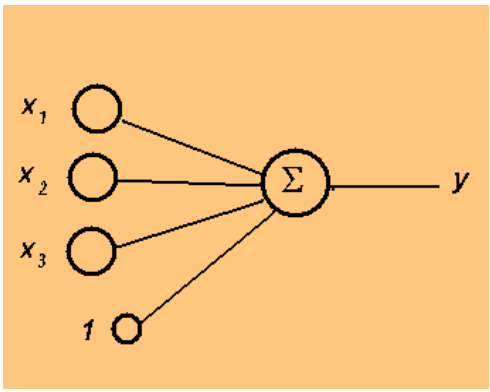
plik w4.tex, 30 października 2008, popr. 11.11.2010

4 Perceptron prosty, wyznaczanie wag

4.1 Perceptron prosty, budowa i znaczenie wag

Perceptron prosty ma budowę: d $\rightarrow$ 1 $\rightarrow$ 1, tzn. d neuronów wejściowych, 1 neuron ukryty i 1 wyjście. Aktywacja neuronu ukrytego jest bezpośrednio wyprowadzana na wyjście. Rysunek 4.1 przedstawia schemat takiego perceptronu {fig4:1, *pppngcopperc.eps*}.



Rysunek 4.1: Interpretacja wag perceptronu prostego. Neuron centralny sumuje dane wejściowe z wagami, przetwarza przez funkcję aktywacji i przekazuje wynik na wyjście

Wektor wag składa się z *właściwych wag* $w_1, w_2, \ldots, w_d$ z jakimi są sumowane składowe wektora wejściowego $x_1, x_2, ..., x_d$, oraz z tak zwanego *obciążenia* (ang. *bias*) oznaczanego przez różnych autorów rozmaitymi symbolami: demonstracje nnet używają symbolu b (od bias), Rojas używa symbolu θ, tego oznaczenia używali McCulloch i Pitts którzy sformułowali pierwszy model sztucznego neuronu w r. 1943; w literaturze coraz częściej używa się oznaczenia w_0, co pozwala zapisać jednorodnie wszystkie wagi właściwe oraz obciążenie w postaci jednego wektora wag

$$w_1, w_2, \ldots, w_d, w_{d+1}\ (= w_0)$$

którą to sytuację pokazuje rysunek 4.1, na którym przyjęto $w_0 = -\theta$. Wektor wag z włączonym obciążeniem nosi nazwę rozszerzonego wektora wag (ang. **extended weight vector**) [4].

Odpowiednio do rozszerzonego wektora wag definiujemy rozszerzony wektor danych (ang. **extended data vector**)

$$\mathbf{x} = (x_1, \ldots, x_d, 1)$$

Podział danych wielowymiarowych na dwie grupy danych

Niech $\mathbf{x} \in R^d$ oznacza punkt w d-wymiarowej przestrzeni, interpretowany również jako wektor $\mathbf{x}$ o d składowych: $\mathbf{x} = (x_1, w_2, \ldots, x_d)$. Generalnie, równanie $w_1 * x_1 + w_2 * x_2 + w_d * x_d + w_0$ przy ustalonych wartościach $w_1, \ldots w_d, w_0$ przedstawia równanie hiperpłaszczyzny rozdzielającą całą przestrzeń R^d na dwie części: dodatnią (tam gdzie wartość funkcji $z = w_1 * x_1 + w_2 * x_2 + w_d * x_d + w_0$ jest dodatnia) oraz ujemną (tam gdzie wartość funkcji

$z = w_1 * x_1 + w_2 * x_2 + w_d * x_d + w_0$ jest ujemna). Punkty $\mathbf{x}$ spełniające warunek $z = 0$ leżą na hyperpłaszczyźnie, i arbitralnie mogą być zaliczane do jednej z dwóch cześci przestrzeni. Tym samym otrzymaliśmy podział punktów przestrzeni na dwie kategorie (na dwie grupy), czyli **klasyfikację punktów danych**.

W przypadku liniowego zadania dyskryminacyjnego mamy zadane grupy 1 i grupy 2 w postaci punktów indywidualnych danych. Chcemy zidentyfikować punkty należące do grupy oznaczonej umownie grupą 1 – na podstawie liniowej funkcji dyskryminacyjnej. Aby to móc zrobić, należy wyznaczyć współczynniki $w_1, \ldots, w_d, w_0$ w ten sposób, aby punkty grupy 1 wyznaczone na podstawie funkcji liniowej {wzór flin}

$$z = w_1 * x_1 + w_2 * x_2 + w_d * x_d + w_0 \tag{4.1}$$

wykazały wartości dodatnie, natomiast punkty grupy 2 – wartości ujemne. W przypadku używania perceptronu prostego, jako wartości docelowe możemy postawić np. wartości t=1 (dla grupy 1) i t=2 dla grupy 2.

Uwaga. Różne pakiety w różny sposób interpretują wartość w_0, szczególnie jeśli chodzi o znak. Przykładowo pakiet Netlab oblicza wartości b ktże podstawiamy bezpośrednio do wzoru 4.1.

Wnioskowania tego typu noszą nazwę **threshold computations** [4]. Można je sprowadzić do wyznaczania iloczynu skalarnego $\mathbf{w} \cdot \mathbf{x}$ i sprawdzanie nierówności

$$\mathbf{w} \cdot \mathbf{x} \geq \theta \qquad \text{lub} \quad \mathbf{w} \cdot \mathbf{x} \geq 0,$$

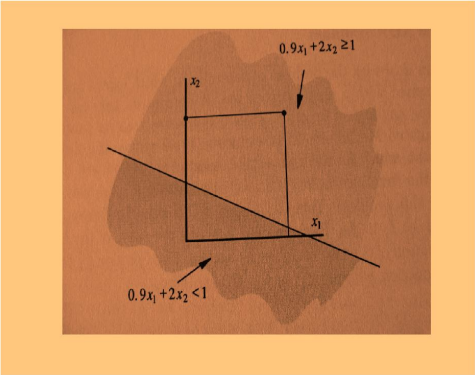
w zależności od tego, czy używamy zwykłych czy też roszerzonych wektorów wag i wektorów danych.

Uwaga. Różne pakiety obliczają w różny sposób wartość w_0, szczególnie jeśli chodzi znak. Przykładowo pakiet Netlab oblicza wartości b ktże podstawiamy bezpośrednio do wzoru 4.1. **W przypadku d=2** otrzymujemy równanie $w1*x+w2*y+w_0 = 0$ określające prostą rozgraniczającą płaszczyznę (x,y) na 2 części. Ilustruje to rysunek 4.2, na którym pokazano podział płaszczyzny (x,y) określony dwoma nierówności: $0.9x_1 * 2x_2 \geq 1$, oraz $0.9x_1 * 2x_2 < 1$.

Wypisane równanie określa funkcję liniową z zależną od zmiennych (x,y)

$$z = w1 * x + w2 * y + w_0$$

Wartość dodatnia funkcji z określa domenę pierwszej grupy (w demonstracjach nnet Matlaba oznaczanych kolorem czarnym), a wartość ujemna z – domenę grupy drugiej (punktów białych w demonstracjach nnet Matlaba). Zobacz demonstracje nt. wyznaczania granicy decyzyjnej, ang, *decision boundary*, moduł nnd4db (wykład 3), oraz uczenie się neuronu za pomocą reguły perceptronu, ang. Perceptron rule, moduł nnd4pr. Działanie ostatniego modułu jest zilustrowane na rysunku 4.2 poniżej {fig4:2, *r34copper.jpg*}.



Rysunek 4.2: Wagi perceptronu prostego w przypadku $d = 2$ wyznaczają prostą która rozdziela całą płaszczyznę na 2 części. Skopiowany rysunek 3.4 Rojasa [4]

4.2 Zagadnienie separabilności dwóch grup

Definicja separabilności (separability) i absolutnej separabilności (absolute separability) – patrz Rojas [4], Rozdział 4.

Definition. Two sets A and B of points in an d-dimensional space are called **absolutely linearly separable** if $d + 1$ real numbers $w_1, \ldots, w_{d+1}$ exist such that every point $(x_1, \ldots, x_d \in A)$ satisfies $\sum_{i=1}^{d} w_i x_i > w_{d+1}$ and every point $(x_1, \ldots, x_d \in B)$ satisfies $\sum_{i=1}^{d} w_i x_i < w_{d+1}$.

Definicja: Zwykła separabilność. Mamy dwa skończone zbiory A i B leżące w R^d. Zbiory te są liniowo separabilne, jeśli punkty $(\mathbf{x})$ z A spełniają nierówność $\sum_{i=1}^{d} w_i x_i \geq w_{d+1}$, a punkty należące do B spełniają nierówność $\sum_{i=1}^{d} w_i x_i < w_{d+1}$.

Twierdzenie. Two finite sets of points, A and B, in d-dimensional space which are lineraly separable, are also absolutely linearly separable.

Definicja. Równość $\sum_{i=1}^{d} w_i x_i - w_{d+1}$ definiuje hiperpłaszczyznę rozdzielającą całą przestrzeń danych na dwie części: dodatnią i ujemną półpłaszczyznę (ang. *positive and negative halfspace*).

Zauważmy, że wyraz w_{d+1} jest tutaj brany ze znakiem ujemnym, a nie ze znakiem dodatnim, jak to było wcześniej przez nas wprowadzone w sekcji 4.1.

4.3 Wyznaczanie wag – metody bezpośrednie

4.3.1 Rozwiązywanie układu równań liniowych

Domyślnie przyjmujemy, że funkcją aktywacji perceptronu prostego jest funkcja tożsamościowa, czyli 'linear'. Dla danych rozszerzonych zawartych w macierzy $\mathbf{X}$ o rozmiarze $[n \times (d + 1)]$, i wartościach docelowych $\mathbf{t} = (t_1, \ldots, t_n)^T$, wagi perceptronu prostego powinny spełniać (z możliwie małymi odchyłkami) układ równań:

$$\mathbf{Xw} = \mathbf{t},$$

gdzie $\mathbf{w} = (w_1, \ldots, w_d, 1)^T$ oznacza wektor kolumnowy rozszerzonych wag. Metoda najmniejszych kwadratów błędów prowadzi do kwadratowej powierzchni błedu E której minimum można otrzymać za pomocą funkcji mldivide (left divide, symbol funcji: \ ):

$$\mathbf{w} = \mathbf{X}\backslash\mathbf{t}.$$

4.3.2 Reguła perceptronu – perceptron rule

Jest to metoda specyficzna dla perceptronu prostego. Algorytm jest następujący (przepisany z Rojasa [4], str 84–89)

```
given sets P (positives, group 1) and N (negatives, group 2)
start: generate randomly weight vector w
       set t:=0
test:  evaluate test criterion, if true then STOP
       select x from [P; N] % combined data matrix
       if (x from P) and (w_t * x)>0, go to test
       if (x from P) and (w_t * x)<=0, go to add
       if (x from N) and (w_t * x)<0, go to test
       if (x from N) and (w_t * x)>=0, go to substract
add:   set w_(t+1):=w_t+x, set t:=t+1, go to test
substract: set w_(t+1):=w_t-x, set t:=t+1, go to test
```

Można wykazać, że dla zbiorów liniowo separabilnych algorytm *perceptron rule* jest zbieżny w skończonej liczbie kroków. Działanie tego algorytmu jest zilustrowane w interakcyjnym module nnd4pr firmowego pakietu Matlab. Zrzut okien z jednego trenowania jest pokazany na rysunku 4.3 {fig4:3, *perc1c.eps, perc2c.eps, perc3c.eps, perc4c.eps* }.

Rysunek ten pokazuje 4 etapy działania modułu przy opcji Learn pokazującej rezultaty trenowania po każdym punkcie przedstawianym sieci. Wybrano opcję, że wyznaczana prosta dyskryminująca przechodzi przez początek układu, czyli że $w_0 \equiv 0$. Grupą docelową jest grupa czarnych. Niewłaściwy stan elementów jest zaznaczony kolorem czerwonym.

Opis czterech paneli widocznych na rysunku 4.3.

(a) Wygenerowano 3 punkty: $P1 = (1, 2)$, $P2 = (0, -1)$ oraz $P3 = (-1, 2)$; punkty te są umieszczone w kolejności wskazówek zegara, począwwszy od godz. 12-tej.
Wygenerowano również losowe wagi w

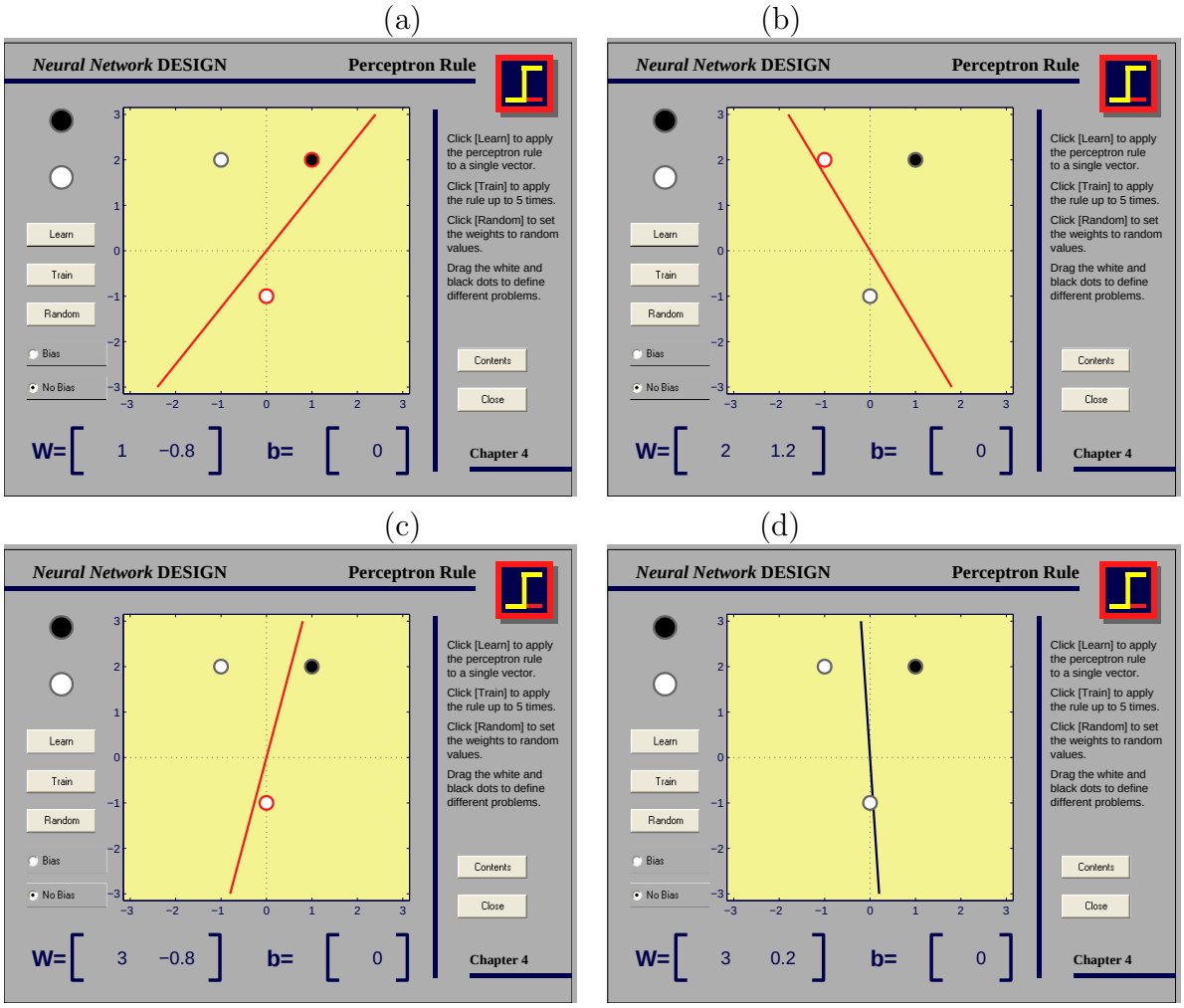
$$[\ 1 \quad -0.8]$$

Można zauważyć, że punkty $P1$ i $P2$ są źle sklasyfikowane, wektor wag pokazuje zły kierunek, wobec tego prosta jest zła. Sieć nie jest wystarczająco wytrenowana, aktualny wektor wag nie potrafi dobrze rozpoznać 'czarnych' Wobec tego dodajemy wektor P1 do wektora wag $\mathbf{w}$.

(b) Po przedstawieniu sieci punktu $P1$ otrzymano wagi

$$[\ 2 \quad 1.2]$$

$P1$ i $P2$ zostały sklasyfikowane dobrze, ale punkt $P3$ źle.
Wektor wag wskazywał dobry kierunek, ale prosta była niedobra. Sieć została przetrenowana, bo rozpoznaje obcy punkt jako punkt czarny. decyzja: punkt $P3$ należy odjąć od w, czyli obliczyć w-P3.



Rysunek 4.3: Trenowanie perceptronu metodą *perceptron rule*. Zrzut ekranów podczas działania modułu nnd4pr Matlaba. Należało wyznaczyć prostą, rozdzielającą grupy czarnych i białych punktów, przy dodatkowch warunkach (i) żeby wektor wag $[w_1, w_2]$ wskazywał na grupę 'czarnych', oraz (ii) żeby wyznaczona prosta przechodziła przez punkt (0,0)

(c) Teraz przedstawiono sieci punkt $P3$, poczym otrzymano wagi

$$[\,3 \quad -0.8]$$

Okazało się, że teraz punkty $P1$ i $P3$ są klasyfikowane dobrze, ale Punkt $P2$ jest klasyfikowany do niewłaściwej grupy. Wektor wag wskazuje dobry kierunek, ale prosta jest niedobra. Należy odjąć od w zły punkt $P2$.

(d) Teraz przedstawiono sieci punkt $P2$. Wagi się zmieniły na

$$[\,3 \quad 0.2]$$

Okazało się, że teraz wszystkie punkty są klasyfikowane dobrze, wektor wag wskazuje dobry kierunek, i prosta jest też dobra. Żaden element rysunku nie jest zaznaczony na czerwono. W tym stanie algorytm się zatrzymał.

4.3.3 Algorytm kieszonkowy – The pocket algorithm

Algorytm ten jest opisywany w kontekscie uczenia perceptronu prostego za pomocą reguły perceptronu; można go stosować również przy innych regułach uczenia.

Podamy tu dwie wersje tego algorytmu opisane w książkach Rojas [4] oraz Duda i in. [2]; wersja z [2] jest zaimplementowana w pakiecie Matlab Classification Toolbox autorstwa D. Stark i Elad Yom-Tov [3].

Algorytm kieszonkowy według książki Rojasa [4]

Algorytm ten jest opisany w książce Rojas [4], str 90–91, z powołaniem się na pracę Gallant, "Perceptron-based learning algorithms", IEEE Transactions on Neuron Networks, Vol. 1, No. 2, pp. 179–191. Do pracy tej nie mam dostępu. Dalszy opis jest oparty wyłącznie na źródle [4].

start: Generate randomly weight vector $\mathbf{w}$
Define a 'stored' weight vector $\mathbf{w}_S = \mathbf{w}$.
Set h_S, the history of succcessful classification by $\mathbf{w}_S$, to zero.
iterate: Update $\mathbf{w}$ using a single iteration of the perceptron learning algorithm.
Keep track of the number h of consecutively successfully tested vectors.
If at any moment $h > h_S$ holds, substitute $\mathbf{w}_S$ with $\mathbf{w}$ and h_S with h.
Continue iterating.

Zmienna h określa długość sekwencji składającej się z dobrze sklasyfikowanych elementów próbki uczącej. Pamiętamy parę $h_S, \mathbf{w}_S$. Algorytm może okazyjnie zamienić $\mathbf{w}_S$ na gorszy, jednak prawdopodobieństwo stwo tego zdarzenia maleje ze wzrostem t. Gallant (patrz wyżej) pokazał, że jeżeli zbiór trenujący jest skończony, a wartości wag i wektorów danych są rozsądne (ang.: are rational), to prezentowany algorytm zbiega się do optymalnego rozwiązania z prawdopodobieństem równym 1.

Algorytm kieszonkowy według książki Stork, Yom-Tov [3]

Cytuję:

> The Pocket algorithm is a simple modification over the Perceptron algorithm. The improvement is that updates to the weight vector are retained only if they perform better on a random sample of the data. In the current MATLAB implementation, the weight vector is trained for 10 iterations. If the new weight vector succeeded in classifying more pattern before it misclassified a pattern compared to the old weight vector, the new vector replaces the old vector. The procedure is repeated until convergence or maximal number of iterations is reached.

Wzmiankowana implementacja w Matlabie jest bardzo szczątkowa. W wersji z 2005 r należy poprawić algorytm uczenia wg reguły perceptronu oraz sposób liczenia długości serii poprawnie sklasyfikowanych elementów (w wersji z 2005 r liczy się długość serii elementów zaklasyfikowanych do pierwszej grupy, czyli prognozowanych jako jedynki).

Autorzy Duda et al. radzą przetestować początkowe wagi trenowanego perceptronu na wszystkich danych (i w ten sposób określić początkowe h_S). Testowanie wag perceptrunu po wykonaniu kolejnych iteracji może być wykonywane na losowej próbce danych.

4.4 Wyznaczanie wag za pomocą gradientów funkcji błędu

Na wykładzie 3 pokazaliśmy, że iteracyjne poprawianie wag może następować według wzoru:

$$\mathbf{w}^{(s+1)} = \mathbf{w}^{(s)} + \Delta\mathbf{w},$$

gdzie $\Delta\mathbf{w} = -\eta\{\partial E/\partial\mathbf{w}\}|_{w=w^{(s)}}$; $\mathbf{w}^{(s)}$ jest *starym* a $\mathbf{w}^{(s+1)}$ *nowym* wektorem wag; natomiast η jest przyjętym współczynnikiem uczenia.

Jest ewidentne, że gradient funkcji błędu zależy w sposób istotny od zastosowanej funkcji aktywacji.

Przedstawione wyżej 'poprawianie' wektora wag można zastosować bezpośrednio do wyznaczania wag w perceptronie prostym. Przy znanych funkcjach aktywacji (linear, logistic, tanh) algorytm implementuje się w bardzo prosty sposób (lista zadań nr. 3). Poniżej przedstawiamy wyprowadzone explicite wzory dla przypadków, gdy błąd jest definiowany za pomocą kryterium najmniejszych kwadratów (LSE, Least Squares Error) oraz entropii krzyżowej wynikającej z zasady maximum funkcji wiarygodności.

4.4.1 Błąd LSE czyli metodą najmniejszych kwadratów

Niech t oznacza wartość docelową dla wektora $\mathbf{x}$, a f funkcję aktywacji perceptronu prostego. Przedstawiamy sieci wektor $\mathbf{x}$ celem nauczenia się danych, w wyniku czego sieć poprawiła swoje wagi i wyprowadziła na wyjście wynik sieci y. Błąd sieci metodą najmniejszych kwadratów (ang. LSE, Least Square Error) w tym momencie jest określany jako

$$E = \frac{1}{2}(t-y)^2, \qquad \text{gdzie } \; y = f(\mathbf{x}^T\mathbf{w} + w_0) = f(a), \quad \text{natomiast } \; a = \mathbf{x}^T\mathbf{w} + w_0.$$

Obliczmy teraz gradient funkcji błędu E względem argumentu $\mathbf{w}$ {wzór gradientLSE}

$$\frac{\partial E}{\partial \mathbf{w}} = -(t-y)\frac{\partial y}{\mathbf{w}} = -(t-y)\frac{\partial y}{\partial f}\frac{\partial f}{\partial a}\frac{\partial a}{\partial \mathbf{w}}\,. \tag{4.2}$$

Pochodna $\frac{\partial y}{\partial f}$ jest tożsamościowo równa 1 (ponieważ $y \equiv f$).
Pochodna $\partial f/\partial a$ zależy od przyjętej funkcji aktywacji; oznaczymy tę pochodną ogólnie $\partial f/\partial a = f'_a$. Pochodna te będzie wyznaczana dla argumentu a (aktywacja wejściowa neuronu), co zapiszemy jako $(\partial f/\partial a)|_a = f'_a(a)$.
Pochodna formy liniowej $\mathbf{w}^T\mathbf{x}$ względem $\mathbf{w}$ wynosi $\frac{\partial(\mathbf{w}^T\mathbf{x})}{\partial\mathbf{w}} = \frac{\partial(\mathbf{x}^T\mathbf{w})}{\partial\mathbf{w}} = \mathbf{x}$.

Podstawiając to wszystko do wzoru 4.2 otrzymujemy {wzór gradientLSE1}

$$\frac{\partial E}{\partial \mathbf{w}} = -(t-y)\frac{\partial y}{\mathbf{w}} = -(t-y)\frac{\partial y}{\partial \mathbf{f}}\frac{\partial f}{\partial a}\frac{\partial a}{\partial \mathbf{w}} = -\underbrace{(t-y)\,f'_a(a)}_{delta}\mathbf{x}. \tag{4.3}$$

W podobny sposób otrzymamy gradient funkcji E obliczany dla wagi w_0 jako {wzór gradientLSE1a}

$$\frac{\partial E}{\partial w_0} = -(t-y)\frac{\partial y}{\partial w_0} = -(t-y)\frac{\partial y}{\partial f}\,\frac{\partial f}{\partial a}\,\frac{\partial a}{\partial w_0} = -\underbrace{(t-y)\,f'_a(a)}_{delta}\cdot 1. \tag{4.4}$$

Stosując **notację rozszerzoną**, tj. $\mathbf{x} = (x_1, \ldots, x_d, 1)^T$, $\mathbf{w} = (w_1, \ldots, w_d, 1)$, możemy wzory (4.3) i (4.4) połączyć w jeden otrzymując { wzór GRADIENTlse}

$$\frac{\partial E}{\partial \mathbf{w}} = -\underbrace{(t-y)\ f'_a(a)}_{delta}\ \mathbf{x}. \tag{4.5}$$

Przyjmując jako **funkcję aktywacji** funkcję **liniową** ('linear') mamy $f'_a(a) \equiv 1$, wobec czego wzór (4.5) redukuje się do następującego (przypominamy, w notacji rozszerzonej):

$$\frac{\partial E}{\partial \mathbf{w}} = -\underbrace{(t-y)}_{delta}\ \mathbf{x}.$$

Przyjmując jako funkcję aktywacji **funkcję logistyczną** ('logistic') mamy: $f'_a(a) = y(1-y)$, wobec czego wzór (4.5) redukuje się do następującego (przypominamy, w notacji rozszerzonej):

$$\frac{\partial E}{\partial \mathbf{w}} = -\underbrace{\overbrace{(t-y)}^{error}\ \overbrace{y(1-y)}^{slope=f'_a}}_{delta}\ \cdot\mathbf{x}.$$

Widzimy, że w obu przypadkach gradient jest proporcjonalny do wektora $\mathbf{x}$ oraz do obserwowanej różnicy $(t-y)$.

4.4.2 Błąd definowany jako minimum entropii krzyżowej

Niech t oznacza zmienną losową binarną, przyjmującą wartości 0 i 1 z prawdopodobieństwem p i $1-p$ odpowiednio, $0 \leq p \leq 1$. Wtedy wiarygodność wystąpienia wartości t w wykonanym doświadczeniu zapisuje się jako

$$L = p^t(1-p)^{1-t}$$

. W przypadku klasyfikacji do dwóch grup oraz stosowania logistycznej funkcji aktywacji wynik sieci (y) możemy przedstawić jako modelowe prawdopodobieństwo a posteriori, że zmienna t przyjmie wartość 1 pod warunkiem, że zmienne objaśniające przyjęły wartości $x_1, \ldots, x_d$. Sposób uzależnienia prawdopodobieństwa y($\mathbf{x}$) jest podany za pomocą funkcji logistycznej $y = f_{logi}(a)$, gdzie $a = \mathbf{w}^T\mathbf{x}$ w notacji wektorów rozszerzonych (zobacz Bishop [1]). W tej sytuacji funkcję wiarygodności możemy zapisać jako (używamy notacji rozszerzonej)

$$L = y^t(1-y)^{1-t}, \quad 0 < y < 1, \quad \text{gdzie} \quad y = f_{logi}(a), \quad a = \mathbf{w}^T\mathbf{x}$$

Wagi $\mathbf{w}$, czyli współczynniki kombinacji liniowej tworzącej aktywność neuronu, należy wyznaczyć tak, aba zmaksymizować wiarygodność zdarzenia. Funkcja błędu sieci, nazywana powierzchnią błędu (E) jest określana jako

$$E = -logL = -[t\,log(y) + (1-t)\,log(1-y)]$$

a wagi $\mathbf{w}$ należy wybrać tak, aby znaleźć minimum funkcji błędu (co jest równoważne znajdywaniu maksimum funkcji wiarygodności, ang. ML, maximum likelihood).

Postępując podobnie, jak przy minimalizacji funkcji błędu metodą LSE otrzymujemy **przy logistycznej funkcji aktywacji** i wektorów **w** oraz **x** zapisanych w **rozszerzonej notacji**

$$\frac{\partial E}{\partial \mathbf{w}} = -\Big[\ \frac{t}{y} - \frac{1-t}{1-y}\ \Big]\ \frac{\partial y}{\partial f}\ \frac{\partial f}{\partial a}\frac{\partial a}{\partial \mathbf{w}} = -\Big[\ \frac{t-y}{y(1-y)}\ \Big]\ \cdot 1 \cdot y(1-y)\cdot \mathbf{x}$$

skąd ostatecznie wynika, że przy logistycznej funkcji aktywacji

$$\frac{\partial E}{\partial \mathbf{w}} = -\underbrace{(t-y)}_{delta}\cdot \mathbf{x} \tag{4.6}$$

4.4.3 Metoda GLM, Generalized Linear Model

Jest to swoista metoda rozwinięta na gruncie statystyki matematycznej [5]. Model GLM dopuszcza różnego rodzaju linki imitujące funkcje aktywacji. W przypadku linku wyrażonego za pomocą funkcji logistycznej model GLM bardzo przypomina perceptron prosty.

Pakiet NETLAB oferuje bardzo efektywną sieć GLM opartą na metodzie uogólnionych modeli liniowych. Sieć ta może mieć kilka neuronów w warstwie ukrytej, i odpowiednio do tego kilka wyjść.

4.5 Przykłady zastosowań do danych Mnist i Faces

Dane Seunga: seungMNIST, seungFaces – zobacz lista zadań nr. 3 i 4.
Seung (profesor MIT) napisał efektywny skrypt matlabowski pozwalający różnicować między zbiorami graficznymi zawierającymi obrazy cyfr $0, 1, \ldots 9$, oraz między 'twarzami' i 'nie-twarzami'. Skrypt ten implementuje model prostego perceptronu wykorzystując wzór (4.5) podstawiając do niego funkcję logistyczną. Na zajęciach laboratoryjnych możemy posługiwać się funkcją seung lub seung1 (ta ostatnia jest bardziej dostosowana do oznaczeń używanych na wykładzie). Funkcje te mają postać:

```
[y,b]=seung(train,trainLabs,options);
```

```
[y,b]=seung1(train,trainLabs,options);
```

Tablica train zawiera dane, natomiast wektor trainLabs identyfikatory kolejnych kolumn (obrazków) z których wnioskuje się, co dany obrazek przedstawia. *Dla cyferek* (dane Mnist) są to wartości $0, 1, \ldots, 9$; wobec czego należy pod zmienną options(11) podstawić wartość cyferki, którą chce się rozpoznać. W przypadku rozpoznawania *twarzy i nie-twarzy* trainLabs przyjmują wartość 1 (dla twarzy), oraz 0 (dla nie-twarzy). Chcąc rozpoznać 'twarze', należy zmiennej options(11) nadać wartość równą jedności. Zmienna options(14) zawiera liczbę iteracji do trenowania perceptronu. Jeżeli zmienna ta ma nadaną wartość 0, to automatycznie zostanie podstawiona wartość 5000.

Należy wyraźnie zaznaczyć, że układ danych seungMnist i seungFaces jest nietypowy, gdyż poszczególne obrazki (stanowiące 'osobniki', czyli wektory danych) są umieszczone kolumnami. Dane zapamiętane w formacie uint8 przed obliczeniami numerycznymi wymagają konwersji na format double. Służy temu funkcja matlabowska o tej samej nazwie. Ponadto zaleca się, aby dane były przeskalowane w ten sposób, aby zawierały się w przedziale [0, 1]. Jest to wykonywane we wnętrzu funkcji seung i seung1. Wagi obliczone przez sieć są ważne dla danych przeskalowanych do odcinka [0,1].

Jak widać z wyników, funkcje seung (lub seung1) spełniają rolę procedury trenującej sieć, która ma w tym przypadku architekturę perceptronu prostego. Na wyjściu sieci otrzymujemy wagi w oraz bias b, minimalizujące błąd E zdefiniowany wcześniej w tym rozdziale. Potrzebna funkcja logistyczna jest zaimplementowana w jednym wierszu skryptu matlabowskiego wykorzystując możliwość inline:

```
f=inline('1.0/(1+exp(-x))');
```

Funkcję tę można wykopiować i użyć do obliczania wyniku sieci, tj. wektora y po wytrenowaniu sieci. Dane, dla których chcemy obliczyć wyniki, powinny być formatu *double* i sprowadzone do przedziału [0,1]:

```
y=f( (Ttrain'w + b),
```

gdzie Ttrain=(double(train))/255. Zmienna wynikowa y jest zymienną wektorową kolumnową; zawiera ona liczby z przedziału [0, 1] określające prawdopodobieństwa, że obrazki zawartych w pliku trenującym Ttrain należą do grupy określonej wartością zmiennej options(11).

Aby sprawdzić, czy otrzymane wyniki nie są artefaktami, należy sprawdzić możliwości otrzymanej funkcji dyskryminacyjnej na zbiorze testowym składającym się z takich danych, które nie brały udziału w trenowaniu sieci.

Literatura

[1] Ch. M. Bishop, *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1996.

[2] R.O. Duda, P.E. Hart, D.G. Stork: *Pattern Classification*, 2nd Edition, Wiley 2001.

[3] D.G. Stork, Elad Yom-Tov, *Computer Manual in MATLAB to accompany Pattern Classification*. Wiley Interscience 2004.

[4] Raul Rojas, *Neural Networks, A systematic Introduction*. Springer 1996.
Rozdział 3: Weighted Networks – The Perceptron, str 55–76; Rozdział 4: Perceptron Learning, str 77–99 .

[5] Dobson A. J., 1990, *An Introduction to Generalized Linear Models*. Chapman and Hall, London, New York. 2nd edition 2002.

Spis treści