dr inż. Maciej Ławryńczuk
Instytut Automatyki i Informatyki Stosowanej
Politechnika Warszawska

IDENTYFIKACJA REKURENCYJNYCH MODELI NEURONOWYCH TYPU RBF

W pracy omówiono algorytm identyfikacji (*uczenia*) *rekurencyjnych sieci neuronowych typu RBF* (*ang. Radial Basis Function*)*, które mogą być zastosowanie w modelowaniu nieliniowych procesów dynamicznych. W porównaniu z powszechnie stosowanym klasycznym algorytmem wstecznej propagacji błędów, który prowadzi do wyznaczenia modeli służących do predykcji jednokrokowej, proponowany algorytm umożliwia identyfikację predyktorów wielokrokowych. Przedstawiony algorytm wykorzystano do modelowania reaktora chemicznego.*

IDENTIFICATION OF RECURRENT RBF NEURAL MODELS

This paper details an identification (*training*) *algorithm of RBF* (*Radial Basis Function*) *recurrent neural networks which can be used for modelling nonlinear dynamic processes. In comparison with the widely used classical backpropagation algorithm, which leads to one-step ahead predictors, the described one results in many-steps ahead predictors. The algorithm is used for modelling a chemical reactor.*

1. WSTĘP

Warunkiem prawidłowego funkcjonowania zaawansowanych algorytmów regulacji, przede wszystkim algorytmów regulacji predykcyjnej [15, 21, 22], oraz algorytmów diagnostyki [8, 9] jest znajomość dokładnych modeli dynamicznych procesów. Choć teoretycznie można wykorzystać modele fizykochemiczne, z praktycznego punktu widzenia rozwiązanie takie ma kilka wad. Opracowanie modeli fizykochemicznych wymaga dogłębnej wiedzy technologicznej, jest zwykle długotrwałe i kosztowne, poważną trudnością może okazać się konieczność empirycznego wyznaczenia wartości parametrów, np. szybkości reakcji chemicznych. Modele fizykochemiczne są zwykle bardzo złożone (nieliniowe układy równań różniczkowych i algebraicznych), ich bezpośrednie zastosowanie w algorytmach regulacji i diagnostyki nie jest zazwyczaj możliwe, ponieważ podczas ich rozwiązywania mogą wystąpić problemy numeryczne.

W zaawansowanych algorytmach regulacji i diagnostyki bardzo często wykorzystuje się modele neuronowe [7, 8, 9, 14, 16, 18, 19, 22, 23]. Modele neuronowe charakteryzują się bardzo dużą dokładnością. Dowiedziono mianowicie, że są one w stanie aproksymować dowolną funkcję nieliniową z zadaną dokładnością [6]. Co więcej, modele neuronowe, w przeciwieństwie do modeli wykorzystujących zasady logiki rozmytej, mają stosunkowo mało parametrów, nie występuje tzw. zjawisko „przekleństwa wymiarowości", polegające na gwałtownym zwiększaniu ilości parametrów przy wzroście ilości argumentów. Modele neuronowe bezpośrednio obliczają wyjście na podstawie wejść, w przeciwieństwie do modeli fizykochemicznych nie rozwiązuje się cyklicznie układów równań różniczkowych, w których mogą wystąpić problemy numeryczne. Warto podkreślić, że sieci neuronowe mogą być stosunkowo łatwo wykorzystane w różnych odmianach algorytmów regulacji predykcyjnej i diagnostyki.

Modele neuronowe są modelami empirycznymi, ich struktura dobierana jest arbitralnie, natomiast parametry wyznaczane w trakcie procesu identyfikacji (uczenia) przy wykorzystaniu danych zarejestrowanych w trakcie działania procesu. Z punktu widzenia dokładności modelu, a tym samym skuteczności projektowanego następnie algorytmu regulacji lub diagnostyki, kluczowe znaczenie ma algorytm identyfikacji modelu. Zastosowanie klasycznego algorytmu uczenia sieci neuronowej (algorytm wstecznej propagacji błędów) [2, 4, 17, 20] w celu identyfikacji modelu procesu dynamicznego prowadzi do wyznaczenia predyktora jednokrokowego (ARX, ang. Auto Regressive with eXogenous input). Wykorzystanie takiego modelu do predykcji kilka kroków do przodu, co jest niezbędne przy prognozowaniu, w algorytmach regulacji predykcyjnej i w zagadnieniach diagnostyki, jest zwykle problematyczne. Dlatego też warto zastosować uczenie rekurencyjne sieci neuronowej, dzięki czemu wyznacza się predyktor wielokrokowy (OE, ang. Output Error).

W pracy opisano rekurencyjny algorytm uczenia sieci neuronowej typu RBF (ang. Radial Basis Function), która może być następnie wykorzystana jako predyktor wielokrokowy. Algorytm jest uniwersalny, rozważa się proces o wielu wejściach i jednym wyjściu, przy czym ilość wejść procesu nie jest ograniczona. Ilustrację algorytmu stanowi zagadnienie modelowania reaktora chemicznego. Omawiany algorytm podobny jest do algorytmu uczenia rekurencyjnego sieci neuronowej typu perceptronowego przedstawionego w pracy [12]. Warto podkreślić, że zarówno sieci neuronowe perceptronowe jak i RBF są uniwersalnymi aproksymatorami, ale sposób ich działania jest nieco inny. Sieci perceptronowe są globalnymi aproksymatorami, ponieważ funkcja aktywacji neuronów ukrytych zawsze przyjmuje wartość niezerową. W rezultacie, wszystkie neurony ukryte są wykorzystane do wyznaczenia wartości sygnału wyjściowego. W sieciach RBF wykorzystuje się funkcje przyjmujące wartości niezerowe jedynie w pewnej przestrzeni wokół centrów. W rezultacie, sieci RBF są lokalnymi aproksymatorami.

2. STRUKTURA MODELU NEURONOWEGO

W dalszej części pracy omawia się modele procesów o wielu wejściach i jednym wyjściu (MISO, ang. Multi-Input-Single-Output). Argumentami stosowanego do predykcji jednokrokowej modelu typu ARX są wartości sygnałów wejściowych i wyjściowych procesu w poprzednich dyskretnych chwilach próbkowania

$$\hat{y}(k) = f(\boldsymbol{x}(k)) = f(u_1(k-\tau^1),\ldots,u_1(k-n_B^1),\ldots,u_{n_u}(k-\tau^{n_u}),\ldots,u_{n_u}(k-n_B^{n_u}), \\ y(k-1),\ldots,y(k-n_A)) \tag{1}$$

gdzie $\hat{y}(k)$ oznacza predykcję wyjścia w chwili k na podstawie pomiarów prowadzonych do chwili k–1 (włącznie), n_u jest ilością wejść procesu, natomiast stałe τ^n, n_B^n, gdzie $\tau^n \le n_B^n$ dla wszystkich $n = 1,\ldots,n_u$, oraz n_A definiują rząd dynamiki modelu.

Argumentami stosowanego do predykcji wielokrokowej modelu typu OE, oprócz sygnałów wejściowych, są wartości sygnału wyjściowego modelu prognozowane w poprzednich dyskretnych chwilach

$$\hat{y}(k) = f(\boldsymbol{x}(k)) = f(u_1(k-\tau^1),\ldots,u_1(k-n_B^1),\ldots,u_{n_u}(k-\tau^{n_u}),\ldots,u_{n_u}(k-n_B^{n_u}), \\ \hat{y}(k-1),\ldots,\hat{y}(k-n_A)) \tag{2}$$

Strukturę modelu neuronowego typu RBF odpowiadającego równaniu (2) pokazano na rys. 1. W celu ujednolicenia oznaczeń symbolem $x_j(k)$ oznaczono wartość sygnału wejściowego mo-

delu podawanego na *j*-te wejście sieci w chwili *k* ($x_0(k)$=1 dla dowolnego *k* jest polaryzacją warstwy ukrytej). Sposób ponumerowania wejść sieci jest następujący

$$\begin{array}{ll} x_1(k) = u_1(k-\tau^1) & x_{I_u+1}(k) = \hat{y}(k-1) \\ \vdots & \vdots \qquad\qquad k = S,\ldots,P \\ x_{I_u}(k) = u_{n_u}(k-n_B^{n_u}) & x_I(k) = \hat{y}(k-n_A) \end{array} \tag{3}$$

Symbolem *K* oznaczono ilość neuronów ukrytych, natomiast wejścia sieci ponumerowano od 1 do *I*. Ilość wejść modelu, które są zależne od sygnałów sterujących jest równa

$$I_u = \sum_{i=1}^{n_u} \left(n_B^i - \tau^i + 1 \right) \tag{4}$$

natomiast ilość wejść zależnych od sygnałów wyjściowych modelu wynosi n_A. Całkowita ilość węzłów wejściowych jest więc równa

$$I = I_u + n_A = \sum_{i=1}^{n_u} \left(n_B^i - \tau^i + 1 \right) + n_A \tag{5}$$

Symbolem w_i oznaczono wagi warstwy wyjściowej (i=0,...,K). Wyjście modelu wyraża się wzorem

$$\hat{y}(k) = \sum_{i=0}^{K} w_2(i) v_i(k) \tag{6}$$

Wielkość $v_i(k)$ jest stanem wyjść kolejnych neuronów ukrytych ($i = 1,\ldots,K$), natomiast sygnał $v_0(k)$=1 jest polaryzacją warstwy wyjściowej. Neurony ukryte są nieliniowe, realizują one funkcję

$$v_i(k) = \exp(-z_i(k)) \tag{7}$$

gdzie symbolem $z_i(k)$ oznaczono sumę sygnałów wejściowych neuronów ukrytych (i=1,...,K)

$$z_i(k) = \left\| \boldsymbol{x}(k) - \boldsymbol{c}_i \right\|_{\boldsymbol{Q}_i} \tag{8}$$

Dla rozważanego modelu danego wzorem (2) otrzymuje się

$$\begin{aligned} z_i(k) &= \sum_{j=1}^{I} q_{i,j} (x_j(k) - c_{i,j})^2 \\ &= \sum_{j=1}^{I_u} q_{i,j} (x_j(k) - c_{i,j})^2 + \sum_{j=1}^{n_A} q_{i,I_u+j} (\hat{y}(k-j) - c_{i,I_u+j})^2 \end{aligned} \tag{9}$$

gdzie $x_j(k)$ jest uogólnionym wejściem sieci ($j = 0,\ldots,I$) zgodnie z przypisaniem (3). Wektory

$$\boldsymbol{c}_i = \begin{bmatrix} c_{i,1} & \ldots & c_{i,I} \end{bmatrix}^T \tag{10}$$

oraz diagonalne macierze wagowe

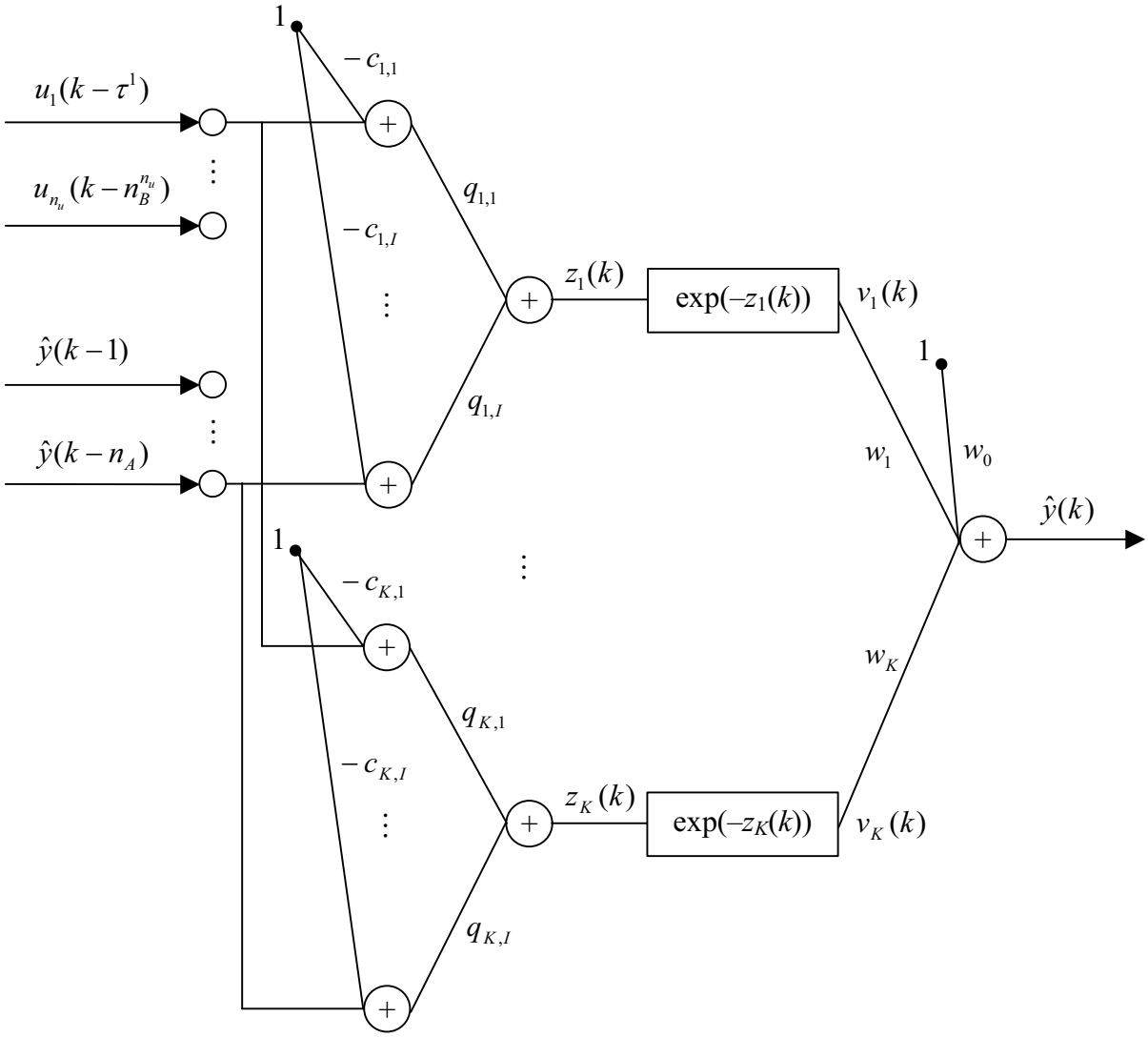
$$\boldsymbol{Q}_i = diag(q_{i,1},\ldots,q_{i,I}) = \begin{bmatrix} q_{i,1} & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & q_{i,I} \end{bmatrix} \tag{11}$$

definiują centra oraz kształty poszczególnych neuronów ukrytych ($i = 1,\ldots,K$).

Korzystając z (6), (7) oraz (9), wyjście modelu można zapisać jako

$$\hat{y}(k) = w_0 + \sum_{i=1}^{K} w_i \exp\left[-\sum_{j=1}^{I_u} q_{i,j}(x_j(k) - c_{i,j})^2 - \sum_{j=1}^{n_A} q_{i,I_u+j}(\hat{y}(k-j) - c_{i,I_u+j})^2 \right] \qquad (12)$$

Zastosowana sieć neuronowa jest nazywana siecią typu HRBF (ang. Hyper Radial Basis Function) [17]. W klasycznych sieciach RBF występuje tylko jedna macierz wagowa $\boldsymbol{Q}$.



Rys. 1. Struktura modelu neuronowego

3. ALGORYTM UCZENIA MODELU NEURONOWEGO

Parametrami modelu neuronowego są wagi w_i, ($i = 0,\ldots,K$), centra $c_{i,j}$ oraz współczynniki wagowe $c_{i,j}$ funkcji przynależności ($i = 1,\ldots,K$, $j = 1,\ldots,I$). Są one wyznaczane w wyniku minimalizacji funkcji jakości modelu (funkcji celu)

$$E(W) = \frac{1}{2}\sum_{k=S}^{P}\left(\hat{y}(k) - d(k)\right)^2 \tag{13}$$

przy czym $\hat{y}(k)$ jest wyjściem modelu (prognozą), $d(k)$ rzeczywistą (żądaną) wartością sygnału wyjściowego (wzorcem uczącym), P jest ilością dostępnych próbek, natomiast

$$S = \max\left(n_B^1,\ldots,n_B^{n_u},n_A\right)+1 \tag{14}$$

Przed rozpoczęciem uczenia modelu należy przygotować zbiór danych w postaci sygnałów wejściowych procesu oraz odpowiadających im żądanych sygnałów wyjściowych $d(k)$.

W trakcie uczenia wykorzystuje się gradientowe metody optymalizacji funkcji celu (13). Gradienty sumaryczne względem parametrów sieci są sumą pochodnych cząstkowych dla kolejnych próbek

$$\begin{aligned}
\frac{dE}{dw_i} &= \sum_{k=S}^{P}\frac{\partial E(k)}{\partial w_i} \quad i = 0,\ldots,K \\
\frac{dE}{dc_{i,j}} &= \sum_{k=S}^{P}\frac{\partial E(k)}{\partial c_{i,j}} \quad i = 1,\ldots,K,\ j = 1,\ldots,I \\
\frac{dE}{dq_{i,j}} &= \sum_{k=S}^{P}\frac{\partial E(k)}{\partial q_{i,j}} \quad i = 1,\ldots,K,\ j = 1,\ldots,I
\end{aligned} \tag{15}$$

Dla pojedynczej próbki otrzymuje się

$$\begin{aligned}
\frac{\partial E(k)}{\partial w_i} &= (\hat{y}(k) - d(k))\frac{\partial \hat{y}(k)}{\partial w_i} \quad i = 0,\ldots,K \\
\frac{\partial E(k)}{\partial c_{i,j}} &= (\hat{y}(k) - d(k))\frac{\partial \hat{y}(k)}{\partial c_{i,j}} \quad i = 1,\ldots,K,\ j = 1,\ldots,I \\
\frac{\partial E(k)}{\partial q_{i,j}} &= (\hat{y}(k) - d(k))\frac{\partial \hat{y}(k)}{\partial q_{i,j}} \quad i = 1,\ldots,K,\ j = 1,\ldots,I
\end{aligned} \tag{16}$$

Wyprowadzenie wzorów na pochodne cząstkowe wyjścia modelu względem jego parametrów rozpoczyna się od wag warstwy wyjściowej

$$\frac{\partial \hat{y}(k)}{\partial w_i} = \sum_{n=1}^{K} w_n \frac{\partial v_n(k)}{\partial w_i} + v_i(k) \quad i = 0,\ldots,K \tag{17}$$

Pochodna sygnału wyjściowego neuronów ukrytych względem wag jest równa

$$\frac{\partial v_n(k)}{\partial w_i} = -\exp(-z_n(k))\frac{\partial z_n(k)}{\partial w_i} \quad i = 0,\ldots,K, n = 1,\ldots,K \tag{18}$$

Pochodna sumarycznego sygnału warstwy ukrytej jest następująca

$$\frac{\partial z_n(k)}{\partial w_i} = 2\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}\left(\hat{y}(k-k_0) - c_{n,I_u+k_0}\right)\frac{\partial \hat{y}(k-k_0)}{\partial w_i} \quad i = 0,\ldots,K, n = 1,\ldots,K \tag{19}$$

Powyższy wzór zawiera pochodną sygnału wyjściowego modelu uwzględnianą z przesunięciem czasowym $k_0 = 1, ..., n_a$. Oznacza to więc, że błąd (13) dla chwili k zależy nie tylko od odpowiedzi modelu na aktualne sygnały wejściowe, lecz również od błędów w przeszłości (dokładnie do chwili S). Korzystając ze wzorów (15), (16), (17), (18) oraz (19)

$$\frac{dE}{dw_i} = \sum_{k=S}^{P}(d(k)-\hat{y}(k))\times \\ \times\left[2\sum_{n=1}^{K} w_n \exp(-z_n(k))\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}(\hat{y}(k-k_0)-c_{n,I_u+k_0})\frac{\partial\hat{y}(k-k_0)}{\partial w_i} - v_i(k)\right] \tag{20}$$

dla wszystkich $i = 0,\ldots,K$.

Obliczając pochodne względem centrów $c_{i,j}$ otrzymuje się

$$\frac{\partial\hat{y}(k)}{\partial c_{i,j}} = \sum_{n=1}^{K} w_n \frac{\partial v_n(k)}{\partial c_{i,j}} \quad i=1,\ldots,K,\ j=1,\ldots,I \tag{21}$$

Pochodna sygnału wyjściowego neuronów ukrytych względem centrów ma postać

$$\frac{\partial v_n(k)}{\partial c_{i,j}} = -\exp(-z_n(k))\frac{\partial z_n(k)}{\partial c_{i,j}} \quad i=1,\ldots,K,\ j=1,\ldots,I,\ n=1,\ldots,K \tag{22}$$

Pochodna sygnału wejściowego neuronów ukrytych jest następująca

$$\frac{\partial z_n(k)}{\partial c_{i,j}} = \begin{cases} 2\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}(\hat{y}(k-k_0)-c_{n,I_u+k_0})\frac{\partial\hat{y}(k-k_0)}{\partial c_{i,j}} - 2q_{n,j}(x_j(k)-c_{n,j}) & n=i \\ 2\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}(\hat{y}(k-k_0)-c_{n,I_u+k_0})\frac{\partial\hat{y}(k-k_0)}{\partial c_{i,j}} & n\neq i \end{cases} \\ i=1,\ldots,K,\ j=1,\ldots,I,\ n=1,\ldots,K \tag{23}$$

co można zapisać w zwięzłej postaci

$$\frac{\partial z_n(k)}{\partial c_{i,j}} = 2\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}(\hat{y}(k-k_0)-c_{n,I_u+k_0})\frac{\partial\hat{y}(k-k_0)}{\partial c_{i,j}} - 2\delta_{n,i}q_{n,j}(x_j(k)-c_{n,j}) \tag{24}$$

gdzie

$$\delta_{n,i} = \begin{cases} 1 & n=i \\ 0 & n\neq i \end{cases} \tag{25}$$

Korzystając ze wzorów (15), (16), (21), (22) oraz (24)

(26)

$$\frac{dE}{dc_{i,j}} = 2\sum_{k=S}^{P}(d(k)-\hat{y}(k))\times \\ \times\left\{\sum_{n=1}^{K} w_n \exp(-z_n(k))\left[\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}(\hat{y}(k-k_0)-c_{n,I_u+k_0})\frac{\partial\hat{y}(k-k_0)}{\partial c_{i,j}} - \delta_{n,i}q_{n,j}(x_j(k)-c_{n,j})\right]\right\}$$

dla wszystkich $i = 1,\ldots,K$, $j = 1,\ldots,I$.

Pochodne względem parametrów $q_{i,j}$ oblicza się podobnie jak pochodne względem centrów $c_{i,j.}$ Otrzymuje się

$$\frac{\partial \hat{y}(k)}{\partial q_{i,j}} = \sum_{n=1}^{K} w_n \frac{\partial v_n(k)}{\partial q_{i,j}} \quad i = 1,\ldots,K,\ j = 1,\ldots,I \tag{27}$$

Pochodna sygnału wyjściowego neuronów ukrytych ma postać

$$\frac{\partial v_n(k)}{\partial q_{i,j}} = -\exp(-z_n(k))\frac{\partial z_n(k)}{\partial q_{i,j}} \quad i = 1,\ldots,K,\ j = 1,\ldots,I, n = 1,\ldots,K \tag{28}$$

Pochodna sygnału wejściowego neuronów ukrytych jest następująca

$$\frac{\partial z_n(k)}{\partial q_{i,j}} = \begin{cases} 2\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}(\hat{y}(k-k_0) - c_{n,I_u+k_0})\frac{\partial \hat{y}(k-k_0)}{\partial q_{i,j}} + (x_j(k) - c_{n,j})^2 & n = i \\ 2\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}(\hat{y}(k-k_0) - c_{n,I_u+k_0})\frac{\partial \hat{y}(k-k_0)}{\partial q_{i,j}} & n \neq i \end{cases} \tag{29}$$

$$i = 1,\ldots,K,\ j = 1,\ldots,I, n = 1,\ldots,K$$

co można zapisać w zwięzłej postaci

$$\frac{\partial z_n(k)}{\partial q_{i,j}} = 2\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}(\hat{y}(k-k_0) - c_{n,I_u+k_0})\frac{\partial \hat{y}(k-k_0)}{\partial q_{i,j}} - \delta_{n,i}(x_j(k) - c_{n,j})^2 \tag{30}$$

Korzystając ze wzorów (15), (16), (27), (28) oraz (30)

(31)

$$\frac{dE}{dc_{i,j}} = \sum_{k=S}^{P}(d(k) - \hat{y}(k)) \times$$

$$\times \left\{ \sum_{n=1}^{K} w_n \exp(-z_n(k)) \left[2\sum_{k_0=1}^{n_A} q_{n,I_u+k_0}(\hat{y}(k-k_0) - c_{n,I_u+k_0})\frac{\partial \hat{y}(k-k_0)}{\partial q_{i,j}} + \delta_{n,i}(x_j(k) - c_{n,j})^2 \right] \right\}$$

dla wszystkich $i = 1,\ldots,K$, $j = 1,\ldots,I$.

Z uwagi na rekurencyjny sposób prowadzenia obliczeń wszystkie pochodne sygnału wyjściowego do chwili S–1 włącznie są zerowe

$$\begin{aligned} &\frac{\partial \hat{y}(k-k_0)}{\partial w_i} = 0 \quad k_0 = 1,\ldots,S-1, i = 0,\ldots,K \\ &\frac{\partial \hat{y}(k-k_0)}{\partial c_{i,j}} = 0 \quad k_0 = 1,\ldots,S-1, i = 1,\ldots,K, j = 1,\ldots,I \\ &\frac{\partial \hat{y}(k-k_0)}{\partial q_{i,j}} = 0 \quad k_0 = 1,\ldots,S-1, i = 1,\ldots,K, j = 1,\ldots,I \end{aligned} \tag{32}$$

Niech $g(W)$ oznacza wektor zawierający wszystkie gradienty dane wzorami (15)

$$g(W) = \left[\frac{dE}{dw_0},\ldots\frac{dE}{dw_K},\ \frac{dE}{dc_{1,1}}\ldots\frac{dE}{dc_{K,I}},\frac{dE}{dq_{1,1}}\ldots\frac{dE}{dq_{K,I}} \right]^T \tag{33}$$

Dla n-tej iteracji algorytmu minimalizacji funkcji celu (13) aktualizacja parametrów modelu odbywa się zgodnie ze wzorem

$$W_{n+1} = W_n + \eta_n p(W_n) \tag{34}$$

gdzie $p(W_n)$ jest kierunkiem poszukiwań, natomiast η_n jest krokiem, którego wartość wyznaczana jest w wyniku optymalizacji kierunkowej, wykonywanej np. metodą złotego podziału [1, 3]. Sposób wyznaczania kierunku zależy od metody optymalizacji [1, 3]. W przypadku najprostszej (lecz jednocześnie najbardziej zawodnej) metody najszybszego spadku jest on przeciwny do kierunku gradientu, tzn. $p(W_n) = -g(W_n)$. W efektywnych metodach zmiennej metryki kierunek wyznaczany jest na podstawie gradientu $g(W_n)$ oraz macierzy hesjanu $H(W_n)$

$$p(W_n) = -(H(W_n))^{-1} g(W_n) \tag{35}$$

Aproksymacja macierzy drugich pochodnych wyznaczana jest w sposób iteracyjny. W bardzo skutecznym algorytmie BFGS oblicza się odwrotność hesjanu $V_n = (H(W_n))^{-1}$ ze wzoru

$$V_n = V_{n-1} + \left[1 + \frac{r_n^T V_{n-1} r_n}{s_n^T r_n}\right] \frac{s_n s_n^T}{s_n^T r_n} - \frac{s_n r_n^T V_{n-1} + V_{n-1} r_n s_n^T}{s_n^T r_n} \tag{36}$$

gdzie $s_n = W_n - W_{n-1}$, $r_n = g_n - g_{n-1}$.

Algorytm uczenia modelu neuronowego jest następujący:

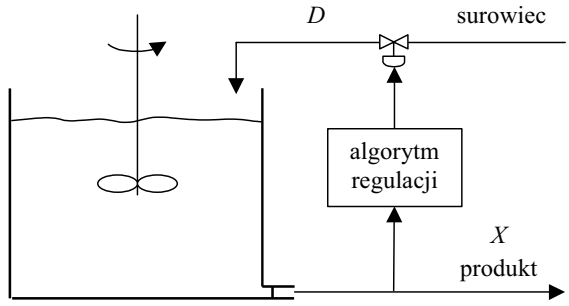
1. Nadanie wagom sieci w_i wartości początkowych (np. losowych), nadanie parametrom $c_{i,j}$ oraz $q_{i,j}$ wartości początkowych.
2. Wyzerowanie pochodnych sygnału wyjściowego do chwili S–1 – wzory (32), inicjalizacja odwrotności hesjanu V_n=I, ustawienie licznika iteracji n=0.
3. Obliczenie sumarycznych gradientów dla całego zakresu czasu k – wzory (20), (26) i (31), wpisanie tych wartości do wektora $g(W_n)$ – wzór (33).
4. Wyznaczenie odwrotności hesjanu V_n – wzór (36).
5. Wyznaczenie kierunku optymalizacji $p(W_n)$ – wzór (35).
6. Minimalizacja kierunkowa – wyznaczenie kroku η_n.
7. Aktualizacja parametrów modelu – wzór (34).
8. Wyznaczenie aktualnej wartości funkcji celu (13).
9. Jeżeli spełnione kryterium stopu (np. norma gradientu $g(W_n)$ jest mała) – koniec obliczeń, w przeciwnym wypadku n=n+1, przejście do punktu 3.

O ile celowa jest losowa inicjalizacja wag w_i sieci neuronowej, to parametry $c_{i,j}$ oraz $q_{i,j}$ powinny być dobierane. Najłatwiej przyjąć równomierne rozłożenie centrów w całej dziedzinie argumentów modelu oraz takie same współczynniki skalujące $q_{i,j}$. W najprostszej wersji algorytmu można przyjąć, że adaptacji (uczeniu) podlegają jedynie wagi, natomiast parametry $c_{i,j}$ oraz $q_{i,j}$ są ustalane raz, w trakcie inicjalizacji modelu i nie podlegają uczeniu.

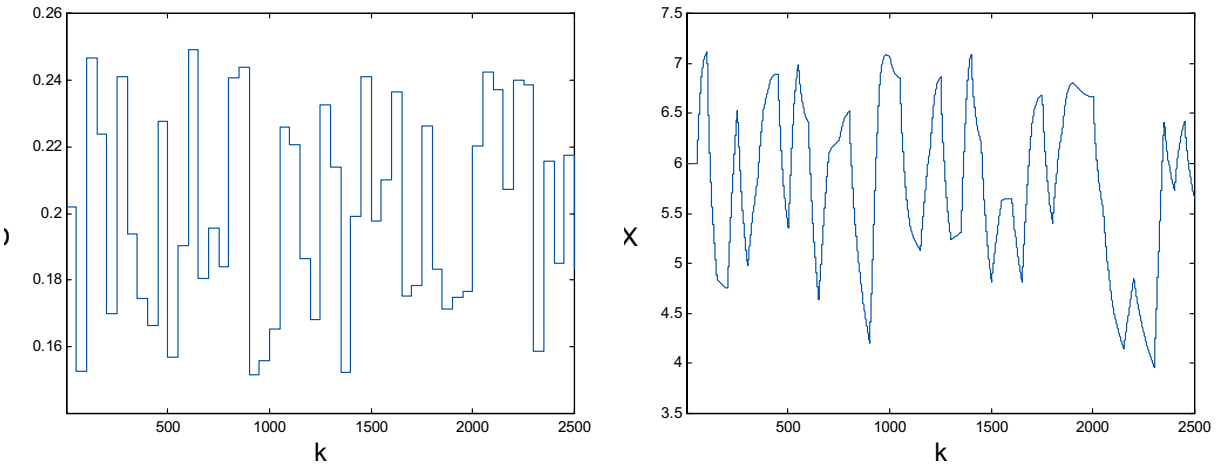
Dokładność aproksymacji macierzy odwrotnej hesjanu może ulec pogorszeniu w trakcie trwania obliczeń. Dlatego też zaleca się cykliczną odnowę kierunku, tzn. V_n=I. Okres odnowy ustalany jest eksperymentalnie, wynosi on zwykle od kilkunastu do kilkudziesięciu iteracji.

4. WYNIKI EKSPERYMENTÓW

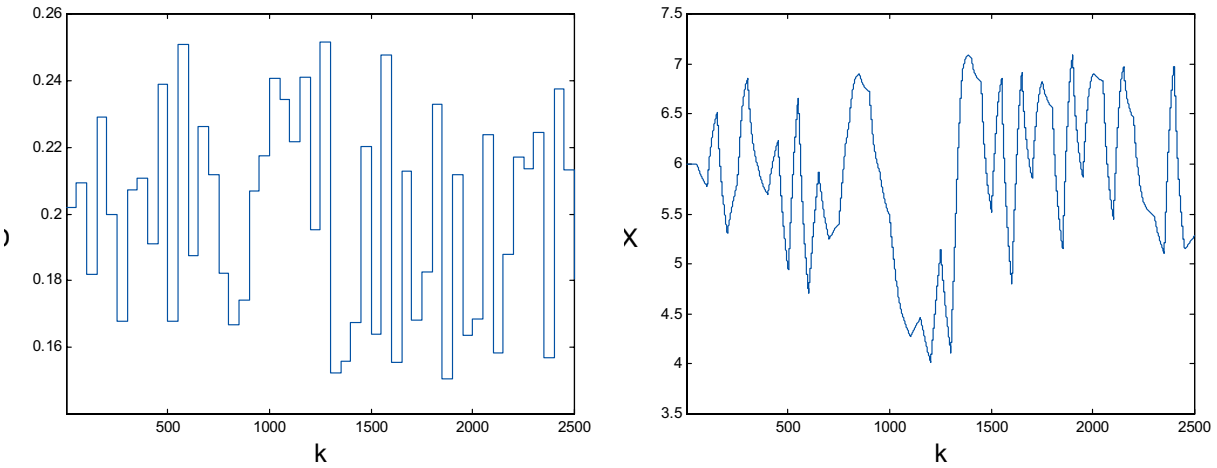
Rozważanym procesem dynamicznym jest pokazany na rys. 2 reaktor chemiczny opisany w pracy [5]. Zmienną wejściową (manipulowaną) jest natężenie przepływu D rozpuszczalnika (głównego surowca), zmienną wyjściową (regulowaną) jest skład produktu X.



Rys. 2. Reaktor chemiczny



Rys. 3. Zestaw danych uczących



Rys. 4. Zestaw danych testowych

Argumenty modelu są ustalone, ma on następującą strukturę

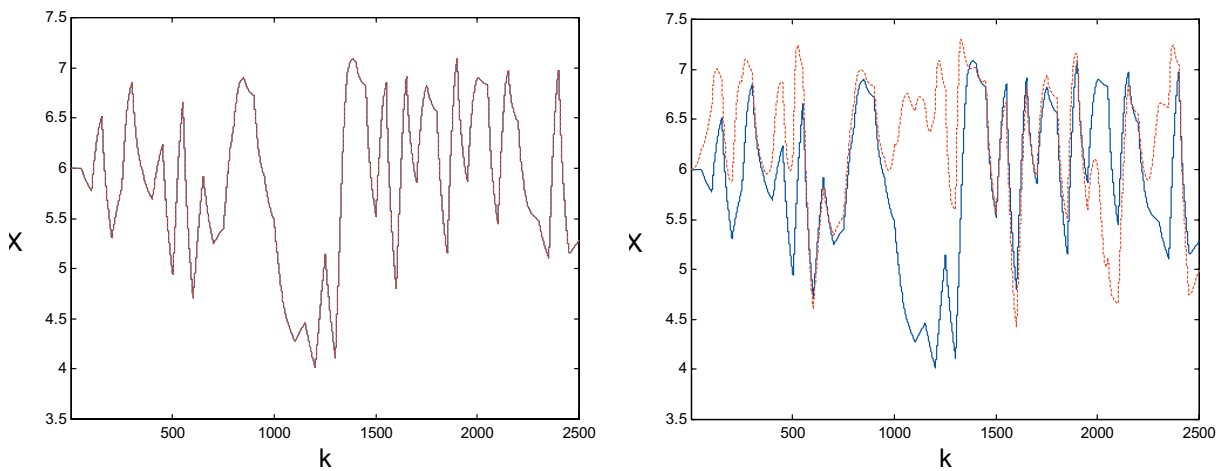
$$\hat{y}(k) = f(u(k-1), u(k-2), \hat{y}(k-1), \hat{y}(k-2)) \tag{37}$$

gdzie $u = D$, $y = X$. Do celów uczenia i testowania sieci wykorzystano pokazane na rys. 3 i rys. 4 dwa zestawy danych zawierające po 2500 próbek (okres próbkowania 10 min).

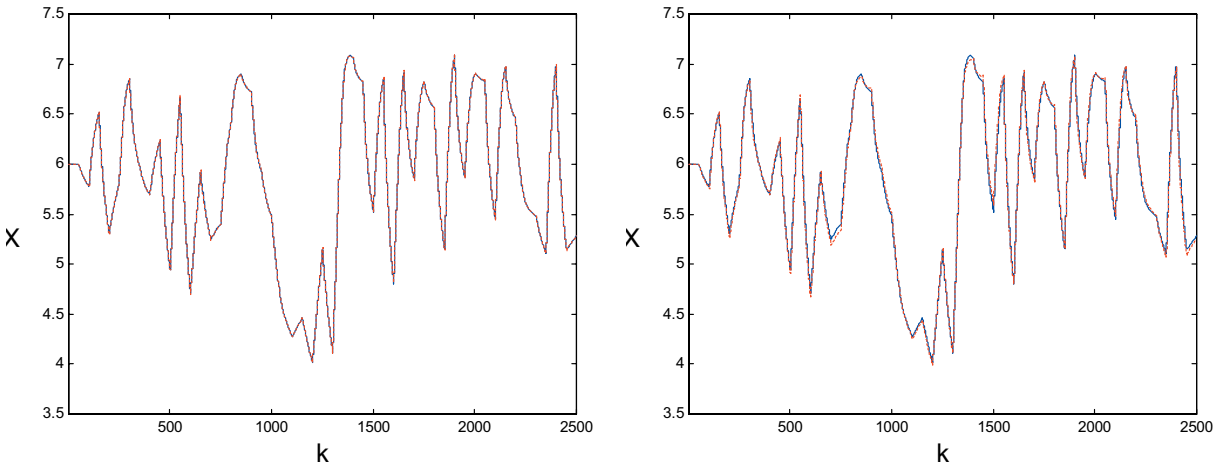
W tabeli 1 porównano wskaźniki jakości modelu, przy czym jest on uczony w dwu trybach: jako predyktor jednokrokowy (ARX) oraz wielokrokowy (OE). Model jest testowany zarówno w trybie ARX i OE. Przeprowadzono 10 cykli uczących w każdym z trybów, zaprezentowano najlepsze uzyskane rezultaty. Na rys. 5 pokazano test modelu (tzn. wyjście modelu na tle danych testowych) uczonego w trybie ARX. Oczywiście, działa on bardzo dobrze w trybie predykcji jednokrokowej ARX (E_{ARX}=4,52·10^{-5}), natomiast jego zastosowanie jako predyktor wielokrokowy OE kończy się niepowodzeniem (E_{OE}=2,09·10^{3}). W przeciwieństwie do tego, model uczony w trybie OE umożliwia nie tylko predykcję jednokrokową (E_{ARX} = 1,45·10^{-1}), lecz również jednokrokową (E_{OE}=2,95·10^{0}) co pokazano na rys. 6. Na rys. 7 porównano efektywność uczenia modelu OE algorytmem zmiennej metryki BFGS oraz najszybszego spadku. Algorytm najszybszego spadku, wbrew swojej nazwie, jest bardzo wolno zbieżny, istnieje duże prawdopodobieństwo utknięcia procedury optymalizacji w minimum lokalnym.

Tabela 1. Porównanie dokładności modeli

tryb uczenia	dane uczące		dane testowe	
	E_{ARX}	E_{OE}	E_{ARX}	E_{OE}
ARX	4,79·10^{-5}	1,65·10^{3}	4,52·10^{-5}	2,09·10^{3}
OE	1,29·10^{-1}	2,37·10^{0}	1,45·10^{-1}	2,95·10^{0}



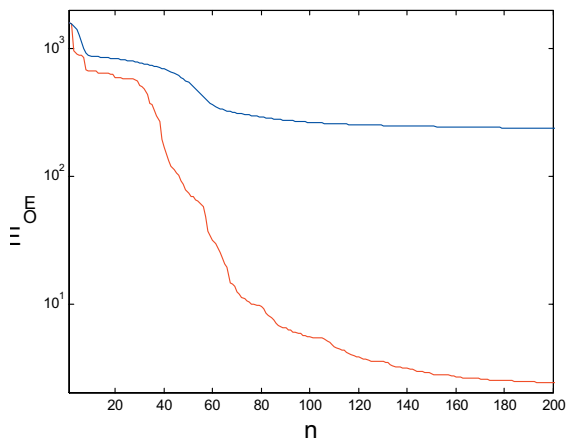
Rys. 5. Dane testowe oraz wyjście modelu uczonego w trybie ARX: test w trybie ARX (po lewej stronie), test w trybie OE (po prawej stronie)



Rys. 6. Dane testowe oraz wyjście modelu uczonego w trybie OE: test w trybie ARX (po lewej stronie), test w trybie OE (po prawej stronie)

5. PODSUMOWANIE

W pracy omówiono algorytm identyfikacji rekurencyjnych sieci neuronowych typu RBF. W przeciwieństwie do klasycznego algorytmu wstecznej propagacji błędów, który prowadzi do wyznaczenia modeli pozwalających na predykcję jednokrokową, algorytm ten umożliwia identyfikację predyktorów wielokrokowych. Otrzymane modele mogą być zastosowane do celów prognozowania, w algorytmach regulacji predykcyjnej oraz w diagnostyce procesów. W pracy [11] przedstawiono nieliniowy algorytm regulacji predykcyjnej procesu polimeryzacji wykorzystujący model neuronowy typu RBF, do identyfikacji którego zastosowano omówiony w niniejszej pracy algorytm.



Rys. 7. Uczenie modelu trybie OE: algorytm BFGS oraz najszybszego spadku

Warto podkreślić, że w przypadku ustalenia parametrów opisujących nieliniowe funkcje sieci (wielkości $c_{i,j}$ oraz $q_{i,j}$), uczenie jest stosunkowo proste, gdyż adaptowane są wyłącznie wagi w_i. Najwygodniej przyjąć równomierne rozłożenie centrów w całej dziedzinie argumentów modelu, oraz takie same współczynniki skalujące $q_{i,j}$. W opisanej sytuacji uczenie sieci RBF jest znacznie bardziej efektywne niż uczenie sieci neuronowej typu perceptronowego, w której wszystkie wagi inicjalizowane są w sposób losowy.

LITERATURA

[1] M. S. Bazaraa, J. Sherali, K. Shetty (1993): *Nonlinear programming: theory and algorithms*. John Wiley & Sons.

[2] C. M. Bishop (1995): *Neural networks for pattern recognition*. Oxford University Press.

[3] P. Gill, W. Murray, M. Wright (1981): *Practical optimization*. Academic Press.

[4] S. Haykin (1999): *Neural networks – a comprehensive foundation*. Prentice Hall.

[5] M. A. Henson, D. E. Seborg (1991): An internal model control strategy fornonlinear systems. *AIChE Journal*, tom 37, nr 7, s. 1065-1081.

[6] K. Hornik, M. Stinchcombe, H. White (1989): Multilayer feedforward networks are universal approximators. *Neural networks*, tom 2, s. 359–366.

[7] M. A. Hussain (1999): Review of the applications of neural networks in chemical process control – simulation and online implementation. *Artificial Intelligence in Engineering*, tom 13, s. 55-68.

[8] J. Korbicz (red.) (2002): *Diagnostyka procesów: modele, metody sztucznej inteligencji, zastosowanie*. WNT.

[9] J. M. Kościelny (2001): *Diagnostyka zautomatyzowanych procesów przemysłowych*. EXIT.

[10] M. Ławryńczuk (2007): A family of model predictive control algorithms with artificial neural networks. *International Journal of Applied Mathematics and Computer Science*, tom 17, nr 2, s. 217-232.

[11] M. Ławryńczuk, P. Tatjewski: A computationally efficient nonlinear predictive control algorithm with RBF neural models and its application. *Lecture Notes in Artificial Intelligence, vol. 4585, Springer: The International Conference Rough Sets and Emerging Intelligent Systems Paradigms, RSEISP 2007*, Warszawa, Poland, 2007, pp. 603-612.

[12] M. Ławryńczuk (2007): Rekurencyjne sieci neuronowe w modelowaniu nieliniowych procesów dynamicznych. *Pomiary Automatyka Robotyka*, nr 2, CD-ROM: Konferencja Automation 2007.

[13] M. Ławryńczuk, P. Tatjewski (2006): An efficient nonlinear predictive control algorithm with neural models and its application to a high-purity distillation process. *Lecture notes in Artificial Intelligence, tom 4029, Springer: The Eighth International Conference on Artificial Intelligence and Soft Computing ICAISC 2006*, Zakopane, s. 76-85.

[14] M. Ławryńczuk (2000): Wykorzystanie sieci neuronowych do modelowania silnie nieliniowych procesów. *Automation 2000*, Warszawa, s. 293-300.

[15] J. M. Maciejowski (2002): *Predictive control with constraints*. Prentice Hall, Harlow.

[16] M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen (2000): *Neural networks for modelling and control of dynamic systems*. Springer.

[17] S. Osowski (1996): *Sieci neuronowe w ujęciu algorytmicznym*. WNT.

[18] S. Piche, B. Sayyar-Rodsari, D. Johnson, M. Gerules (2000): Nonlinear model predictive control using neural networks. *IEEE Control Systems Magazine*, tom 20, nr 3, s. 56-62.

[19] M. Pottmann, D. E. Seborg (1997): A nonlinear predictive control strategy based on radial basis function networks. *Computers and Chemical Engineering*, tom 21, nr 9, s. 965-980.

[20] B. D. Ripley (1996): *Pattern recognition and neural networks*. Cambridge University Press.

[21] J. A. Rossiter (2003): *Model-based predictive control*. CRC Press, Boca Raton.

[22] P. Tatjewski (2007): *Advanced control of industrial processes. Structures and algorithms*. Springer.

[23] P. Tatjewski, M. Ławryńczuk (2006): Soft computing in model-based predictive control. *International Journal of Applied Mathematics and Computer Science*, tom 16, nr 1, s. 101-120.