(\underbrace{net_2}_{b_2+w_2 z_1})\ldots)))\\
&= L(f(b_m+w_m f(b_{m-1}+w_{m-1}f(b_{m-2}+\ldots+w_3 f(b_2+w_2 z_1)\ldots)))\\
&= L(f(b_m+w_m f(b_{m-1}+w_{m-1}f(b_{m-2}+\ldots+w_3 f(b_2+w_2 f(\underbrace{net_1}_{b_1+w_{11}x_1+\ldots+w_{1d}x_d})\ldots)))
\end{aligned}
$$

Pentru a deriva funcţia L în raport cu b_1, ne vom aminti mai întâi regula de derivare a unei compuneri multiple de funcţii $f_1(f_2(\dots f_n(x)\dots))$. Pe lângă forma clasică pe care o ştim din liceu,

$$f_1'(f_2(\dots f_n(x)\dots)) \cdot f_2'(\dots f_n(x)\dots) \cdot f_n'(x),$$

ea se poate scrie sub forma aşa-numitei *reguli de înlănţuire*:

$$\frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdot \dots \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial x}.$$

Adoptând / aplicând această regulă pentru a calcula derivata parţială cerută în enunţ, vom avea:

$$\begin{aligned}
\frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial net_m} \cdot \frac{\partial net_m}{\partial net_{m-1}} \cdot \dots \cdot \frac{\partial net_2}{\partial net_1} \cdot \frac{\partial net_1}{\partial b_1} \\
&= L'(f(net_m)) \cdot f'(net_m) \cdot \\
&\quad w_m \cdot f'(net_{m-1}) \cdot \\
&\quad w_{m-1} \cdot f'(net_{m-2}) \cdot \\
&\quad \dots \\
&\quad w_2 \cdot f'(net_1) \cdot \\
&\quad 1 \\
&= L'(f(net_m)) \cdot f'(net_1) \cdot \prod_{k=2}^{m} (f'(net_k) \cdot w_k)
\end{aligned}$$

b. Ştim că $\sigma(x) \in (0,\, 1)$ şi $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ pentru orice $x \in \mathbb{R}$, iar maximul funcţiei $z(1 - z)$ este $1/4$ şi se atinge în punctul $z = 1/2$. Prin urmare, $|\prod_{k=2}^{m} \sigma'(net_k)| \leq \left(\frac{1}{4}\right)^{m-1}$.

Întrucât $|w_k| < 1$ pentru orice k (conform ipotezei din enunţ), rezultă imediat că $\left|\frac{\partial L}{\partial b_1}\right| = |L'(\sigma(net_m)) \cdot \sigma'(net_1)| \cdot \prod_{k=2}^{m} |\sigma'(net_k) \cdot w_k|$ tinde la 0 pentru $k \to +\infty$ şi pentru orice input $\bar{x}$ fixat.

Pentru a contracara acest fenomen de „dispariţie“ a gradientului, ar trebui să impunem condiţii de forma $|w_k \sigma'(net_k)| > 1$. Trebuie însă să remarcăm faptul că $\sigma(net_k)$ depinde de w_k: $\sigma(net_k) = \sigma(b_k + w_k z_{k-1})$.

În consecinţă, dacă îi vom da lui w_k posibilitatea să ia valori mari în modul, atunci $z_k \overset{not.}{=} b_k + w_k z_{k-1}$ va lua de asemenea valori mari în modul, iar $\sigma(z_k)$ va tinde fie la 0 fie la 1, deci $\sigma'(z_k) = \sigma(z_k)(1 - \sigma(z_k))$ va tinde la 0.

În sfârşit, se poate verifica în mod riguros că $\lim_{z \to \pm\infty} z\sigma'(z) = 0$ (lăsăm această demonstraţie ca exerciţiu).

c. Dacă f este funcţia de activare ReL, atunci $f'(x) = 0$ pentru $x < 0$ şi $f'(x) = 1$ pentru $x > 0$. Prin urmare, folosind această funcţie putem împiedica „dispariţia“ gradientului.

d. Pornind de la expresia distribuţiei probabiliste $P(\bar{v}, \bar{h})$ care a fost dată în enunţ, explicitând mai întâi constanta de normalizare Z, vom putea scrie apoi expresia distribuţiei marginale $P(\bar{v})$:

$$P(\bar{v}, \bar{h}) = \frac{1}{\sum_{\bar{v}', \bar{h}'} \exp(\sum_k \theta_k \phi_k(\bar{v}', \bar{h}'))} \exp\left(\sum_k \theta_k \phi_k(\bar{v}, \bar{h})\right)$$

$$\Rightarrow P(\bar{v}) = \frac{1}{\sum_{\bar{v}', \bar{h}'} \exp(\sum_k \theta_k \phi_k(\bar{v}', \bar{h}'))} \sum_{\bar{h}} \exp\left(\sum_k \theta_k \phi_k(\bar{v}, \bar{h})\right)$$

Prin urmare,

$$\ln P(\bar{v}) = \ln \sum_{\bar{h}} \exp\left(\sum_k \theta_k \phi_k(\bar{v}, \bar{h})\right) - \ln \sum_{\bar{v}', \bar{h}'} \exp(\sum_k \theta_k \phi_k(\bar{v}', \bar{h}')).$$

În consecinţă, derivata parţială a funcţiei $\ln P(\bar{v})$ în raport cu parametrul θ_i se poate calcula astfel:

$$\frac{\partial \ln P(\bar{v})}{\partial \theta_i} =$$

$$= \frac{\sum_{\bar{h}} \exp\left(\sum_k \theta_k \phi_k(\bar{v}, \bar{h})\right) \cdot \phi_i(\bar{v}, \bar{h})}{\sum_{\bar{h}} \exp\left(\sum_k \theta_k \phi_k(\bar{v}, \bar{h})\right)} - \frac{\sum_{\bar{v}', \bar{h}'} \exp(\sum_k \theta_k \phi_k(\bar{v}', \bar{h}')) \cdot \phi_i(\bar{v}, \bar{h})}{\underbrace{\sum_{\bar{v}', \bar{h}'} \exp(\sum_k \theta_k \phi_k(\bar{v}', \bar{h}'))}_{Z}}$$

$$= \frac{\sum_{\bar{h}} \frac{1}{Z} \exp\left(\sum_k \theta_k \phi_k(\bar{v}, \bar{h})\right) \cdot \phi_i(\bar{v}, \bar{h})}{\sum_{\bar{h}} \frac{1}{Z} \exp\left(\sum_k \theta_k \phi_k(\bar{v}, \bar{h})\right)} - \sum_{\bar{v}', \bar{h}'} \frac{1}{Z} \exp(\sum_k \theta_k \phi_k(\bar{v}', \bar{h}')) \cdot \phi_i(\bar{v}, \bar{h})$$

$$= \frac{\sum_{\bar{h}} P(\bar{v}, \bar{h}) \cdot \phi_i(\bar{v}, \bar{h})}{P(\bar{v})} - \sum_{\bar{v}', \bar{h}'} P(\bar{v}', \bar{h}') \phi_i(\bar{v}', \bar{h}')$$

$$= \sum_{\bar{h}} \frac{P(\bar{v}, \bar{h})}{P(\bar{v})} \cdot \phi_i(\bar{v}, \bar{h}) - \sum_{\bar{v}', \bar{h}'} P(\bar{v}', \bar{h}') \phi_i(\bar{v}', \bar{h}')$$

$$= \sum_{\bar{h}} P(\bar{h}|\bar{v}) \cdot \phi_i(\bar{v}, \bar{h}) - \sum_{\bar{v}', \bar{h}'} P(\bar{v}', \bar{h}') \phi_i(\bar{v}', \bar{h}')$$

The kernelized *Perceptron*

Liviu Ciortuz, following

CMU, 2013 spring, A. Smola, B. Poczos, HW2, pr. 2.d

Stanford, 2016 fall, A. Ng, J. Duchi, HW2, pr. 2

MIT, 2006 fall, Tommy Jaakkola, HW2, pr. 3

Majoritatea clasificatorilor liniari pot fi kernel-izaţi, în vederea clasificării de date care sunt neseparabile liniar. Aici vom elabora varianta kernel-izată [dual㠹] a algoritmului *Perceptron*.
Pseudo-codul variantei simple, ne-kernel-izate a algoritmului *Perceptron* este prezentat alăturat. (Operatorul $\cdot$ reprezintă produsul scalar.)

```
initialize w̄ ← 0̄
for i = 1, ..., n do
    if y_i (w̄ · x̄_i) ≤ 0 then
        w̄ ← w̄ + y_i x̄_i
    end if
end for
```

Ca input, *Perceptron*-ul kernel-izat va lua secvenţa de instanţe etichetate $\{x_i, y_i\}_{i=1}^n$ cu $x_i \in \mathbb{R}^d$ şi $y_i \in \{-1, 1\}$, precum şi funcţia-nucleu $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^m$, cu proprietatea că există $m > d$ şi o funcţie („mapare“) $\phi : \mathbb{R}^d \to \mathbb{R}^m$ astfel încât $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ pentru orice x_i, x_j. (Din punct de vedere practic, este necesar ca funcţia K să poată fi calculată în mod eficient.)

***Perceptron*-ul kernel-izat va lucra nu cu instanţele x_i, ci cu imaginile lor, $\phi(x_i)$, şi va încerca să găsească pentru acestea un separator liniar în spaţiul $\mathbb{R}^m$.**

Arătaţi cum anume se scrie regula de actualizare a coeficienţilor $\alpha_1, \ldots, \alpha_n$ la procesarea unui nou exemplu (x_i, y_i) de către algoritmul *Perceptron* kernelizat, şi cum se face predicţia ($y = 1$ sau -1) pentru o instanţă nouă x.

Răspuns

În spaţiul $\mathbb{R}^m$ (care se mai numeşte, în contextul kernel-izării, *spaţiul de trăsături*), regula de actualizare a ponderilor w se scrie astfel:

$$w^{(i)} \leftarrow w^{(i-1)} + y_i w^{(i-1)} \cdot \phi(x_i) \text{ dacă } y_i w^{(i-1)} \cdot \phi(x_i) \leq 0,$$

unde operatorul $\cdot$ desemnează produsul scalar.

În consecinţă, întrucât facem iniţializarea $w^{(0)} = \bar{0}$, vectorul $w \in \mathbb{R}^m$ va fi întotdeauna o combinaţie liniară de vectori de trăsături, $\phi(x_i)$. Aceasta înseamnă că există coeficienţii $\alpha_i \in \mathbb{R}$ astfel încât $w^{(i)} = \sum_{l=1}^{i} \alpha_l \phi(x_l)$ după procesarea primelor i instanţe de antrenament. Aşadar, vectorul $w^{(i)}$ poate fi reprexentat în mod compact prin coeficienţii α_l (cu $l = 1, \ldots, i$) din această combinaţie liniară.

În particular, valoarea iniţială $w^{(0)}$ corespunde cazului când suma nu conţine niciun termen (adică avem o listă vidă de coeficienţi α_l).

Arătăm acum că putem actualiza în mod eficient coeficienţii α_i.

La iteraţia i trebuie să calculăm produsul scalar $w^{(i-1)} \cdot \phi(x_i)$. Întrucât produsul scalar în spaţiul de trăsături $\mathbb{R}^m$ este o operaţie costisitoare atunci când m este mare, vom ţine cont că

$$w^{(i-1)} \cdot x_i = \left(\sum_{l=1}^{i-1} \alpha_l \phi(x_l) \right) \cdot \phi(x_i) = \sum_{l=1}^{i-1} \alpha_l (\phi(x_l) \cdot \phi(x_i)) = \sum_{l=1}^{i-1} \alpha_l K(x_l, x_i),$$

ceea ce înseamnă că aceşti coeficienţi se pot calcula într-adevăr în mod eficient.

În mod similar, se poate face în mod eficient *predicţia* clasei/etichetei pentru o instanţă nouă (de test) $x \in \mathbb{R}^d$:

$$w^{(i)} \cdot \phi(x) = \sum_{l=1}^{i} \beta_l \phi(x_l) \cdot \phi(x) = \sum_{l=1}^{i} \beta_l K(x_l, x).$$

Sumarizând, algoritmul pentru antrenarea *Perceptron*-ului kernel-izat se poate scrie în pseudo-cod astfel:

initialize $\alpha_i = 0$ for $i = 1, \ldots, n$;
for $i = 1, \ldots, n$ do
  if $y_i \sum_{l=1}^{i-1} \alpha_l K(x_l, x_i) \leq 0$ then
    $\alpha_i \leftarrow y_i$
  end if
end for

Observaţie importantă: **Se poate arăta relativ uşor că raţionamentele (şi rezultatele) din acest exerciţiu se pot extinde şi la cazul perceptronului care foloseşte funcţie de activare de tip prag (în particular, funcţia $sign$), rată de învăţare oarecare $\eta > 0$ — dar face iniţializarea vectorului de ponderi $w \in \mathbb{R}^d$ cu $\bar{0}$, ceea ce este echivalent cu $\alpha_i = 0$, pentru $i = 1, \ldots, n$ — şi care eventual parcurge de mai multe ori setul de date de antrenament. În acest caz, regula de actualizare a perceptronului kernel-izat este de forma**

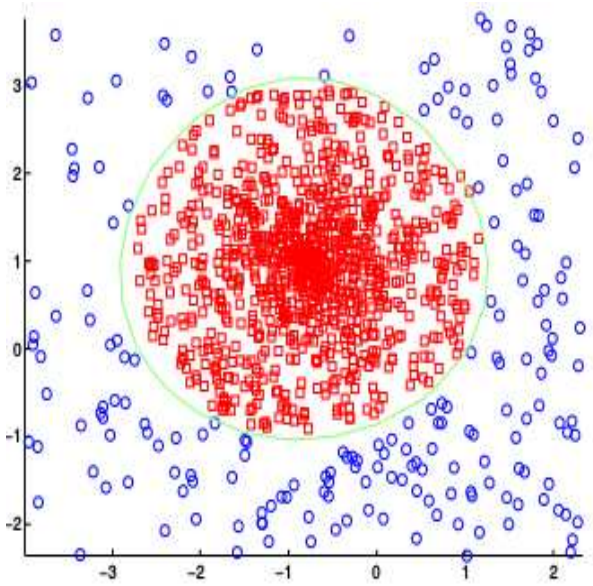
$$\alpha_i \leftarrow \alpha_i + \eta(y_i - sign(w^{(i-1)} \cdot \phi(x_i)) \text{ dacă } y_i sign(w^{(i-1)} \cdot \phi(x_i)) \leq 0$$

unde, exact ca şi mai sus, $w^{(i-1)} \cdot \phi(x_i) = \left(\sum_{l=1}^{i-1} \alpha_l \phi(x_l)\right) \cdot \phi(x_i) = \sum_{l=1}^{i} \alpha_l K(x_l, x_i)$.

kernel = 'exp(-transpose$(x_i - x_j)(x_i - x_j)/(2s^2)$)'; $s = 3$;
function α=TRAIN_KERNEL_PERCEPTRON(X, y, kernel)
    n, d = size(X);
    K = [ ];
    for $i = 1 : n$
        $x_i = X(i, :)'$;
        for $j = 1 : n$
            $x_j = X(j, :)'$;
            $K(i, j)$ = eval(kernel);
        end
    end

    α = zeros(n, 1);
    mistakes = 1;
    while mistakes > 0
        mistakes = 0;
        for $i = 1 : n$
            if $\alpha' K(:, i) y(i) \leq 0$
                $\alpha(i) = \alpha(i) + y(i)$;
                mistakes = mistakes + 1;
            end
        end
    end

Source: MIT, 2006 fall, Tommy Jaakkola, HW2, pr. 3



function f=DISCRIMINANT_FUNCTION(α, X, kernel, X_{test})
    n, d = size(X);
    K = [ ];
    for $i = 1 : n$
        $x_i = X(i, :)'$;
        $x_j = X_{\text{test}}$;
        $K(i)$ = eval(kernel);
    end
    $f = \alpha' K$;