Krzysztof BARTECKI, Marek CZORNY
POLITECHNIKA OPOLSKA, INSTYTUT AUTOMATYKI I INFORMATYKI
ul. Sosnkowskiego 31, 45-272 Opole

Implementacja sztucznej sieci neuronowej w architekturze równoległej z wykorzystaniem protokołu MPI

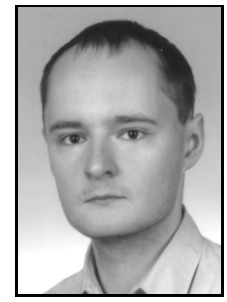
Dr inż. Krzysztof BARTECKI

Stopień naukowy doktora nauk technicznych uzyskał w 2004 roku w Politechnice Opolskiej. Pracuje w Instytucie Automatyki i Informatyki Politechniki Opolskiej na stanowisku adiunkta. Główny kierunek badań naukowych obejmuje zastosowanie sztucznych sieci neuronowych w zagadnieniach modelowania, identyfikacji oraz sterowania obiektami dynamicznymi.

e-mail: k.bartecki@po.opole.pl

Mgr inż. Marek CZORNY

Tytuł zawodowy magistra inżyniera informatyka uzyskał w 2008 roku na Wydziale Elektrotechniki, Automatyki i Informatyki Politechniki Opolskiej. Aktualnie zatrudniony jest jako programista w firmie LGBS Polska Sp. z o.o.



Streszczenie

W artykule wskazano na pewne aspekty związane z implementacją jednokierunkowej sieci neuronowej w architekturze równoległej z wykorzystaniem standardu przesyłania komunikatów MPI. Zaprezentowany przykład zastosowania sieci dotyczy klasycznego problemu aproksymacji funkcji. Zbadano wpływ liczby uruchamianych procesów na efektywność procedury uczenia i działania sieci oraz zademonstrowano negatywny wpływ opóźnień powstałych przy przesyłaniu danych za pomocą sieci LAN.

Słowa kluczowe: sztuczna sieć neuronowa, architektura równoległa, aproksymacja funkcji.

Parallel implementation of artificial neural network with use of MPI protocol

Abstract

In the paper some characteristic features concerning feed-forward neural network implementation in parallel computer architecture using MPI communication protocol are investigated. Two fundamental methods of neural network parallelization are described: neural (Fig. 1) as well as synaptic parallelization (Fig. 2). Based on the presented methods, an original application implementing feed-forward multilayer neural network was built. The application includes: a Java runtime interface (Fig. 3) and a computational module based on the MPI communication protocol. The simulation tests consisted in neural network application to classical problem of nonlinear function approximation. Effect of the number of processes on the network learning efficiency was examined (Fig. 4, Tab. 1). The negative effect of transmission time delays in the LAN is also demonstrated in the paper. The authors conclude that computational advantages of neural networks parallelization on a heterogeneous cluster consisting of several personal computers will become apparent only in the case of very complex neural networks, composed of many thousands of neurons.

Keywords: artificial neural network, parallel architecture, function approximation.

1. Wprowadzenie

Sztuczne sieci neuronowe z definicji wykonują swoje obliczenia w sposób równoległy. Każdy neuron stanowi prostą jednostkę obliczeniową, działającą równocześnie z innymi neuronami wchodzącymi w skład sieci. Operacje wykonywane przez pojedyncze neurony nie wymagają dużych mocy obliczeniowych, ani samodzielnie nie mają większego zastosowania. Siła obliczeniowa sieci neuronowych wynika zatem głównie z współdziałania większej liczby neuronów [5, 8, 11, 14].

Powszechnie stosowaną techniką implementacji sieci neuronowych jest symulacja ich działania na komputerach klasy PC. Komputery te obecnie wyposażone są w procesory dwu-, cztero- i sześciordzeniowe, które w pewnym zakresie pozwalają na zrównoleglenie obliczeń neuronowych.

Jednak aby zasymulować większą sieć, wskazane byłoby posiadanie większej liczby procesorów, z których każdy pełniłby np. rolę jej pojedynczego neuronu. Pewnym rozwiązaniem może być tutaj zastosowanie klastera komputerowego. Połączenie przy pomocy sieci wielu jednostek obliczeniowych, między którymi przesyłane są informacje, powinno umożliwić realizację rozbudowanej sieci neuronowej [6, 10, 13, 15].

Jednak wymagany jest tu pewien dodatkowy element, dzięki któremu obliczenia będą przebiegały w sposób synchroniczny, umożliwiający efektywną komunikację pomiędzy procesami (np. neuronami sieci). Role takiego elementu pełni odpowiedni protokół komunikacyjny, definiujący sposób przesyłania komunikatów pomiędzy procesami programów równoległych.

W artykule omówiono realizację jednokierunkowej sieci neuronowej, pracującej w architekturze równoległej na prostym klasterze złożonym z komputerów klasy PC, połączonych siecią lokalną, z wykorzystaniem protokołu komunikacyjnego MPI. Działanie utworzonej w ten sposób sieci neuronowej przetestowano na przykładzie klasycznego zadania aproksymacji funkcji [1, 4]. Zbadano m.in. wpływ liczby uruchamianych procesów na efektywność uczenia i działania sieci oraz zademonstrowano negatywny wpływ opóźnień powstałych przy przesyłaniu danych za pomocą sieci LAN [3].

2. Protokół MPI

MPI (ang. *Message Passing Interface*) jest standardem przesyłania informacji w programach wykorzystujących przetwarzanie równoległe. Dzięki mechanizmowi przesyłania komunikatów możliwa jest komunikacja między procesami uruchomionymi na różnych jednostkach obliczeniowych. Standard MPI jest przenośny – został zaimplementowany dla każdej architektury opartej o rozproszoną pamięć. Inną zaletą jest jego prędkość – każda implementacja jest optymalizowana pod kątem sprzętu, na którym zostanie uruchomiona. MPI może być on stosowany zarówno w specjalizowanych komputerach równoległych (superkomputerach), jak też w sieciach heterogonicznych, złożonych z różnorodnych komputerów (klasterach) [7, 10, 13].

MPI posiada wiele implementacji. Wielu dostawców sprzętu tworzy swoje własne implementacje, przygotowane do działania w określonej konfiguracji sprzętowej, dzięki czemu mogą one na takim sprzęcie uzyskiwać maksymalną wydajność. Dostępne są również darmowe implementacje ogólnego przeznaczenia. Wśród nich można wymienić m.in. [3]:

- LAM/MPI,
- *Open* MPI,
- MPICH.

Najpopularniejszą wśród wyżej wymienionych implementacji jest biblioteka MPICH. Jest ona darmowa, można ją pobrać ze strony domowej jej twórców, dostępne są tam również jej kody źródłowe. Biblioteka ta jest ciągle udoskonalana – co pewien czas

wychodzą jej nowe wersje, dzięki czemu można mieć pewność, że dysponuje się sprawdzoną, bezpieczną i poprawnie działającą implementacją. Z tych powodów zdecydowano się wykorzystać tę właśnie implementację standardu MPI do zrealizowania sztucznej sieci neuronowej, działającej w architekturze równoległej.

3. Metody zrównoleglenia sieci neuronowej

Jak wcześniej wspomniano, symulowanie sieci neuronowej na klasycznym komputerze szeregowym, wyposażonym w pojedynczą jednostkę obliczeniową, wiąże się z rezygnacją z jednej z najważniejszych cech tej sieci, jaką jest jej równoległe działanie. Jednak odrębną i o wiele bardziej istotną kwestię, głównie ze względu na znacznie większy nakład obliczeniowy, stanowi zrównoleglenie procesu uczenia sieci. W literaturze można spotkać wiele sposobów zrównoleglenia – zarówno działania, jak i uczenia sieci neuronowych [3, 6, 10, 13, 15]. Niektóre z nich zostaną omówione w kolejnych podpunktach.

3.1. Równoległość sesji treningowej

Równoległość sesji treningowej polega na uruchomieniu na każdej jednostce obliczeniowej procedury uczenia niezależnej sieci neuronowej o takiej samej strukturze, lecz różniącej się np. początkowymi wartościami współczynników wagowych, wartością współczynnika prędkości uczenia oraz współczynnika *momentum*.

Ucząc jednocześnie kilka sieci z różnymi wagami początkowymi oraz parametrami uczenia zwiększamy szansę, że podczas uczenia jednej z nich zostanie znalezione globalne minimum funkcji błędu. W metodzie tej nie jest zwiększana szybkość uczenia ani działania sieci, lecz jedynie prawdopodobieństwo uzyskania założonej wartości błędu średniokwadratowego.

3.2. Zrównoleglenie danych treningowych

W tej metodzie na każdej jednostce obliczeniowej uczona jest dokładna kopia tej samej sieci, z takimi samymi wartościami początkowymi współczynników wagowych oraz wartościami parametrów treningowych. Każdej z tych kopii podawany jest jednak inny podzbiór zbioru danych uczących. Modyfikacje wartości współczynników wagowych poszczególnych neuronów wyznaczane są lokalnie dla każdej sieci w trakcie jej uczenia. Zmiany te są następnie globalnie sumowane i ostatecznie zmieniane są we wszystkich lokalnych kopiach sieci.

Sposób ten powinien przynieść znaczne skrócenie czasu uczenia – sieć neuronowa jest uczona jednocześnie na podstawie wielu wzorców uczących. Jednak, podobnie jak w przypadku zrównoleglenia sesji treningowej, również tutaj nie występuje zrównoleglenie działania sztucznej sieci neuronowej.

3.3. Potokowość

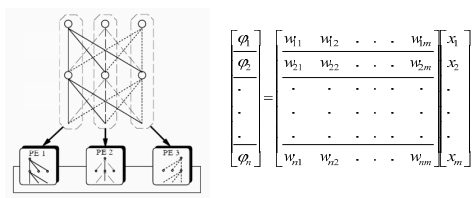
Idea potokowości polega na przetwarzaniu poszczególnych warstw jednokierunkowej sieci neuronowej na różnych jednostkach obliczeniowych. Do pierwszej jednostki podawany jest zestaw danych uczących – realizuje ona działanie pierwszej warstwy ukrytej. Po wyznaczeniu wszystkich sygnałów wyjściowych tej warstwy, wyniki przekazywane są do drugiej jednostki obliczeniowej, realizującej działanie warstwy wyjściowej. Jednocześnie do pierwszej jednostki podawane są kolejne dane uczące, które przetwarzane są równolegle do wcześniejszych danych uczących, przetwarzanych w tej samej chwili przez druga jednostkę obliczeniową. Gdy druga jednostka skończy obliczenia dla warstwy wyjściowej, wyznaczany jest błąd działania sieci. Na podstawie tego błędu obliczane są przez drugą jednostkę poprawki wag neuronów warstwy wyjściowej. Następnie błąd jest przekazywany, zgodnie z algorytmem propagacji wstecznej, do warstwy ukrytej, gdzie następuje modyfikacja współczynników wagowych neuronów [7, 8].

Równoległość jest w tej metodzie ograniczona liczbą warstw sieci – najczęściej posiadają one tylko jedną, nieliniową warstwę ukrytą oraz warstwę wyjściową [1, 2, 4]. W takim przypadku wykorzystywane są maksymalnie dwie jednostki obliczeniowe, podczas gdy idea działania sztucznych sieci neuronowych polega na zrównolegleniu działania poszczególnych neuronów sieci.

3.4. Zrównoleglenie węzłów

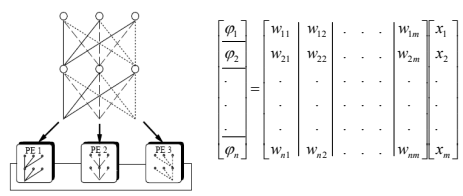
Metoda zrównoleglenia węzłów w największym stopniu, spośród wcześniej wymienionych, wykorzystuje naturalną dla sieci neuronowych równoległość. Można wyróżnić dwie jej odmiany: zrównoleglenie neuronowe oraz zrównoleglenie synaptyczne.

W przypadku zrównoleglenia neuronowego, wszystkie wejścia i operacje danego, *i*-tego neuronu warstwy są przyporządkowane do *i*-tej jednostki przetwarzającej PE (rys. 1). Jednostka ta dysponuje kompletnym wektorem sygnałów wejściowych neuronu $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$ oraz jego współczynników wagowych $\boldsymbol{w_i} = [w_{i1}, w_{i2}, \ldots, w_{in}]$. Wylicza ona potencjał synaptyczny φ_i tego neuronu oraz, dla danej funkcji aktywacji, jego sygnał wyjściowy y_i.



Rys. 1. Schemat ideowy zrównoleglenia neuronowego [15]
Fig. 1. Schematic diagram of neural parallelization [15]

Z kolei w przypadku zrównoleglenia synaptycznego, jednostka przetwarzająca PE nie dysponuje kompletem wejść jednego neuronu, ale częścią wejść każdego z neuronów, odpowiadającą tym samym sygnałom wejściowym danej warstwy. Każda jednostka wylicza częściową sumę iloczynów wartości wag i wejść. Metodę zrównoleglenia synaptycznego zilustrowano na rys. 2.



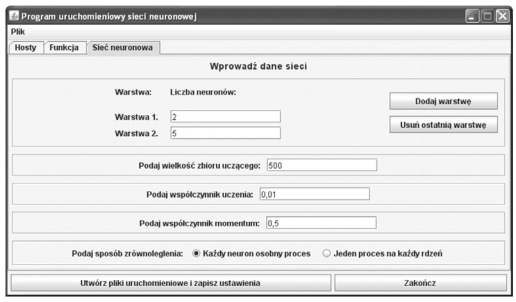
Rys. 2. Schemat ideowy zrównoleglenia synaptycznego [15]
Fig. 2. Schematic diagram of synaptic parallelization [15]

Charakter zrównoleglenia może być zróżnicowany w ramach jednej sieci neuronowej – dla części warstw może zostać użyte zrównoleglenie neuronowe, dla innych – synaptyczne. W przypadku sieci o małej liczbie wejść i wyjść, najbardziej optymalny wydaje się taki podział, w którym potencjał synaptyczny neuronów pierwszej warstwy ukrytej wyliczany jest z wykorzystaniem zrównoleglenia neuronowego, warstwy wyjściowej zaś przez zrównoleglenie synaps. Gdy w sieci znajduje się więcej warstw ukrytych, sposób wyliczania potencjału neuronów tych warstw nie powinien mieć znaczenia dla wydajności procesu uczenia oraz działania sieci [3, 13, 15].

4. Implementacja sieci neuronowej oraz wybrane wyniki badań

W oparciu o wymienione w poprzednim punkcie metody zrównoleglenia zbudowano aplikację, implementującą jednokierunkową, wielowarstwową sieć neuronową. Sieć ta uczona jest metodą gradientową z wsteczną propagacją błędów oraz z współczynnikiem *momentum*. Na aplikację składają się dwie główne części:

napisany w *Javie* interfejs uruchomieniowy, oraz właściwa implementacja sieci, zrealizowana z wykorzystaniem standardu MPI. Jedno z okien interfejsu uruchomieniowego, umożliwiające wprowadzanie parametrów sieci, przedstawiono na rys. 3.



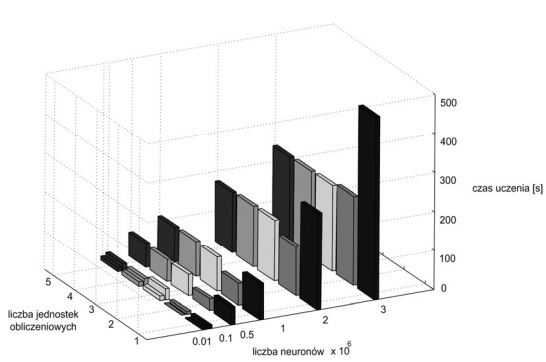
Rys. 3. Widok interfejsu uruchomieniowego – zakładka *Sieć neuronowa*
Fig. 3. Runtime user interface – *Neural network* tab

Do badań wykorzystano prosty klaster złożony z 3 komputerów klasy PC. Pierwszy z nich wyposażony jest w dwurdzeniowy procesor *Core2 Duo*, o częstotliwości taktowania 2,13 GHz i 2 GB pamięci RAM. Drugi to laptop *HP Compaq 6715B*, wyposażony w dwurdzeniowy procesor *AMD Turion* 1,9 GHz i 3 GB pamięci. Trzeci posiada jednordzeniowy procesor *AMD Sempron 2800+*, o częstotliwości 1,6 Ghz i 512 MB pamięci. Komputery połączone są ze sobą siecią *LAN Fast Ethernet* 100 Mbit/s. Do komunikacji między nimi wykorzystywany jest router z 4-portowym switchem *Linksys WAG160N* [3].

Przeprowadzone badania polegały na aproksymacji kilku różnych funkcji przy użyciu sieci neuronowych o różnych strukturach, uruchamianych na różnej liczbie jednostek obliczeniowych, przy użyciu różnej liczby wzorców uczących oraz dla różnych parametrów uczenia. Ze względu na szczupłość dostępnego miejsca przedstawiono tu jedynie wyniki uzyskane dla jednej z kategorii badań. Rozpatrywane zadanie polegało na aproksymacji funkcji $f(x)=\sin(x)$ dla wartości argumentu $x\in[-10,10]$. Uzyskane wyniki, reprezentujące czasy uczenia sieci, uzyskane dla zbioru uczącego złożonego 200 próbek oraz dla 50 cykli uczących przedstawiono w tab. 1. Z kolei wykres otrzymany na podstawie tych danych zaprezentowano na rys. 4.

Tab. 1. Czasy uczenia sieci neuronowej dla 50 cykli uczących (w sekundach)
Tab. 1. Neural network learning times for 50 learning epochs (in seconds)

liczba neuronów	liczba jednostek obliczeniowych				
	1	2	3	4	5
10^4	1,10498	0,89382	10,87420	10,89160	10,89690
10^5	8,95341	7,24807	27,93150	16,40500	18,68010
$5\cdot10^5$	44,34160	28,56530	54,54590	56,00050	60,64860
10^6	91,48280	55,70380	87,39070	90,04260	89,38170
$2\cdot10^6$	247,86700	124,32000	156,51600	154,75700	152,97300
$3\cdot10^6$	469,62200	228,17100	219,46500	219,17800	218,75700



Rys. 4. Graficzna ilustracja wyników z tab. 1
Fig. 4. Chart of data of tab. 1

Na podstawie otrzymanych wyników można zauważyć, że w przypadku małych sieci uruchomienie pojedynczego procesu, odpowiedzialnego za obliczenia warstwy ukrytej daje lepsze rezultaty, niż wykorzystanie obu dostępnych rdzeni procesora i włączenie dwóch procesów. Dopiero wówczas, gdy liczba neuronów w warstwie ukrytej przekracza 10^4, zwiększenie liczby jednostek obliczeniowych do dwóch pozwoliło uzyskać czas obliczeń mniejszy, niż dla pojedynczej jednostki obliczeniowej. Uruchomienie obliczeń na większej liczbie komputerów w omawianym przypadku przynosi stosunkowo małe korzyści. Czas uczenia dla sieci uruchomionej na dwóch różnych komputerach był mniejszy niż w przypadku sieci uruchomionej na pojedynczej maszynie dopiero dla $3\cdot10^6$ neuronów w warstwie ukrytej.

5. Podsumowanie

Korzyści wynikające ze zrównoleglenia sieci neuronowych na heterogenicznym klasterze złożonym z kilku komputerów klasy PC widoczne są dopiero w przypadku sieci złożonych z milionów neuronów. Jest to głównie wynikiem opóźnień powstałych przy przesyłaniu danych za pomocą sieci LAN. Wraz ze wzrostem liczby rdzeni w produkowanych współcześnie procesorach można będzie pokusić się o zrównoleglenie działania sieci neuronowej na pojedynczym komputerze. W komputerach domowych, często najwydajniejszym procesorem nie jest jednostka centralna, lecz jednostka służąca do obliczeń graficznych. Jej architektura, złożona z setek prostych jednostek przetwarzających, tzw. *procesorów strumieniowych*, wydaje się być odpowiednia dla implementacji sztucznych sieci neuronowych. Jednak niezależnie od metody realizacji, zastosowanie architektury równoległej w aplikacjach realizujących uczenie i działanie sieci neuronowych, z definicji działających w sposób równoległy, jest obiecującym sposobem na ich wydajną implementację.

6. Literatura

[1] Bartecki K.: Niektóre osobliwości aproksymacji neuronowej na przykładzie odwrotnego zadania kinematyki. Pomiary Automatyka Kontrola (6): 589-592 (2010).
[2] Bartecki K.: Sztuczne sieci neuronowe w zastosowaniach. Skrypt Politechniki Opolskiej nr 289, Opole, 2010.
[3] Czorny M.: Implementacja jednokierunkowej sieci neuronowej w architekturze równoległej z wykorzystaniem standardu MPI. Praca dyplomowa magisterska, Politechnika Opolska, Opole, 2008.
[4] Hornik K., Stinchcombe M., White H.: Mulitilayer Feedforward Networks are Universal Approximators. Neural Networks (2): 359-366 (1989).
[5] Korbicz J., Obuchowicz A., Uciński D.: Sztuczne sieci neuronowe – podstawy i zastosowania, AOW PLJ, Warszawa, 1994.
[6] Lotrič U., Dobnikar A.: Parallel implementations of feed-forward neural network using MPI and C# on .NET platform. Adaptive and Natural Computing Algorithms (6): 534-537 (2005).
[7] Masters T.: Sieci neuronowe w praktyce: programowanie w języku C++. WNT, Warszawa, 1996.
[8] Osowski S.: Sieci neuronowe w ujęciu algorytmicznym. WNT, Warszawa, 1996.
[9] Pacheco P. S.: A User's Guide to MPI. University of San Francisco, San Francisco, 1998.
[10] Pethick M., Liddle M., Werstein P., Huang Z.: Parallelization of a Backpropagation Neural Network on a Cluster Computer. Proceedings of the 15th IASTED International Conference on Parallel and Distributed Computing and Systems, Marina del Ray, pp. 574–582 (2003).
[11] Rojas R.: Neural networks: a systematic introduction. Springer-Verlag, Berlin, 1996.
[12] Snir M., Otto S., Huss-Lederman S., Walker D., Dongarra J.: MPI: The Complete Reference, The MIT Press, Massachusetts, 1996.
[13] Sundararajan N., Saratchandran P.: Parallel Architectures for Artificial Neural Networks Paradigms and Implementations. Wiley-IEEE Computer Society Press, 1998.
[14] Tadeusiewicz R.: Sieci neuronowe. AOW RM, Warszawa, 1993.
[15] Torresen T.: Parallelization of Backpropagation Training for Feed-Forward Neural Networks. Ph.D Thesis, University of Trondheim, 1996.

otrzymano / received: 07.12.2010
przyjęto do druku / accepted: 04.05.2011 *artykuł recenzowany*