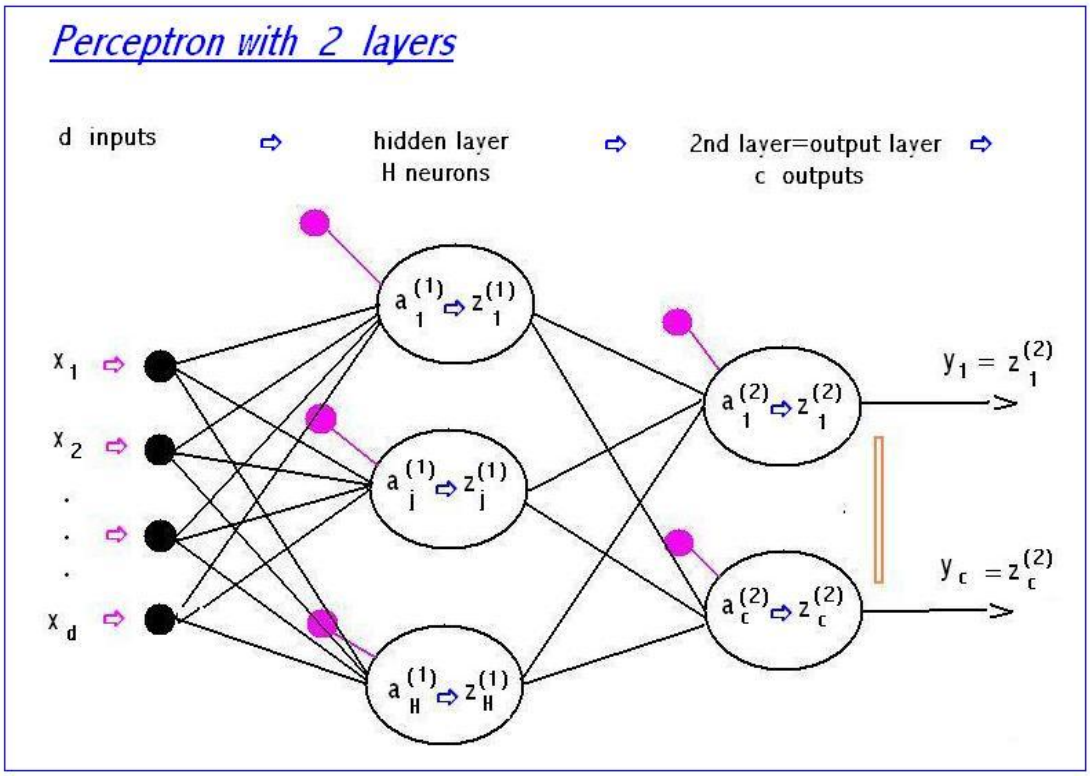
plik wk2.tex, October 28, 2008, popr 19.X.2010

2 Sieć MLP – Multilayer perceptron

Omówimy tutaj model perceptronu dwuwarstwowego i jego implementację w Netlabie. Perceptron wielowarstwowy powstaje przez nałożenie (dodanie) więcej warstw ukrytych.

2.1 Architektura sieci MLP z pakietu Netlab

Architektura sieci MLP zaimplementowanej w pakiecie Netlab jest pokazana na Rysunku 2.1.



Rysunek 2.1: *Architektura sieci MLP o d wejściach (warstwa zerowa) i 2 warstwach ukrytych, z których druga stanowi warstwę wyjściową. Podawanie informacji następuje tylko wprzód, tzn. od lewej (wejście) poprzez przetwarzanie najpierw przez neurony warstwy ukrytej pierwszej (o H neuronach). Wyniki przetwarzania w pierwszej warstwie są podawane jako dane wejściowe do następnej warstwy, gdzie są ponownie przetwarzane poprzez zadeklarowaną funkcję aktywacji. Wynik tego drugiego przetworzenia jest podawany na wyjście sieci.* {*mlpj.jpg*}

Oznaczenia
d – l. wejść, $\mathbf{x} = [x_1, \ldots, x_d]^T$ – wektor wejściowy, zmienne objaśniające,
c – l. wyjść, $\mathbf{y} = [y_1, \ldots, y_c]^T$ – wektor wyników, odpowiedź sieci na sygnał $\mathbf{x}$,
H – liczba neuronów pierwszej warstwy ukrytej.
$\mathbf{w}_j^{(1)}, b_j^{(1)} (j = 1, \ldots, H)$ – wagi i biasy neuronów pierwszej warstwy,
$\mathbf{w}_k^{(2)}, b_k^{(2)} (k = 1, \ldots, c)$ – wagi i biasy neuronów drugiej warstwy.

Neurony pierwszej warstwy

Każdy neuron (j) ma przypisany wektor wag $\mathbf{w}_j^{(1)}$ oraz bias $b_j^{(1)}$. Każdy neuron wyznacza swoją aktywację wejściową według wzoru:
$a_j^{(1)} = \sum_{i=1}^{d} w_{ji}^{(1)} x_i + b_j^{(1)}, \quad j = 1, \ldots, H.$
Otrzymane w ten sposób *aktywacje wejściowe* są przetwarzane przez funkcję aktywacji f_1. Powstaje wtedy wynik: $z_j^{(1)} = f_1(a_j^{(1)})$.
Otrzymane w ten sposób wyniki $z_j^{(1)}$, $j = 1, \ldots, H$ stanowią dane wejściowe dla drugiej warstwy.

Neurony drugiej warstwy

Działanie neuronów drugiej warstwy jest podobne.

Każdy neuron (k) ma przypisany wektor wag $\mathbf{w}_k^{(2)}$ oraz bias $b_k^{(2)}$. Danymi dla neuronów drugiej warstwy są wyniki $z_j^{(1)}$ wytworzone przez 1-szą warstwę. Każdy neuron 2-giej warstwy wyznacza swoją aktywację wejściową według wzoru:
$a_k^{(2)} = \sum_{j=1}^{H} w_{kj}^{(2)} z_j^{(1)} + b_k^{(2)}, \quad k = 1, \ldots, c.$
Otrzymane w ten sposób *aktywacje wejściowe* $a_k^{(2)}$ są przetwarzane przez funkcję aktywacji f_2. Powstaje wtedy wynik $z_k^{(2)}$ wyznaczony według wzoru: $z_k^{(2)} = f_2(a_k^{(2)})$.
Otrzymane w ten sposób wartości $z_k^{(2)}$, $k = 1, \ldots, c$ stanowią wyniki, które są przekazywane na wyjście sieci.

Funkcje aktywacji w module MLP pakietu Netlab

Pierwsza warstwa ma wbudowaną na stałe **funkcję** tanh(.) czyli tangens hiperboliczny.

Druga warstwa może używać jednej z następujących funkcji aktywacji:
'linear', 'logistic', 'softmax'. Funkcję tę wybiera użytkownik.

2.2 Praca z siecią MLP: trzy podstawowe funkcje

Istotną rolę odgrywają tu trzy funkcje: mlp, mlpfwd i netopt.
Zamiast netopt możemy użyć prostszej funkcji mlptrain.

MLP. Wywoływanie podstawowych funkcji

- Tworzenie sieci:

net = mlp(n_in, n_hidden,n_out, actfn)

- Trenowanie sieci:

[net, options, varargout]= netopt(net, options, data,targets, alg);
lub [net, error] = mlptrain(net, x, t, its) korzysta z 'scg' i 'netopt'
'options' są opisane w komentarzu 'glmtrain'

- Prognoza lub klasyfikacja wyuczonej sieci:

[y, z, a] = mlpfwd(net, x);
lub tylko y = mlpfwd(net, x);

- Obliczanie błędu E

e=mlperr(net,x,t).

Omówimy teraz te funkcje.

♣ **Funkcja MLP – Tworzenie podstawowej struktury sieci neuronowej**.
Strukturę tę będziemy dalej oznaczać 'net'.
Oczywiście zamiast tej nazwy można użyć dowolnej innej, np. 'my_net' lub 'jan' :

```
net = mlp(n_in, n_hidden,n_out, actfn)
```

Znaczenie parametrów wejściowych:

n_in :	d, liczba neuronów warstwy wejściowej
n_hidden:	H, liczba neuronów pierwszej warstwy ukrytej
n_out:	c, liczba neuronów warstwy wejściowej
actfn :	rodzaj funcji aktywacji: 'linear', 'logistic', 'softmax'.

Mogą jeszcze wystąpić dalsze parametry, np. prior, parametr opcyjny, wykorzystywany przy modelach Bayesowskich, określający rozrzut inicjowanych wag.

Utworzona struktura net ma następujące pola:

type	'mlp'
n_in	d, liczba wejść (liczba cech danych)
n_hidden	H, liczba neuronów warstwy ukrytej
n_out	c, liczba wyjść sieci
nwts	liczba wszystkich współczynników wagowych i biasów
actfn	funkcja aktywacji: string 'linear', 'logistic', 'softmax'
w1	tablica wag 1-ej warstwy, wymiar $d \times H$
b1	tablica biasów 1-ej warstwy, wymiar $1 \times H$
w2	tablica wag neuronów 2-ej warstwy, wymiar $H \times c$
b2	tablica biasów neuronów 2-ej warstwy, wymiar $1 \times c$

W przypadku używania złożonego modelu Bayesowskiego, struktura mlpnet przewiduje dalsze pola zawierające parametry Bayesowskie: prior i beta. Jednak my nie będziemy zajmować się tym przypadkiem.

Struktura 'net', zainicjowana jak wyżej, zawiera przyjęte losowo wartości wag oraz biasów. Np. Wartości wag zostały zainicjowane jako $randn(d, c)/sqrt(d-1)$

Oczywiście, jeśli wagi są przypadkowe, to otrzymane wyniki będą też przypadkowe. Dlatego też utworzona sieć powinna być najpierw 'wyuczona', inaczej mówiąc, wytrenowana, za pomocą specjalnej procedury. Stanowi ją funkcja netopt lub funkcja mlptrain. Obydwie te funkcje uczą sieć rozpoznawania zadanych wzorców.

♣ **Trenowanie sieci za pomocą funkcji** mlptrain
Funkcja ta ma postać: [net, error] = mlptrain(net, x, t, its)

Funkcja ta znajduje optymalne wagi za pomocą algorytmu gradientowego ('scaled conjugate gradient'). Jest to algorytm iteracyjny. Maksymalną liczbę iteracji 'its' jest deklarowana przez użytkownika.
Znaczenie pozostałych parametrów jest omówione poniżej przy omawianiu funkcji netopt.

♣ **Trenowanie sieci za pomocą funkcji** netopt

```
[net, options, varargout]= netopt(net, options, data, targets, alg);
```

Parametry wejściowe:
options: Znaczenie takie samo, jak dla modelu GLM (zobacz 'help' dla funkcji 'GLM').

W szczególności:

options(14) oznacza liczbę iteracji,
options(1) oznacza rodzaj komunikatów drukowanych podczas wykonywania iteracji uczenia; If *options(1)* is set to 0, then only warning messages are displayed. If *options(1)* is -1, then nothing is displayed,
options(2) is a measure of the precision required for the value of the weights W at the solution,
options(3) is a measure of the precision required of the objective function at the solution. Both this and the previous condition must be satisfied for termination.

data oraz targets: Dane są przygotowywane w postaci tablic o wymiarach $n \times d$ oraz $n \times c$, gdzie n oznacza liczbę wiersz (wektorów danych, 'osobników'); d jest liczbą atrybutów ('cech') danego osobnika; c jest liczbą wyjść sieci.
Tablica 'x' zawiera dane, na podstawie których sieć ma się nauczyć rozpoznawać wartości podane w tablicy 'c'.
alg: Algorytm optymalizacyjny, używany przy minimalizacji funkcji błędu. Obecna wersja Netlaba przewiduje tu podstawienie następujących algorytmów: 'quasinew', 'scg', 'conjgrad'. Są to algorytmy gradientowe.

Zauważmy, że dla przyjętych funkcji aktywacji wzory na gradienty (czyli pochodne funkcji $y = f(a)$ względem jej argumentu a) są wyjątkowo proste:

- Dla funkcji 'linear': $f'(a) = 1$,
- Dla funkcji 'logistic': $f'(a) = f(a)(1 - f(a))$,
- Dla funkcji 'tanh': $f'(a) = (1 - f^2(a))$.

Podczas 'uczenia się' sieć poprawia iteracyjnie swoje wagi w ten sposób, aby wyniki dostarczone przez sieć (tj. znajdujące się w tablicy $\mathbf{y}$) możliwie mało różniły się od wektora wartości pożądanych (docelowych) danych w tablicy $\mathbf{t}$ ('targets').

♣ **Funkcja** mlpfwd **– praca wytrenowanej sieci w trybie odtworzeniowym.**

`[y, z, a] = mlpfwd(net, x);` lub tylko `y = mlpfwd(net, x);`

Parametr wejściowy x oznacza tu tablicę danych $\mathbf{X}$ o wymiarze $N \times d$ dla których chcemy otrzymać wyniki.

Parametr wyjściowy y oznacza tablicę wyników $\mathbf{Y}$ o wymiarze $N \times c$. Ponadto możemy otrzymać:
tablicę z o wymiarach $N \times H$ zawierającą aktywacje wyjściowe $z_{Nj}^{(1)}$ pierwszej warstwy,
tablicę a o wymiarach $N \times c$ zawierającą aktywacje wejściowe $a_{Nk}^{(2)}$ drugiej warstwy.

W dodatku do wykładu zamieszczamy fragmenty funkcji mlpfwd pokazujące, jak krótki jest algorytm zaprogramowany w Matlabie.

2.3 Proces trenowania, czego można nauczyć sieć MLP

Cytat z książki Nabney'a [3]:

> The goal of training a network is to model the underlying generator of the data in order to make the best possible predictions when new input data is presented. The most general information about the target vector $\mathbf{t}$ for inputs $\mathbf{x}$ is given by the conditional density $p(\mathbf{t}|\,\mathbf{x})$.

Zagadnienia regresyjne i dyskryminacyjne w sieci MLP

Sieć MLP może nauczyć się predykcji w nieliniowych zagadnienia regresyjnych oraz zagadnieniach klasyfikacyjnych (nazywanych również zagadnieniami dyskryminacyjnymi).

Model *nieliniowej regresji* wynikają z przyjętej funkcji aktywacji tanh dla 1-ej warstwy sieci, oraz z deklarowanej funkcji aktywacji 'linear' dla drugiej warstwy; również – z możliwości przyjęcia dowolnej liczby neuronów (H) w warstwie ukrytej. Pod tym względem sieć MLP jest znacznie ogólniejsza niż sieć Netlabowska GLM.

Modele *nieliniowej dyskryminacji* otrzymujemy przyjmujc nieliniowe funkcji aktywacji: 'logistic' lub 'softmax' dla neuronw warstwy wyjciowej; oczywiście funkcja 'tanh' transformująca aktywacje pierwszej warstwy wprowadza już nieliniowe zależności między 'wejściem' i 'wyjściem' sieci) oraz możliwości deklarowania większej liczby neuronów, które potrafiłyby uwzględnić krzywoliniową strukturę danych. Model MLP potrafi nauczyć się różnicowania grup danych, które są określone przez przynależność do dowolnych zbiorów wypukłych.

2.4 Oceny błędu sieci

Sekcja ta jest zapowiedzią tematu, który zostanie rozwinięty podczas Wykładu 3.

Błąd sieci (błąd prognozy) jest oznaczany zazwyczaj literą E (ang. Error). Błąd ten w wersji klasycznej (np. w pakiecie firmowym nnet) jest określany metodą najmniejszych kwadratów różnicy między wartościami y_i prognozowanymi przez sieć i wartościami docelowymi t_i

$$E = \sum_i (y_i - t_i)^2.$$

Pakiet NETLAB korzysta z tego sposobu tylko wtedy, gdy neurony warstwy wyjściowej korzystają z *liniowej* funkcji aktywacji.

W przypadku zagadnień klasyfikacyjnych bardziej właściwe jest korzystanie z funkcji *logistycznej* lub *softmax*. W tym przypadku NETLAB korzysta z modeli probabilistycznych, opartych na prawdopodobieństwach warunkowych $p(\mathbf{t}|\mathbf{x})$, które pozwalają na określenie tzw. *prawdopodobieństw a posteriori* (wnioskowanie Bayesowskie). Ostateczne kryterium pokrywa się z tzw. entropią krzyżową używaną w teorii informacji Shannona (patrz Rozdział 3).

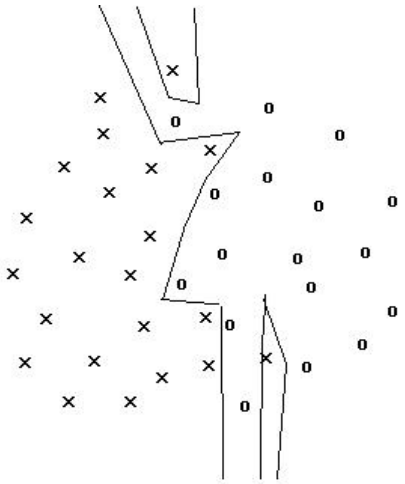
Sposób obliczania błędu E dla danej próbki uczącej lub testowej jest bardzo prosty – odpowiednie wzory s podane w rozdziale 3. Poniżej podajemy fragment funkcji 'mlperr' wykonującej te obliczenia dla danych tablic 'x' oraz 't'. Błąd E obliczany przez funkcję 'mlperr' dla sieci o nazwie 'net' jest tu oznaczony jako 'edata' (tj. error of data).

```
y = mlpfwd(net, x);~~ % obliczmy wynik sieci dla danych 'x'
switch net.outfn
  case 'linear'     % Linear outputs
    edata = 0.5*sum(sum((y - t).^2));
  case 'logistic'   % Logistic outputs
    edata = - sum(sum(t.*log(y) + (1 - t).*log(1 - y)));
  case 'softmax'    % Softmax outputs
    edata = - sum(sum(t.*log(y)));
  otherwise
    error(['Unknown activation function ', net.outfn]);
end
```

Błąd ten należy koniecznie sprawdzić na próbce testującej (dla której również są znane wartości docelowe).

2.5 Trenowanie sieci – ile neuronów

Procedura MLP dopuszcza dowolną liczbę neuronow w warstwie ukrytej. Większa liczba neuronów powoduje zmniejszenie się błędu dla danych próbki uczącej. Jednak może tu nastąpić zjawisko *przetrenowania* neuronu. Wagi przystosowują się nadmiernie do pewnych przypadkowych osobliwości danych uczących. Pokazuje to Rysunek 3.2 poniżej.



Rysunek 2.2: *Zjawisko 'przetrenowania' sieci. Widoczna jest linia łamana wynikająca z wprowadzenia dużej liczby neuronów do przyjętego modelu sieci. Linia ta (ang. decision boundary) rozdziela perfekcyjnie dwie grupy obiektów próbki uczącej. Jednak wykreślona granica decyzyjna wygląda dość sztucznie.* {*dis20j.jpg*}

Wagi otrzymane z przetrenowanej sieci (dające świetne wyniki na próbce uczącej) zastosowane do obliczeń innej próbki (np. testowej) powodują na ogół pogorszenie jakości predykcji. Dlatego przy trenowniu sieci zalecane jest następujące postępowanie

1. Podzielić dane, na których chcemy uczyć sieć, na dwie części: próbkę uczącą i próbkę testującą.
2. Przyjąć wartości z pewnego przedziału, np. H = [1, 2, 5, 7, 10].
3. Dla każdej wartości h∈H (Matlab: for h = H) wykonać punkty:
   - wykonać trenowanie sieci z h neuronami ukrytymi,
   - zapamiętać błąd $E_1(h)$ obliczony przez sieć dla obliczanej próbki (błąd ten jest zapamiętany w 'options(8)')
   - Obliczyć - na aktualnie wytrenowanej sieci – błąd $E_2(h)$ dla próbki testującej. Można tu posłużyć się funkcją e=mlperr(net,x,t).

   Tu jest koniec pętli for wykonywanej dla h=H.
4. Wykreślić dwa wykresy (na subplotach) ilustrujące wielkość błędu $E_1(h)$ i $E_2(h)$ jako funkcję h. Nie zapomnieć o opisie osi. Porównać oba wykresy. Znaleźć wspólne minimum, o ile to możliwe.
   Uwaga! Jeśli obie próbki nie były jednakowo liczne, to wystandaryzować odpowiednio obydwie krzywe E_1 i E_2.

Przy trenowaniu pierwszej próbki się obserwuje zjawisko malenia błędu przy wzroście liczby neuronów. Dlatego też funkcja $E_1(h)$ jest funkcją malejącą argumentu h.

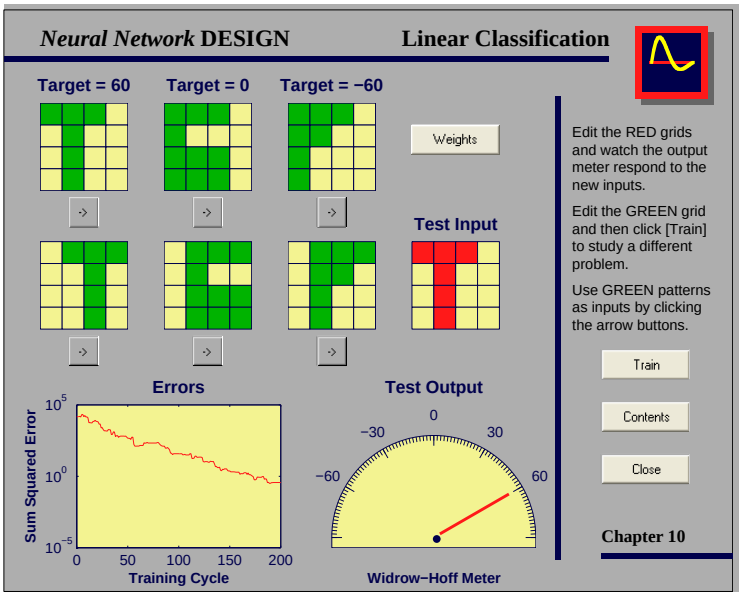
Błąd $E_2(h)$ wyznaczany w próbce testowej, rozpatrywany jako funkcja argumentu h powinien najpierw maleć, a potem znowu zacząć rosnąć. Najbardziej odpowiednią liczbą neuronów jest ta, dla której krzywa $E_2(h)$ wykazuje minimum.

2.6 Ocenianie jakości wyników sieci neuronowych

Dla zagadnień regresyjnych:
Współczynnik determinacji, Wykresy y against t, lub e=(t-y) against t.
Dla zagadnień klasyfikacyjnych:
Confusion matrix – tablica pomieszania. Można skorzystać z funkcji
conffig(y,t); gdzie y,t oznaczają wynik sieci i wartości prawdziwe (docelowe, target) odpowiednio.
Funkcja conffig wywoluje dwie inne funkcje:
[C, rate] = confmat(y, t);, oraz
plotmat(C, textcolour, gridcolour, fontsize);

2.7 Interesujące demonstracje komputerowe

Pakiet NETLAB:
demnlab uruchamia GUI demonstrujące możliwości Netlaba,
demtrain – uruchamia GUI do trenowania perceptronu.
Moduł korzysta ze specjalnie przygotowanych danych – zobacz plik xor.dat w tej samej kartotece.
Firmowy pakiet NNET uruchamiany rozkazem nnd.
Ponizej pokazano interfejs graficzny modułu nnd4lc – learning classification.
W wykładzie 3 pokazano interfejst modułów z tej serii, mianowicie
nnd4db – decision boundary, oraz nnd10nc – adaptive noise cancellation.



Literatura

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

[2] Ch. M. Bishop, *Neural networks: a pattern recognition perspective.* Technical Report NCRG/96/001, http://www.ncrg.aston.ac.uk/

[3] Ian Nabney, *Netlab: Algorithms for Pattern Recognition.* Springer 2001. Seria: Advances in Pattern Recognition. ISBN 1–85233–440–1.

[4] Netlab neural network software, Neural Computing Research Group, Division of Electric Engineering and Computer Science, Aston University, Birmingham UK, http://www.ncrg.aston.ac.uk/

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

Dodatek 1. Fragment funkcji **mlpfwd** obliczający wynik sieci

```
ndata = size(x, 1);
z = tanh(x*net.w1 + ones(ndata, 1)*net.b1);
a = z*net.w2 + ones(ndata, 1)*net.b2;
switch net.outfn     ~~% outf -- zadeklarowana funkcja aktywacji
  case 'linear'    % Linear outputs
    y = a;
  case 'logistic'  % Logistic outputs % Prevent overflow and underflow:  opuszczone
    y = 1./(1 + exp(-a));
  case 'softmax'   % Softmax outputs  % Prevent overflow and underflow: opuszczone
    temp = exp(a);
    y = temp./(sum(temp, 2)*ones(1, net.nout));
  otherwise
    error(['Unknown activation function ', net.outfn]);
end
```

Spis treści