{ plik rbf.tex, 11. grudnia 2008, popr. 05.12.2009 i 14.12.2010}

9 Sieci RBF – o radialnych funkcjach bazowych

9.1 Sieci RBF – ogólne wprowadzenie

Sieci te należą dzisiaj do podstawowych typów sieci neuronowych i są przedstawiane w zasadzie w każdym nowszym podręczniku o tematyce ANN; w szczególności np. w książkach Osowskiego [2] (rozdział 5), Bishopa [1] (rozdział 5), Nabney'a [4] (rozdział 6), oraz w artykule przeglądowym Robiego Polikara [5].

Sieci tego typu służą najczęściej do nieliniowej aproksymacji zmiennych numerycznych. Są używane również w zagadnieniach klasyfikacyjnych (por. Bishop [1], Nabney [4]) gdzie odtwarzają funkcje gęstości $p(\mathbf{x})$ rozkładu zmiennych objaśniających $\mathbf{x} = (x_1, \ldots, x_d)^T$. Z różnych twierdzeń (np. Poggio i Girosi, 1990) wynika, że taka sieć może służyć jako uniwersalny aproksymator i przybliżyć z dowolną dokładnością dowolną funkcję ciągła określoną na dziedzinie wielozmiennej.

Zasadnicza różnica sieci RBF – w porównaniu do sieci GLM lub MLP – ujawnia się w odmiennym działaniu warstwy ukrytej oraz w odmiennym sposobie trenowania takiej sieci. W przeciwieństwie do perceptronu, w sieciach RBF neurony są rozmieszczane w przestrzeni danych jako tzw. *centra*; na ogół proporcjonalnie do gęstości punktów-danych. W ten sposób działają one lokalnie – jako liniowe asocjatory (linear associators).

9.2 RBF – Definicja i przykłady

Definicja radialnej funkcji bazowej (RBF). Niech $\mathbf{x}$ i $\mathbf{c}$ oznaczają dwa punkty leżące w przestrzeni R^d. Punkt $\mathbf{c}$ uważamy za ustalony i nazywamy **punktem–centrum**. Punkt $\mathbf{x}$ uważamy za zmienny.

Definicja. Radialną funkcją bazową (typu RBF) nazywamy funkcję G postaci

$$G(\mathbf{x}; \mathbf{c}) = G(\, r(\mathbf{x}, \mathbf{c}\,)) \tag{9.1}$$

gdzie $r(\mathbf{x}, \mathbf{c}\,)$ jest odległością między punktami $\mathbf{x}$ i $\mathbf{c}$. Centrum $\mathbf{c}$ w powyższej definicji jest ustalone i gra rolę parametru funkcji. Funkcje radialne są nazywane czasami *jądrami* (kernels).

Z zapisu (9.1) wynika, że wartości funkcji $G(.)$ – dla danego argumentu $\mathbf{x}$ – zależą tylko od **odległości** r argumentu ($\mathbf{x}$) od centrum $\mathbf{c}$. Jednak formuła określająca funkcję G, może zawierać dodatkowe parametry, np. parametr σ określający szerokość jądra.

Określanie odległości. Przy określaniu odległości między punktami $\mathbf{x}$ oraz $\mathbf{c}$ używa się najczęściej odległości euklidesowej

$$r = r(\mathbf{x}, \mathbf{c}\,) = ||\mathbf{x} - \mathbf{c}||_2 = \{(\mathbf{x} - \mathbf{c})^T(\mathbf{x} - \mathbf{c})\}^{1/2}.$$

Stosuje się również tzw. odległość Mahalanobisa D, której kwadrat jest określony wzorem:

$$D^2 = D^2(r(\mathbf{x}, \mathbf{c}\,)) = (\mathbf{x} - \mathbf{c})^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{c}). \tag{9.2}$$

Jest to odległość liczona w metryce wyznaczonej macierzą $\mathbf{\Sigma}^{-1}$, gdzie macierz $\mathbf{\Sigma}$ oznacza macierz kowariancji rozważanych zmiennych $X_1, X_2, \ldots, X_d$. Dla $\mathbf{\Sigma}^{-1} = \mathbf{I}$ odległość Mahalanobisa sprowadza się do odległości euklidesowej.

PRZYKŁADY FUNKCJI RADIALNYCH

Przykład 1. Funkcja Gaussa. Radialna funkcja Gaussa jest określona wzorem [1]

$$G(\mathbf{x};\mathbf{c},\sigma^2) = \exp\{-\frac{||\mathbf{x}-\mathbf{c}||^2}{2\sigma^2}\}. \tag{9.3}$$

W pakiecie Netlab radialna funkcja Gaussa jest określona wzorem [3]

$$G(\mathbf{x};\mathbf{c},\sigma^2) = \exp\{-\frac{||\mathbf{x}-\mathbf{c}||^2}{\sigma^2}\}. \tag{9.4}$$

W obydwu przypadkach est to funkcja o kształcie dzwonu. Szerokość dzwonu reguluje parametr σ^2 nazywany szerokością (width) lub parametrem gładkości (smoothness) lub spłaszczenia. Powiększanie parametru σ^2 powoduje, że 'dzwon' staje się coraz to szerszy i niższy.

I tak np. dla wartości $\sigma_2^2 > \sigma_1^2$ funkcja $G(\mathbf{x};\mathbf{c},\sigma_2^2)$ jest szersza i bardziej spłaszczona aniżeli funkcja $G(\mathbf{x};\mathbf{c},\sigma_1^2)$ (por. Rys. 9.1).

Przykład 2. Funkcja TPS, Thin Plate Spline.

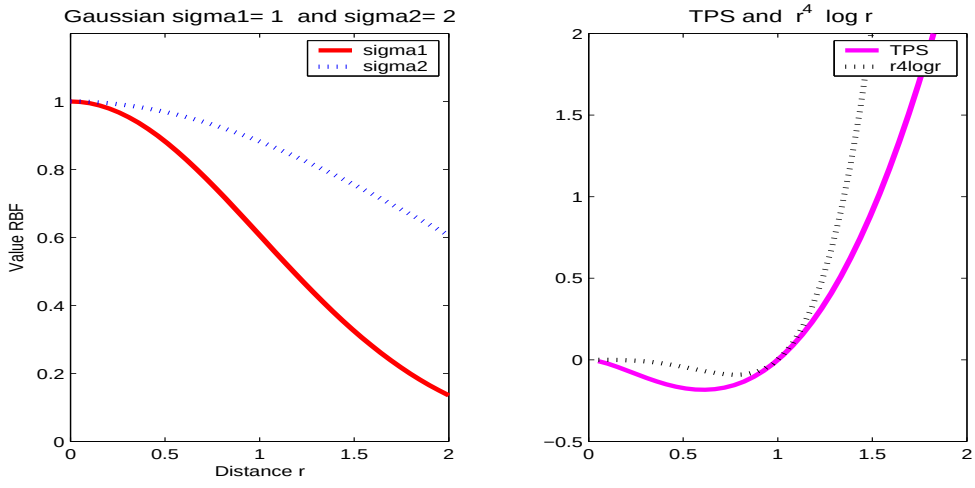
$$G(r) = r^2 log(r). \tag{9.5}$$

Przykład 3. Radykalna funkcja TPS. Jest to funkcja TPS pomnożona przez r^2:

$$G(r) = r^4\, log(r)$$

Dla obydwu funkcji TPS (tj. zwyłej i radykalnej) mamy: $\lim_{r\to 0} G(r) = 0$.

W pakiecie Netlab zaimplementowano wszystkie trzy wymienione wyżej funkcje aktywacji. Rysunek 9.1 pokazuje przebiegi tych funkcji – w zależności od wielkości r.



Rysunek 9.1: Wykresy różnych funkcji bazowych w zależności od argumentu r, gdzie r oznacza odległość między wektorami–punktami $\mathbf{x}$ i $\mathbf{c}$: $r = [(\mathbf{x}-\mathbf{c})^T(\mathbf{x}-\mathbf{c})]^{0.5}$. Lewy rysunek: Funkcja *Gaussian* dla dwóch wartości parametru σ. Prawy rysunek: Funkcje *TPS* ($r^2 log(r)$) oraz $r^4 log(r)$. {Plik rad2.eps}

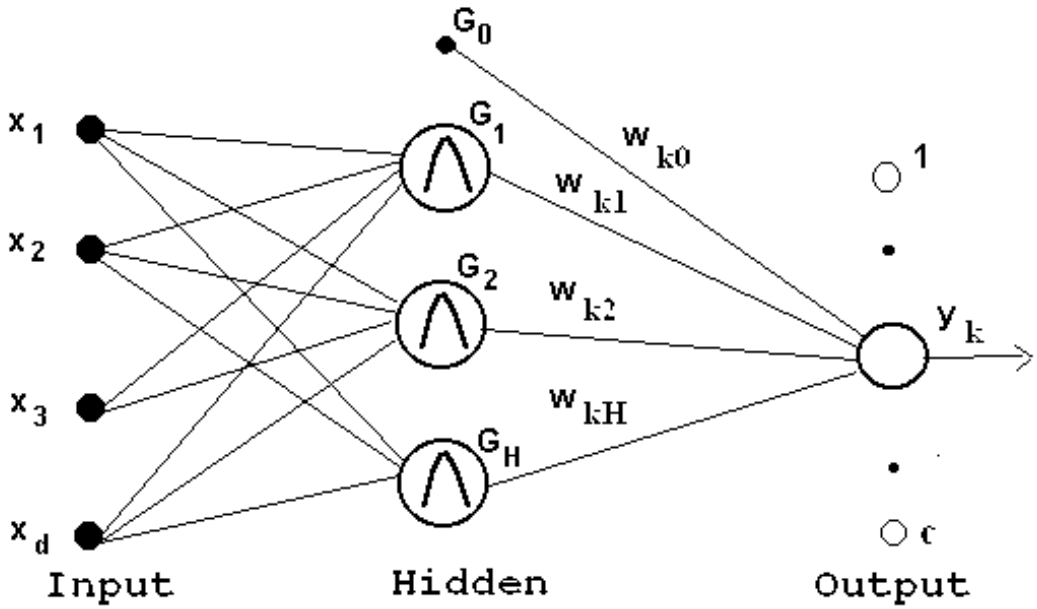
Przykłady innych funkcji radialnych (wraz z wykresami) można znaleźć np. w książce Osowskiego, str. 168, Rys. 5.4.

9.3 Architektura sieci typu RBF

Sieci RBF, oprócz zerowej warstwy wejściowej, zawierają jedną warstwę ukrytą i warstwę wyjściową. Przepływ informacji następuje do przodu, bez wymiany informacji między neuronami w jednej warstwie. Sieć RBF jest siecią, o architekturze:

$$\boxed{d \text{ inputs} \rightarrow H \text{ hidden} \rightarrow c \text{ outputs.}}$$

Wartości d, H, c są podawane przez użytkownika. Dodatkowo użytkownik powinien określić radialną funkcję bazową $G(r(\mathbf{x}, \mathbf{c}))$, przetwarzającą dane w warstwie ukrytej. Architektura sieci RBF jest przedstawiona na rysunku 9.3.



Rysunek 9.2: *Sieć RBF o d wejściach, H radialnych funkcjach bazowych i c wyjściach. Podawanie informacji następuje tylko wprzód, tzn. od lewej (wejście) poprzez przetwarzanie przez neurony warstwy ukrytej – do neuronów warstwy wyjściowej. {netrbf.ps}*

Omówimy teraz działalnie warstw: wejściowej, ukrytej i wyjściowej sieci RBF.

Warstwa wejściowa działa podobnie, jak w znanych nam już sieciach MLP lub GLM. Podajemy tutaj sieci dane z tablicy $\mathbf{X}_{N\times d}$ – podawanie następuje wierszami. Kolejne wiersze tej tablicy są interpretowane jako punkty $\mathbf{x}^n = (x_1^n, \ldots, x_d^n)^T \in R^d$.

Warstwa ukryta składa się z H neuronów. Każdy z neuronów (h) posiada swoje centrum $\mathbf{c}_h \in R^d$. Centra te, w liczbie H, są inicjowane podczas tworzenia struktury sieci, a następnie – podczas procesu uczenia (trenowania) sieci – są dostosowywane do danych treningowych. Należy wyraźnie zaznaczyć, że nie bierze się na tym etapie wartości docelowych, czyli targetu $\mathbf{t}$ ($\mathbf{T}$).

Dla danego wektora wejściowego $\mathbf{x}^n$ zostają obliczone – przez kolejne neurony – ich aktywacje wyjściowe {wzór Gxcn}

$$\mathbf{z}^n = [z_1^n, \ldots, z_H^n] = [G(\mathbf{x}^n, \mathbf{c}_1),\ G(\mathbf{x}^n, \mathbf{c}_2), \ldots,\ G(\mathbf{x}^n, \mathbf{c}_H)]. \tag{9.6}$$

Dla N wektorów wejściowych zostaje obliczona macierz aktywacji $\mathbf{Z}_{N\times d}$. Rozpisując bardziej szczegółowo:

Najpierw na podstawie danych wejściowych $\mathbf{X}$ zostają obliczone wartości $\mathbf{Z}$, czyli aktywacje wyjściowe neuronów warstwy ukrytej:

$$\mathbf{X} = \begin{bmatrix} x_1^1 & \dots & x_d^1 \\ \vdots & \dots & \vdots \\ x_1^N & \dots & x_d^N \end{bmatrix} \Rightarrow \mathbf{Z} = \begin{bmatrix} z_1^1 & \dots & z_H^1 \\ \vdots & \dots & \vdots \\ z_1^N & \dots & z_H^N \end{bmatrix} = \begin{bmatrix} G(\mathbf{x}^1, \mathbf{c}_1) & \dots & G(\mathbf{x}^1, \mathbf{c}_H) \\ \vdots & \dots & \vdots \\ G(\mathbf{x}^N, \mathbf{c}_1) & \dots & G(\mathbf{x}^N, \mathbf{c}_H) \end{bmatrix}$$

Otrzymana w ten sposób macierz $\mathbf{Z}$ stanowi dane wejściowe dla c neuronów warstwy wyjściowej.

Warstwa wyjściowa zawiera neurony, które działają podobnie, jak w sieciach GLM i MLP. Każdy z neuronów (k) ma przypisany wektor wag $\mathbf{w}_k = [w_{k1}, \dots, w_{kH}]^T$ oraz wartość progową (bias) w_{k0}. Tym samym mamy dla warstwy wyjściowej określoną macierz wag i wektor biasów

$$\mathbf{W_{H\times c}} = [\mathbf{w_1}, \dots, \mathbf{w_c}], \quad \mathbf{b_{1\times c}} = [w_{1,0}, \dots, w_{c,0}].$$

Każdy neuron (k) – posługując się swoimi wagami i swoim biasem – wykonuje ważone (swoimi wagami $\mathbf{w}_k$) sumowanie dochodzących do niego z warstwy ukrytej sygnałów $\mathbf{z}^n = [z_1^n, \dots, z_H^n]$, dodaje do otrzymanej sumy swój bias w_{k0}, i przekazuje ostateczną sumę na zewnątrz – jako wynik y_k^n odpowiadający wektorowi wejściowemu $\mathbf{x}^n$.

W końcowym rezultacie, dla całej macierzy $\mathbf{X}$ otrzymamy macierz odpowiedzi $\mathbf{Y}$

$$\mathbf{X}_{N\times d} \;\Rightarrow\; \mathbf{Y}_{N\times c}.$$

Macierz $\mathbf{Y}$ zostaje obliczona w wyniku następującego działania macierzowego:

$$\mathbf{Y}_{N\times c} = [\mathbf{Z}, \mathbf{1}_n]_{N\times H+1} \begin{bmatrix} \mathbf{W} \\ \mathbf{b} \end{bmatrix}_{H+1\times c}.$$

Tak jest dla wytrenowanej sieci. Potrzebne wagi i biasy otrzymuje się w procesie trenowania sieci przy danych wartościach docelowych $\mathbf{T}$.

9.4 RBF – Uczenie sieci w pakiecie Netlab

Rozróżniamy tu dwa etapy: I. Wyznaczenie centrów reprezentujących przestrzeń zmiennych objaśniających. II. Wyznaczanie wag używanych przez sieć przy aproksymacji wartości docelowych. Etap II jest wykonywany po zakończeniu etapu I.

Etap I Rozmieszczanie centrów dokonuje się niezależnie od wartości docelowych (targetu) jakiemu sieć RBF ma służyć. Z wektora $\mathbf{x} = (x_1, \dots, x_d)^T$ zostaje wyznaczony nowy wektor $\mathbf{z} = (z_1, \dots, z_H)^T$, który jest funkcją odległości danego punktu $\mathbf{x}$ od przyjętych H *centrów*.

Etap II Wyznaczone wartości $\mathbf{z}$ służą jako liniowe aproksymatory wartości docelowych (targetu $\mathbf{t}$) rozpatrywanego zagadnienia.

Etap I. Wyznaczanie centrów i ewtl. parametrów spłaszczenia

Centra – w zadeklarowanej liczbie H – są rozstawiane w przestrzeni zmiennych objaśniających R^d. Wyznaczanie centrów może się odbywać w zasadzie na 3 sposoby:

1. rozmieszczając w sposób losowy H punktów w przestrzeni R^d;
2. przez podział punktów–wierszy próbki uczącej na H klasterów. Można to zrobić np. za pomocą procedury kmeans lub metodą mieszanin wykorzystującą algorytm EM. Algorytm k-means jest wyjaśniony niżej;

3. startujemy z liczbą klasterów równą liczebności próbki uczącej – każdy punkt tej próbki stanowi wtedy jedno centrum, a błąd reprezentacji (np. obliczany ze wzoru [9.7]) wynosi 0. Następnie redukujemy stopniowo liczbę centrów, aż zostanie ich tylko H.

Metoda **k-means** startuje z losowego rozmieszczenia H punktów, które pełnią rolę chwilowych centrów w przestrzeni danych R^d. Następnie wykonywane są iteracyjnie dwa kroki:

(a). Dla każdego punktu $(\mathbf{x}^q) \in R^d$ zostaje znalezione najbliższe mu centrum (w sensie przyjętej metryki). Tym samym otrzymamy podział całego zbioru danych na H podzbiorów $S_1, \ldots, S_H$. Dla każdego z nich obliczamy średnią arytmetyczną (środek ciężkości):

$$\mathbf{m}_h = \frac{1}{N_h} \sum_{q \in S_h} \mathbf{x}^q, \quad h = 1, \ldots, H.$$

(b). Wyznaczone w ten sposób średnie $\mathbf{m}_1, \ldots, \mathbf{m}_H$ przyjmujemy jako nowe centra. Zostaje wyznaczony aktualny błąd reprezentacji

$$E = \sum_{h=1}^{H} \sum_{q \in S_h} ||\mathbf{x}^q - \mathbf{m}_h||^2. \tag{9.7}$$

Uaktualnianie centrów (średnich $\{\mathbf{m}_h\}$) – poprzez wykonywanie kroków (a) i (b) kontynuujemy tak długo, aż błąd E ustabilizuje się, lub zostanie przekroczona maksymalna liczba iteracji.

Metoda gmm (general mixture model) dostarcza znacznie lepszego rozłożenia centrów wsród obserwowanych danych $\mathbf{x}$. Pakiet Netlab wyznacza centra właśnie metodą **gmm**. Otrzymane z małej liczby iteracji (np. 5 iteracji metodą k-means) średnie stanowią wstępne przybliżenie, na podstawie których wyznacza się właściwe centra, używane później przez radialne funkcje bazowe.

Szerokość jądra σ^2 (Parametr spłaszczenia, ang. width) występujący przy używaniu funkcji Gaussowskiej określonej wzorem (9.4) szacuje się na podstawie postępowania heurystycznego; np. średniej odległości każdego centrum od najbliższego jego sąsiada:

$$\sigma^2 = \frac{1}{H} \sum_{i=1}^{H} ||\mathbf{m}_i - \mathbf{m}_j||, \quad \text{gdzie } \mathbf{m}_j \text{jest najbliższym sąsiadem } \mathbf{m}_i \,.\mathbf{m}_j, \;\; j \neq i, \; j = 1, \ldots, H$$

Niekiedy przyjmuje się jako wstępne oszacowanie szerokości σ^2 maksimum z kwadratów odległości między parami centrów.

Możliwy jest również algorytm który przyjmuje, że punkty próbki uczącej są *mieszaniną* rozkładów gaussowskich o indywidualnych centrach i indywidualnych szerokościach jąder. Jednoczesne wyznaczanie centrów $\mathbf{c}_h$, $(h = 1, \ldots, H)$ i odpowiadających im szerokości σ_h^2 jest możliwe dzięki naprzemiennemu stosowaniu algorytmu EM. Algorytm takiego wyznaczania klasterów jest zrealizowany w pakiecie Netlab.

Parametr σ można wyznaczać również metodą cross-validation (k-fold cross-validation).

Przy realizacji w Netlabie w wyznaczaniu centrów bierze udział tylko próbka ucząca – dlatego należy zadbać, aby ta próbka była reprezentatywna dla wszystkich danych. Przy rozstawianiu centrów za pomocą funkcji Netlaba nie uwzględnia się wartości docelowych.

Etap II. Wyznaczanie wag.

Wyznaczanie wag, przypadek $c = 1$

Przyjmijmy, że $c = 1$ (jest tylko jedna odpowiedź). Mamy wtedy tylko jeden wektor wag $\mathbf{w} = [w_1, \ldots, w_H]^T$ i jeden bias w_0. Podstawiając jako $\mathbf{x}^n$ kolejno $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N$, oraz korzystając z (9.6) otrzymamy wtedy kolejno dla $n = 1, \ldots, N$: { wzór ypred}

$$y^n = G(\mathbf{x}^n, \mathbf{c}_1)w_1 + G(\mathbf{x}^n, \mathbf{c}_2)w_2 + \cdots + G(\mathbf{x}^n, \mathbf{c}_H)w_H + w_0 \,. \tag{9.8}$$

Przyjmijmy jako błąd sieci sumę kwadratów odchyleń między wartościami docelowymi t^n a wartościami y^n prognozowanymi przez sieć (aproksymacja średniokwadratowa), czyli wielkość: {errRBF}

$$E = \sum_{n=1}^{N} (t^n - y^n)^2. \tag{9.9}$$

Podstawiając do (9.9) na miejsce y^n odpowiednią wartość z równania (9.8) i przyjmując, że centra $\mathbf{c}_1, \ldots, \mathbf{c}_H$ są dane – widzimy, że błąd E jest funkcją kwadratową wag $w_1, \ldots, w_c$. **Wagi** te należy wybrać tak, aby zminimalizować błąd E.

W zaistniałej sytuacji mamy tu zagadnienie minimalizacji funkcji kwadratowej wag $w_1, \ldots, w_c$. Rozwiązanie może być otrzymane explicite poprzez rozwiązanie układu liniowych równań względem szukanych niewiadomych (układ *równań normalnych*).

`Matlab` rozwiązuje taki układ równań poprzez operację dzielenia lewostronnego, czyli operację `left matrix divide` (funkcja mldivide).

Wyznaczanie wag, przypadek $c > 1$

Mamy wtedy do czynienia z tablicą $\mathbf{Y}_{N\times c} = (y_j^n)$, $j = 1, \ldots, c$ zawierającą c kolumn wyników.

Również tablica wartości docelowych $\mathbf{T}_{N\times c} == (t_j)^n$ jest tablicą o c kolumnach.

Błąd E sieci jest teraz sumą błędów dla każdej kolumny wyników:

$$E = \sum_{j=1}^{c} \sum_{n=1}^{N} (t_j^n - y_j^n)^2 = \sum_{j=1}^{c} E_{[j]}, \quad \text{gdzie } E_{[j]} = \sum_{n=1}^{N} (t_j^n - y_j^n)^2.$$

Tym samym zagadnienie ogólnej optymizacji rozpada się na niezależne zagadnienia optymizacji dla kolejnych kolumn wyników – co odpowiada optymizacji dla przypadku $c = 1$.

Sieć RBF z Netlaba, o wbudowanej na sztywno funkcji aktywacji 'linear', oblicza błąd E jako błąd średniokwadratowy pomnożony przez współczynnik $1/2$. Błąd ten jest obliczany w dwóch rozkazach:

`y = rbffwd(net, x);  e = 0.5*sum(sum((y - t).^2));`

Symbole `y`, `t` odpowiadają tutaj naszym tablicom $\mathbf{Y}$ i $\mathbf{T}$. Symbol `e` odpowiada naszemu błędowi E.

Można również skorzytać z funkcji rbferr, obliczając `e = rbferr(net, x, t)`.

9.5 Implementacja sieci RBF w pakiecie Netlab

Radialne sieci neuronowe (RBF)w pakiecie Netlab posiadają swój własny konstruktor używający przedrostka `rbf`.

W pakiecie Netlab znajduje się również moduł demonstracyjny o nazwie `DEMRBF` lub `DEMRBF1`. Moduł ten pokazuje, jak można otrzymać przybliżenie sinusoidy za pomocą

różnego typu funkcji radialnych: Gaussowskiej ('gaussian'), Thin Plate Spline ('TPS', funkcja $G(r) = r^2 \log r$) oraz Enhanced TPS ('r4logr', funkcja $G(r) = r^4 \log r$).

Ponieważ typowe obliczenia w netlabie są oparte na odległościach Euklidesowych, to dane wejściowe ($\mathbf{x}$) powinny zostać wystandaryzowane[1].

Istotną rolę odgrywają trzy funkcje: rbf, rbftrain i rbffwd:

net = rbf(nin, nhidden, nout, rbfunc);
[net, options]= rbftrain(net, options, x, t);
Y = rbffwd(net, X); lub [Y, Z, N2] = rbffwd(net, X)

Funkcja RBF tworząca strukturę `net`

Podstawową strukturę 'net' tworzy się za pomocą funkcji:

net = rbf(nin, nhidden, nout, rbfunc);[2]

Znaczenia parametrów wejściowych funkcji rbf:

nin - d, liczba neuronów wejściowych (składowych wektora danych),

nhidden - H, liczba neuronów ukrytych (funkcji bazowych),

nout - M, liczba neuronów wyjściowych,

rbfun - napis określający typ funkcji bazowych; w Netlabie dopuszcza się funkcje *'gaussian'*, – radialna funkcja gaussowska, *'tps'* – Thin Plate Spline, funkcja $G(r) = r^2 log\, r$ lub *'r4logr'* – funkcja $G(r) = r^4 logr$.

Pola utworzonej struktury `net`:

type - typ sieci, w tym przypadku RBF,

nin - d, liczba neuronów wejściowych,

nhidden - H, liczba neuronów ukrytych (funkcji bazowych),

nout - c, liczba neuronów wyjściowych,

actfn - napis określający funkcję aktywacji jednostek ukrytych, dopuszcza się *'gaussian'*, *'tps'* lub *'r4logr'* (por. parametry wejściowe funkcji rbf).

outfn - napis definiujący funkcję błędu dla wyjść; jeżeli wywołaliśmy funkcję *rbf* tylko z 4 parametrami, to zostanie tu podstawiony napis *'linear'*; inna możliwość to *'neuroscale'* (dla nie omawianej przez nas funkcji błędu *Sammon stress measure*),

nwts - całkowita liczba parametrów modelu (wag, centrów i szerokości funkcji bazowych),

c - tablica wymiaru $nhidden \times d$ zawierająca – zainicjowane wg. rozkładu normalnego $N(0, 1)$ – wartości współrzędnych centrów,

wi - od *width*, wektor $[1 \times nhidden]$ zawierający szerokości szerokości σ_i^2, dla przyjętych funkcji bazowych. Pole to zostaje wypełnione jedynkami tylko przy funkcjach bazowych typu *'gaussian'*, które – jak wiadomo – zawierają parametr σ^2; natomiast przy funkcjach bazowych nie zawierających tego parametru pole to pozostaje puste.

w2 - macierz wymiaru $nhidden \times nout$ zawierająca wagi drugiej warstwy,

[1] nie jest to konieczne, jeśli używa się odległości Mahalanobisa

[2] Możliwe jest również tworzenie struktury *net* za pomocą funkcji rbf dla której określamy dodatkowo trzy dalsze parametry: net = rbf(nin, nhidden, nout, rbfunc, outfunc, prior, beta). Wtedy utworzona tym rozkazem struktura *net* będzie miała określone dodatkowe pola i będzie mogła obliczać nieliniową metodę *Generative Topographic Mapping*

b2 - wektor wymiaru $1 \times nout$ zawierający wartości progowe (biasy) dla drugiej warstwy.

Przykładowo, dla wywołania
net = rbf(3, 5, 1, 'gaussian'); otrzymuje się sieć o strukturze

```
net =
       type: 'rbf'
        nin: 3
    nhidden: 5
       nout: 1
      actfn: 'gaussian'
      outfn: 'linear'
       nwts: 26
          c: [5x3 double]
         wi: [1 1 1 1 1]
         w2: [5x1 double]
         b2: -1.1651
```

Ponieważ nhidden=5, wygenerowanych zostało 5 centrów z rozkładu N(0,1). Ponieważ nout=1, został wygenerowany tylko 1 wektor wag w = [w_1, ..., w_5]^T. Zainicjowane centra oraz wagi wynoszą odpowiednio:

```
c1 = ( 0.7566    0.1727   -0.0220)       w_1 = -1.3355
c2 = ( 1.1642    0.3541    0.6183)       w_2 = -0.1231
c3 = (-1.0235   -0.2463    1.8659)       w_3 = -1.1028
c4 = ( 1.7016   -0.1457    0.0819)       w_4 = -2.7532
c5 = (-0.4942   -1.1690    1.6080)       w_5 =  0.2520
```

Szerokości jąder (wi) (parametr σ^2) otrzymały standardowo wartości 1.

Liczba parametrów wygenerowanego modelu sieci neuronowej wynosi:
15 (centra f. bazowych) + 5 (szerokości) + 6 (wagi plus bias warstwy wyjściowej) = 26.

Funkcja RBFFWD obliczająca wyniki dla nowych danych

Funkcja `rbffwd` pozwala utworzonej sieci pracować w trybie odtworzeniowym.

Nagłówek funkcji:

[Y, Z, N2] = rbffwd(net, X) lub tylko y = glmfwd(net, x);

Tablica $\mathsf{X}_{N\times d}$ zawiera dane wejściowe (próbka testowa lub inna).
Tablica $\mathsf{Y}_{N\times M}$ oznacza wynik sieci (y) dla każdego wiersza danych wejściowych X.
Tablica $\mathsf{Z}_{N\times H}$ oznacza aktywacje wyjściowe H neuronów warstwy ukrytej, czyli wartości $G(\mathbf{x}, \mathbf{c}_1), \ldots, G(\mathbf{x}, \mathbf{c}_H)$ obliczone dla każdego wiersza tablicy danych wejściowych X.
Tablica $\mathsf{N2}_{N\times H}$ jest tablicą kwadratów odległości każdego wiersza tablicy danych X od H centrów funkcji bazowych.

Funkcja RBFTRAIN trenująca sieć wg. próbki uczącej

Nagłówek funkcji:

[net, options]= rbftrain(net, options, x, t);

gdzie x jest próbką uczącą, a t – tablicą wartości docelowych (*target*). Tablica 'options' może być skopiowaną tablicą foptions (tablica jednowierszowa o 18 elementach). W przypadku specjalnego trenowania (na użytek metody 'neuroscale'), tablica options może być dwuwierszowa.

Tablica options określa różne warunki trenowania sieci. Dopuszcza się, aby tablica options miała dwa wiersze. Na samym początku działania funkcja rbftrain sprawdza, ile wierszy ma wejściowa tablica options. Jeżeli jest tylko 1 wiersz, to nie bierze się pod uwagę drugiego wiersza. W przeciwnym przypadku, tzn. jeśli są 2 wiersze, pierwszy z nich zostaje podstawiony jako jednowierszowy 'options', a drugi z nich jako dodatkowa tablica 'setbfoptions'.

A oto odpowiedni fragment kodu:

```
% Allow options to have two rows: if this is the case, then the second row
% is passed to rbfsetbf
if size(options, 1) == 2,
      setbfoptions = options(2, :);  options = options(1, :);
   else setbfoptions = options; end %if size
```

Funkcja `rbftrain` (głównie w swej części dotyczącej `neuroscale`) korzysta z tablicy `options`. Ewentualnie wywoływana we wnętrzu tej funkcji inna funkcja, `rbfsetbf`, korzysta z tablicy `setbfoptions`.

Znaczenia elementów tablicy options:

options(1) - wartość =1: drukowanie wartości kryterium podczas każdej iteracji EM,

options(2) - określa dokładność wag wymaganą na koniec uczenia,

options(3) - określa dokładność funkcji celu (błędu) wymaganą na koniec uczenia,

options(5) - wartością domyślną jest options(5)=0. Oznacza to, że centra znajdujące się w wejściowej strukturze `net` mają – w czasie działania `rbftrain` – zostać przystosowane do rozkładu punktów danych w próbce uczącej; wartość options(5)=1 postuluje, żeby parametry funkcji bazowych (centra) pozostały takimi, jakie są;

Ewentualne 'przystosowanie do danych' jest dokonywane za pomocą funkcji `rbfsetbf`, ta z kolei wywołuje `kmeans` i `gmm`.

options(14) - maksymalna liczba iteracji, default=100 (odnosi się to do procesu rozmieszczania centrów przy metodzie `neuroscale`).

Etapy trenowania sieci RBF za pomocą funkcji `rbftrain`

Funkcja rbftrain działa w dwóch głównych blokach, odpowiadających dwom etapach trenowania.

Funkcja `rbftrain` zaimplementowana w Netlabie jest wykorzystywana również do wizualizacji danych metodą `neuroscale` – dlatego jej przebieg nie jest bardzo przejrzysty.

Na samym początku zostaje określona tablica `setbfoptions` – patrz fragment skryptu wyżej.

Następnie wykonuje się blok 1 (etap I), a po nim blok 2 (etap 2).

Etap I Jeśli setbfoptions(5)==0, to zostaje uruchomiona funkcja `rbfsetbf` (wywołanie:
(net = rbfsetbf(net, setbfoptions, x)). Funkcja ta przystosowuje centra do rozkładu danych. Najpierw zostaje uruchomiona funkcja kmeans rozmieszczająca w sposób dość gruby centra w analizowanej przestrzeni danych R^d. Następnie uruchamia się model mieszanin (*mixture model*), który powoduje lepsze dopasowanie rozmieszczanych centrów do rozkładu danych p($\mathbf{x}$). Wykorzystywane są wtedy funkcje `gmminit` i `gmmem`.

Jeśli options(7)==1, to ustawia się w rbfsetfw szerokości jąder.

Etap II trenowania sieci za pomocą funkcji rbftrain polega na znalezieniu wag i biasów charakteryzujących neurony warstwy wyjściowej.

W obecnej implementacji funkcja `rbftrain` dopuszcza tylko liniową (identycznościową) funkcję aktywacji i kwadratową funkcją błędów. Rozwiązanie otrzymuje się wtedy w jednym kroku, rozwiązując odpowiedni układ równań liniowych.

W Netlabie wykonuje się w tym celu operację `left matrix divide` [3] [4].

9.6 Przykładowy skrypt wykonujący aproksymację funkcji jednej zmiennej

Podajemy przykładowy skrypt wykonujący obliczenia dla 6-u różnych liczebności H warstwy ukrytej. Skrypt ten jest modyfikacją skryptu DEMRBF z pakietu Netlab. Wykonujemy aproksymację funkcji $y = sin(x)$ za pomocą sieci typu RBF o H = 2, 4, 7, 15 i 18 neuronach w warstwie ukrytej. Wyniki aproksymacji pokazujemy na rysunkach 4.3 i 4.4.

Przygotowania wstępne.

```
% Scipt DEMRBF1, based on DEMRBF from Netlab
% Approximation of sine function by a RBF network
% Generate the matrix of inputs x and targets t.
randn('state', 42);  rand('state', 42);
ndata = 20;           % Number of data points.
noise = 0.2;          % Standard deviation of noise distribution.
x = (linspace(0, 1, ndata))'; t = sin(2*pi*x) + noise*randn(ndata, 1);
mu= mean(x);  sigma = std(x);
data = (x - mu) / sigma;  % Standardized data
xf=[x(1):0.1:x(end)]';      % dane 'x' do rysowania funkcji
xfs= (xf-mu) / sigma; % wystandaryzowane
baseF={'gaussian','tps', 'r4logr'};
disp('   RBF training error for 3 kernels')
% Set up network parameters.
nin = 1;  nout = 1;     % Number of inputs and outputs.
```

Pętla po różnych wartościach neuronów.

```
H = [2 4 7 12 15 18]; % Number  of hidden neurons
for nhidden = H
```

[3] Generalnie, jeśli mamy układ równań

$$\mathbf{Ax} = \mathbf{b},$$

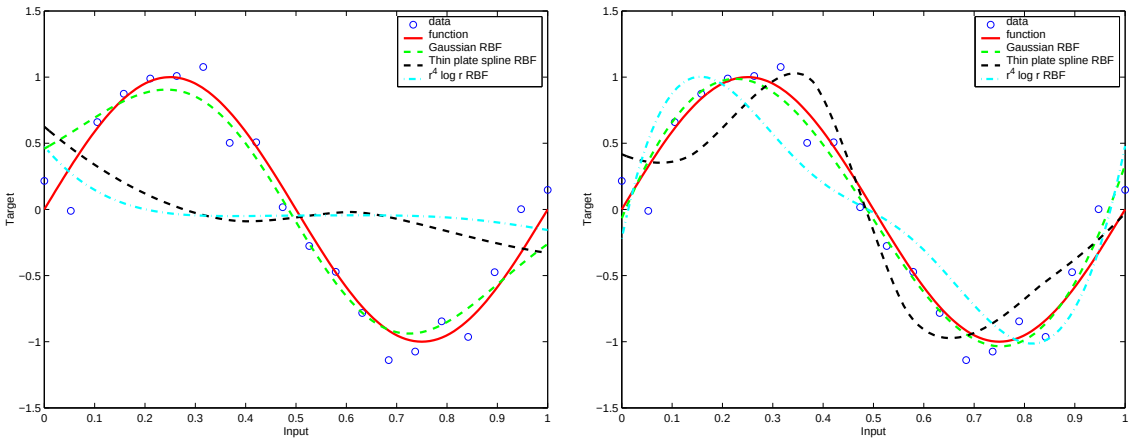
to operację `left matrix divide` zapisuje się jako:

$$\mathbf{x} = \mathbf{A}\backslash\mathbf{b},$$

mówi się również 'A sub b'

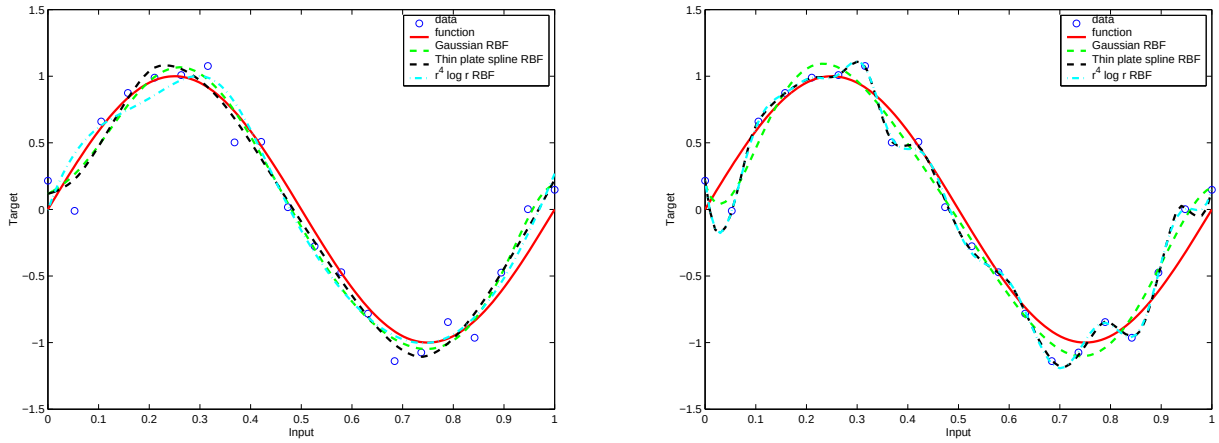
[4] Inną możliwością jest rozwiązanie układu równań liniowych za pomocą funkcji pinv wyznaczająca dla danej macierzy jej pseudoodwrotność (ang. pseudoinverse, patrz dokumentacja Matlaba)
PINV Pseudoinverse. X = PINV(A) produces a matrix X of the same dimensions as A' so that A*X*A = A, X*A*X = X and A*X and X*A are Hermitian. The computation is based on SVD(A) and any singular values less than a tolerance are treated as zero. The default tolerance is MAX(SIZE(A)) * NORM(A) * EPS
Bardziej zrozumiała definicja jest następująca:
K jest pseudoodwrotnością do A, jeśli wymiar(K)=wymiar(A'), a ponadto spełnia warunki: (i) AKA=A, (ii) KAK=K, (iii) (KA)'=KA, (iv) (AK)'=AK.

```
    % Create and initialize network weight and parameter vectors.
    net1 = rbf(nin, nhidden, nout, baseF{1});
    % Use fast training method
    e=zeros(1,3);   % Error for Gaussian, TPS, r^4logr
    options = foptions;
    options(1) = -1;         % No Display of EM when making clusters
    options(14) = 10;       % number of iterations of EM
    net1 = rbftrain(net1, options, data, t); % data - dane standaryz.
    yy = rbffwd(net1, xfs); e(1)=rbferr(net1,data,t);
    for k=2:3
    net = rbf(nin, nhidden, nout, baseF{k});
    net.c=net1.c;   % bierzemy to samo rozstawienie centrow
    options = foptions;
    options(1) = 0;  options(14) = 10; options(5)=1; % pomijamy I etap trenowania
    net = rbftrain(net, options, data, t); % wejscie z tymi samymi danymi
    y = rbffwd(net, xfs); yy=[yy y]; e(k)=rbferr(net,data,t);
    end %k
  figure;
  plot(x,t,'ob',x,sin(2*pi*x),xf,yy(:,1),xf,yy(:,2),xf,yy(:,3));
  legend('data','sinus',baseF{1},baseF{2},baseF{3});
  title(['Approximation by H=',int2str(nhidden),'  RBFs'])

    disp(['nhidden H= ', num2str(nhidden,'%2d'), ' ', ...
         baseF{1}, '  ', num2str(e(1),'%7.5f'), '  ', ...
         baseF{2}, '  ', num2str(e(2),'%7.5f'), '  ', ...
         baseF{3}, '  ', num2str(e(3),'%7.5f')]);

end % nhidden  %close all
```



Rysunek 9.3: *Aproksymacja funkcji sinus wykonana za pomocą sieci* RBF *o H=2 (lewa) i H=4 (prawa) neuronach. Sieci zostały wytrenowane na podstawie 20-elementowej zaburzonej próbki wylosowanej z sinusoidy. Pliki rr2c.eps i rr4c.eps*

Przykładowy wydruk



Rysunek 9.4: *RBF 7 i 15 neuronow . To samo, co na rysunku 5.3, ale warstwa ukryta sieci składa się z H=7 (lewa) i H=15 (prawa) neuronów. Pliki rr7c.eps i rr15c.eps*

```
--------------------------------------------------
RBF training errors for 3 kernels
nhidden H= 2:  gaussian 0.5087  tps 4.5182    r4logr 4.8817
nhidden H= 4:  gaussian 0.24256 tps 0.70335   r4logr 0.9769
nhidden H= 7:  gaussian 0.14586 tps 0.14092   r4logr 0.26191
nhidden H=12:  gaussian 0.10797 tps 0.089096  r4logr 0.097692
nhidden H=15:  gaussian 0.10797 tps 0.0032335 r4logr 0.0073178
nhidden H=18:  gaussian 0.10792 tps 0.0014216 r4logr 0.0043882
--------------------------------------------------
```

Pokażemy jeszcze przykładowe struktury net przed i po trenowaniu – dla omawianej aproksymacji funkcji jednej zmiennej za pomocą 3 funkcji bazowych ($m = 0.5,\ s = 0.3114$ dla wygenerowanych danych):

```
 net after initialization,
net =
       type: 'rbf'
        nin: 1
    nhidden: 3
       nout: 1
      actfn: 'gaussian'
      outfn: 'linear'
       nwts: 10
          c: [3x1 double]
         wi: [1 1 1]
         w2: [3x1 double]
         b2: 0.4912
   centers before training
standardized destandardized
    1.3720     0.9272
    1.3082     0.9074
   -0.8011     0.2506
```

```
net after training
net2 =
       type: 'rbf'
        nin: 1
    nhidden: 3
       nout: 1
      actfn: 'gaussian'
      outfn: 'linear'
       nwts: 10
          c: [3x1 double]
         wi: [3.7209 3.7209 3.7209]
         w2: [3x1 double]
         b2: 15.6285
   centers  after training
 standardized  destandardized
      -1.0019     0.1880
       0.9271     0.7887
      -0.0136     0.4958
 nhidden H= 3 error=2.427
```

Gdybyśmy chcieli na sporządzanych rysunkach zaznaczyć również centra funkcji bazowych, to powinniśmy zauważyć, że w wytrenowanej strukturze net centra te są podane we współrzędnych wystandaryzowanych (unormowanych) u.

Aby wrócić do oryginalnych współrzędnych x należy wykonać podstawienie odwrotne $c_x = c_u \times s + m$, gdzie m i s oznaczają wartość średnią i odchylenie stamdardowe, z jakimi wystandaryzowano dane oryginalne ($u = (x - m)/s$).

9.7 Uwagi użyteczne przy projektowaniu sieci RBF

Przy projektowaniu sieci RBF lub korzystaniu z gotowych algorytmów należy przede wszystkim zwrócić uwagę na standaryzację danych. Jeśli posługujemy się odległością Euklidesową, to wszystkie zmienne analizowanych danych powinny mieć taką samą wariancję. Gdyby tak nie było, to odległość Euklidesowa będzie zdominowana przez zmienną o największej wariancji.

Najchętniej przyjmuje się jako funkcje bazowe radialne funkcje Gaussowskie dane wzorem (9.3). Oznacza to sferyczność (izotropowość) odległości we wszystkich kierunkach przestrzeni zmiennych objaśniających. Moglibyśmy posługiwać się innymi odległościami, np. odległością Mahalanobisa, która bierze pod uwagę wariancje i kowariancje zmiennych objaśniających. Otrzymujemy wtedy eliptyczne jądrem gaussowskim postaci:

$$G(\mathbf{x}; \mathbf{c}, \mathbf{\Sigma}) = \exp\{-\frac{(\mathbf{x} - \mathbf{c})^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{c})}{2}\}. \tag{9.10}$$

Metoda mieszanin i algorytm EM radzą sobie również z takimi funkcjami; przy funkcjach postaci (9.10) nie jest konieczna standaryzacja danych.

Przy odtwarzaniu rozkładu łącznego zmiennej $\mathbf{x}$ określonego na przestrzeni zmiennych objaśniających na ogół potrzeba mniejszej liczby eliptycznych funkcji bazowych postaci (9.10) aniżeli jąder gaussowskich z jednym parametrem 'width'. Jednak funkcje eliptyczne postaci (9.10) wymagają estymacji większej liczby parametrów: oprócz centrów trzeba jeszcze estymować elementy macierzy kowariancji $\mathbf{\Sigma}$.

9.8 Przykłady programowania w Matlabie

Przykład 1. Obliczanie wartości funkcji radialnych dla danych $\mathbf{X}_{N \times d}$

```
% Calculate squared norm matrix, of dimension (ndata, ncentres)
n2 = dist2(x, net.c); % kwadraty odleglosci: kazdy x z kazdym c, -> n x H

% Switch on activation function type
switch net.actfn
  case 'gaussian'   % Gaussian, exp{ - || x-c||^2 / (2*sigma*sigma) }
    % Calculate width factors: net.wi contains squared widths
    wi2 = ones(ndata, 1) * (2 .* net.wi);
    % Compute the activations
    z = exp(-(n2./wi2));      % tablica wymiaru ndata x H
  case 'tps'        % Thin plate spline, r^2 log r
    z = n2.*log(n2+(n2==0));
  case 'r4logr'     % r^4 log r
    z = n2.*n2.*log(n2+(n2==0));
  otherwise
    error('Unknown activation function in rbffwd')
end % od switch

y = z*net.w2 + ones(ndata, 1)*net.b2; % wyniki na wyjściu sieci
```

9.9 Rozszerzenia i zastosowania sieci RBF

Pierwotna koncepcja sieci RBF została opisana w pracy autorów: Moody J., Darken CJ.: Fast learning in Networks of Locally-tuned Processing Units. Neural Computation, Vol. 1, 1989, 281–294. Również - ci sami autorzy: Learning with localized receptive fields. in: [Touretzky et al. 1989], pp. 133-143.

Przedstawione na początku tego rozdziału radialne funkcje bazowe są szczególnym przypadkiem rozszerzonej klasy funkcji nazywanych funkcjami jądrowymi (ang. *kernels*, czasami spolszczanych jako *kernele*). Funkcje te mają duże znaczenie w teorii automatycznego uczenia (prace Vapnika i grupy niemieckiej skupionej dookoła Bernarda Scholkopfa). M.inn. kernele są podstawą ciekawej metody SVM (Support Vector Machines) która obecnie uchodzi za jedną z najlepszych metod klasyfikacji danych (i nie tylko).

Radialne funkcje bazowe są istotnym elementach w metodach GTM (Generative Topographic Mapping) i Neuroscale służących do wizualizacji wielozmiennych danych (metody opracowane w neural Research Group Aston, Birmingham).

Do ciekawszych prac opisujących rozszerzenia i zastosowania sieci RBF należą metody Neuroscale i GTM które sa opisane w nastepnych dwóch pod-rozdziałach.

9.9.1 Neuroscale: nieliniowa redukcja wymiarowości i wizualizacja danych

Metoda ta jest opisana w książce Nabney'a [4] i zaimplementowana w pakiecie Netlab jako funkcja o tej samej nazwie `neuroscale`. W pakiecie tym znajduje się również moduł demonstracyjny o nazwie `demns1.m` ilustrujacy działanie tej funkcji. Autorami metody Neuroscale są D. Lowe i M. Tipping z Aston Research Triangle w Birmingham (1997).

9.9.2 Generative Topographic Mapping: inna nieliniowa redukcja wymiarowości i wizualizacja danych

Metoda ta jest jest opisana w książce Nabney'a [4] i zaimplementowana w pakiecie Netlab jako funkcja o nazwie `GTM`. W pakiecie tym znajdują się dwa moduły demonstracyjn o nazwie `demngtm1.m` i `demgtm2.m` ilustrujacye działanie metody GTM. Autorami metody Neuroscale są C.M. Bishop i z Aston Research Triangle w Birmingham (1997).

Literatura

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

[2] S. Osowski, *Sieci neuronowe w ujęciu algorytmicznym.* WNT W-wa 1996.

[3] 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/

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

[5] Robi Polikar, *Pattern Recognition.* Wiley Encyclopedia on Biomedical Engineering. Copyright 2006 Wiley and Sons, Inc. Artykuł przeglądowy, s. 1–22. Manuscript, http://users.rowan.edu/~polikar/RESEARCH/PUBLICATIONS/

[6] M.R. Berthold, J. Diamond: Boosting the performance of RBF networks with dynamic decay adjustment. *NIPS* 1994.

[7] M.R. Berthold, J. Diamond: Constructive training of probabilistic neural networks. *Neurocomputing* 19 (1998), 167–183.

[8] Mei-Fang Zhao, i inn. (+ 4 osoby): Authomated spectral cassification of QSOs and Galaxis by Radial basis Function network with Dynamic Decay Adjustment. *LNCS 3972*, 2008, pp. 361–368, Springer.

[9] Feng Chu, Lipo Wang: Applying RBF NNs to cancer classification. Joint Conf. on NN, IEEE, Vancouver 2006.

[10] Lowe, D. and Tipping, M. E. 1997. NeuroScale: Novel Topographic Feature Extraction using RBF Networks.In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, Cambridge MA: MIT Press 543–549

[11] Bishop, C.M., Svenson, M., and Williams, C.K.I. GTM: The Generative Topographic Mapping. Neural Computation 10 (1999)(1) 215-234.

Dodatek. From Pissarenko on Kolmogorov's Theorem

The information found in this section was taken from [Bishop, 1996, pp. 137– 141]. There is a theorem due to Kolmogorov (Kolmogorov [1957]) which, although of no direct practical significance, does have an interesting relation to neural networks. The theorem has its origins at the end of the nineteenth century when the mathematician Hilbert compiled a list of 23 unsolved problems as a challenge for twentieth century mathematicians (Hilbert [1900]).

Hilbert's thirteenth problem concerns the issue of whether functions of several variables can be represented in terms of superpositions of functions of fewer variables. He conjectured that there exist continuous functions of three variables which cannot be represented as superpositions of functions of two variables. The conjecture was disproved by Arnold (Arnold [1957]).

However, a much more general result was obtained by **Kolmogorov** (Kolmogorov [1957]) who showed that every continuous function of several variables (for a closed and bounded input domain) can be represented as the superposition of a small number of functions of one variable. *Improved versions* of Kolmogorov's theorem have been given by Sprecher (Sprecher [1965]), Kahane (Kahane [1975]) and Lorentz (Lorentz [1976]).

In neural network terms **this theorem says** that *any* continuous *mapping* y(x) from d input variables xi to an output variable y can be represented exactly by a *three-layer neural network* having d(2d + 1) units in the first hidden layer and ([178] APPENDIX G. KOLMOGOROV'S THEOREM [179]) (2d + 1) units in the second hidden layer.

Without saying, how to obtain the appropriate weights (thus of little practical significance), the theorem states that every function can theoretically be approximated by a MLP with two second layers. So it is a well-founded counterargument to the work of Minsky and Paper (Minsky and Papert [1969]) since it shows that multi-layer networks (contrary to single-layer networks addressed by Minsky and Papert) are able to represent all functions very well.

So Kolmogorov's theorem (although developed in times where ANNs were not used widely) contributed to promoting ANN related research to a high degree.

Spis treści