Levenberg-Marquardt Algoritması ile YSA Eğitiminin Donanımsal Gerçeklenmesi

Hardware Implementation of Neural Network Training with Levenberg-Marquardt Algorithm

Mehmet Ali Çavuşlu[1], *Yaşar Becerikli*[2], *Cihan Karakuzu*[3]

[1]Fen Bilimleri Enstitüsü, Bilgisayar Müh. Anabilim Dalı Kocaeli Üniversitesi, Kocaeli
[2]Bilgisayar Mühendisliği Bölümü, Kocaeli Üniversitesi, Kocaeli
[3]Bilgisayar Mühendisliği Bölümü, Bilecik Üniversitesi, Bilecik

alicavuslu@gmail.com, ybecerikli@kocaeli.edu.tr, cihan.karakuzu@bilecik.edu.tr

Özetçe

Levenberg-Marquardt (LM) algoritması yapay sinir ağlarının eğitiminde sağlamış olduğu hız ve kararlılık nedeni ile tercih edilmektedir. Bu çalışmada yapay sinir ağı (YSA) eğitiminin LM algoritması ile kayan noktalı sayı formatında donanımsal olarak FPGA'da gerçeklenmesi sunulmuştur. Donanımsal gerçekleme ISE Webpack10.1 programı kullanılarak Xilinx Virtex 5 xc5vlx110-3ff1153 FPGA'sı üzerinde gerçeklenmiştir. Çalışmada özellikle YSA mimarisinin doğasında var olan paralelliğin FPGA üzerine aktarılmasının yanı sıra eğitim aşamasında LM algoritması da paralel veri işlemeye uygun olarak gerçeklenmiştir. Elde edilen sentez sonuçları, LM ile YSA eğitiminin FPGA üzerinde başarı ile gerçeklenebileceğini ortaya koymuştur.

Anahtar Kelimeler: *Levenberg-Marquardt, YSA, FPGA, kayan noktalı sayı*

Hardware Implementation of Neural Network Training with Levenberg-Marquardt Algorithm

Levenberg-Marquardt (LM) algorithm is preferred due to providing fast convergence and stability in training of artificial neural networks (ANN). In this study, hardware implementation of ANN training with LM algorithm is presented on FPGA using floating point number representation. Hardware implementation has been realized on Virtex-5 xc5vlx110-3ff1153 FPGA using ISE Webpack 10.1 software. In this work, both ANN and its training using LM have been particularly implemented on FPGA according to the inherent parallel data processing of ANN. Obtained synthesis results have showed that training of ANN using LM algorithm can be successfully implemented on FPGA.

Keywords: *Levenberg-Marquardt, YSA, FPGA, floating point number*

1. Giriş

Son zamanlarda Yapay Sinir Ağları (YSA) birçok uygulama alanında başarıyla kullanılmaktadır. Bu ağlarda, giriş çıkış eşleşmesini eğitim veri seti ile modelleyebilmek için genellikle geriye yayılım algoritması tercih edilmektedir. Birinci dereceden türev bilgisi gerektiren bu algoritma, düşük eğitim verimi ve kötü yakınsama hızı gibi sakıncalara sahiptir [1]. Dinamik öğrenme oranı geriye yayılım algoritmasının hızını, momentum da yakınsama hızını artırabilir [1,2]. Buna rağmen geriye yayılım algoritması bazı pratik uygulamalar için uygun olmayabilir. Newton, Levenberg-Marquardt (LM) gibi ikinci dereceden türev gerektiren algoritmalar öğrenme hızını belirgin bir şekilde artırmaktadır. Newton algoritması hızını ve dik iniş (steepest descent) metodunun kararlılığını birleştiren LM algoritması [3] günümüzde ağ eğitiminde etkili olarak kullanılmaktadır [4-5].

Yazılım uygulamalarının sıralı işlem yapması nedeni ile paralel veri işleme özelliğine sahip YSA'ların gerçek performansı yazılımsal uygulamalarda elde edilemez. Paralel veri işleme ve tekrar düzenlenebilirlik özelliğine sahip FPGA'lar son yıllarda birçok YSA uygulamasında ön plana çıkmıştır [6-11]. Bunlara örnek olarak [12-19] verilebilir.

[12]'de motor hız tahmini, [13]'de yüksek duyarlıklı PET pozisyon tahmini, [14]'de anahtarlamalı reluktans motor sürücülerinde pozisyon ve hız tahmini, [15]'de elastik kaplinli sistem sürücülerinde hız tahmini, [16]'da tıbbi tanı geliştirmek, [17]'de nükleer darbe parametre tahmini için parametreleri LM algoritması ile yazılım ortamında eğitilmiş YSA'nın FPGA üzerinde donanımsal gerçeklenmesi sunulmuştur.

Literatürde YSA'nin FPGA üzerinde geriye yayılım algoritması ile eğitimine ait çalışmalar bulunmaktadır [11, 18-19]. LM algoritması ile YSA eğitiminin FPGA üzerinde eğitimine ilişkin çalışmaya rastlanmamıştır.

Bu çalışmada matematiksel hücre aktivasyon yaklaşımlı YSA ve eğitimi Levenberg-Marquardt algoritması ile kayan noktalı sayı formatı kullanılarak FPGA üzerinde paralel mimaride gerçeklenmiştir.

2. Levenberg-Marquard Algoritması

Dik iniş (steepest descent) ve Newton algoritmalarından türetilen LM algoritması güncellemesi (1)'de verilmiştir.

$$\Delta w = (J^T J + \mu I)^{-1} J^T e \tag{1}$$

w ağırlık vektörü, I birim matris, μ kombinasyon katsayısıdır. J (PxM)xN boyutunda Jacobian matrisini, e (PxM)x1 boyutunda hata vektörünü göstermektedir. P eğitim örnek sayısını, M çıkış sayısını ve N ağırlık sayısını göstermektedir. μ ayarlanabilir bir parametredir. Eğer bu parametre çok büyükse yöntem dik iniş metodu gibi çok küçükse Newton metodu gibi davranmaktadır. Bu parametre için uyarlamalı bir yapı (2)'de verilmiştir.

$$\mu(n) = \begin{cases} \mu(n-1)k & E(n) > E(n-1) \\ \mu(n-1)/k & E(n) \le E(n-1) \end{cases} \tag{2}$$

Denklem (2)'de *k* sabit bir sayıdır. E uygunluk değerini göstermektedir.

3. FPGA Tabanlı YSA Eğitimin LM Algoritması ile Gerçeklenmesi

YSA eğitiminin LM algoritması ile FPGA'da donanımsal gerçeklenmesi YSA'nın gerçeklenmesi ve Jacobian matrisinin oluşturulması, ağırlıkların güncellenmesi aşamalarından meydana gelmektedir.

3.1 YSA'nın gerçeklenmesi ve Jacobian Matrisinin Oluşturulması

Donanımsal YSA gerçeklemesinde kullanılacak olan sayı formatı ve aktivasyon fonksiyonun ağ eğitiminden önce belirlenmesi gerekmektedir. Bu çalışmada yüksek duyarlılık ve hassasiyete sahip kayan noktalı sayılarla çalışılmıştır. Bu çalışmada, ağda kullanılan hücre aktivasyon fonksiyonunun türevlenebilir olması gerekliliği nedeniyle ve literatürde sunulan bakma tablosu ve yüksek çözünürlüklü parçalı doğrusal yaklaşımlarının maliyetli olması nedeniyle logaritmik sigmoidal fonksiyonun türevlenebilir matematiksel yaklaşımı kullanılmıştır.

3.1.1 Kayan noktalı sayı aritmetiği

Kayan noktalı sayı standardı, bir sayının 10'un kuvveti ile gösterilmesine benzer bir gösterim sistemine sahiptir. En çok kullanılan kayan noktalı sayı gösterim standardı IEEE 754 standardıdır. Bu çalışmada (3) ifade edilen tek duyarlı (32 bit) IEEE 754 standardı kullanılmıştır.

$$Sayı = (-1)^s 2^{e-bias}(1+f) \tag{3}$$

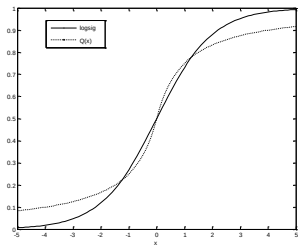
Denklem (3)'de, ***s*** işaret bitini temsil eder. İşaret biti "0" ise sayının pozitif, "1" ise negatif olduğunu belirtir. ***e*** üs değerini ve ***f*** çarpan değerini gösterir. Çarpan değeri daima sıfır ile bir arasında olmalıdır.

3.1.2. Aktivasyon fonksiyonu

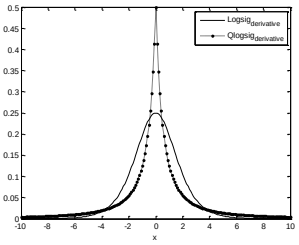
Bu çalışmada, üstel fonksiyon FPGA'da doğrudan gerçeklenemediğinden, YSA aktivasyon fonksiyonu olarak logaritmik sigmoidal fonksiyon yerine (4)'de verilen fonksiyonel yaklaşımı kullanılmıştır. Şekil 1(a)'da da görüldüğü gibi kullandığımız fonksiyon logaritmik sigmoidal fonksiyonuna oldukça benzeyen bir fonksiyondur. Bu fonksiyonun türevi ise (5)'de verilmiş olup değişimi Şekil 1b'de gösterilmiştir.

$$Q(x) = \frac{1}{2}\left(1 + \frac{x}{1+|x|}\right) \tag{4}$$

$$Q'(x) = \begin{cases} 2Q(x)^2, & x < 0 \\ 2(1-Q(x))^2, & x \ge 0 \end{cases} \tag{5}$$



(a)



(b)

Şekil-1: (a) Aktivasyon fonksiyonu (b) türevi

3.1.3 YSA'nın gerçeklenmesi

Uygulamada Şekil 2'de blok yapısı gösterilen 2 girişli, 2 gizli katman hücresine sahip, 1 çıkışlı YSA mimarisi gerçeklenmiştir. Eğitim ölçüt değerinin hesaplanması için (6)'da verilen toplam karesel hata ölçütü kullanılmıştır. Eşitlik 6'daki hata (e_j) (7) ile tanımlanır.

$$E_n = \frac{1}{2}\sum_j je_j^2 \tag{6}$$

$$e_j = yd_j - y_j \tag{7}$$

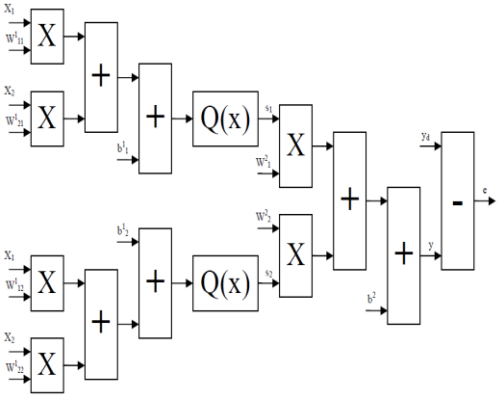
j örnek indeksini, yd_j istenen çıkış değerini göstermektedir. Hataya dayalı olarak, Şekil 2'de verilen YSA için Jacobian matrisinin oluşturulması Şekil 3'te blok şemada gösterildiği gibi yapılır. Paralel işlem yapabilmek amacı ile Jacobian matrisi parametre sayısı kadar (P) giriş örnek uzunluğunda (N) blok ramlara yazılmaktadır. Yani tüm parametrelere ait değerler, kendi matrisinde tutulmaktadır. RAM'ların derinlikleri 32 bit kayan noktalı sayı formatında işlem yapılacağından dolayı 32 bit'tir.

3.2 Parametrelerin Güncellemesi

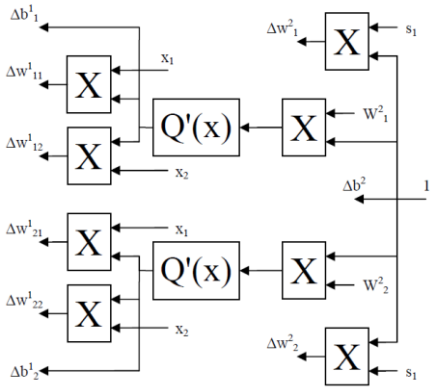
Parametrelerin güncellemesi işleminde (1)'deki formül adım adım uygulanmaktadır. Eşitlikten de görüleceği üzere, parametre güncelleme değerlerini elde etmek için matris çarpımı ve matris tersi işlemleri yapılması gerekmektedir. Matris çarpım sonuçlarında elde edilecek sonuçlar da paralel mimariye uygun olarak oluşturulacak yeni RAM bloklarında saklanmaktadır. Örneğin NxP boyutunda Jacobian matrisinin tersi ile PxN boyutunda Jacobian matrisinin çarpımının sonucu NxN boyutunda matriste saklanmalıdır. Bu nedenle bu işlem sonucu için N adet N uzunluğunda blok RAM oluşturulmuştur.

3.2.1 Paralel matris çarpım mimarisi

Şekil 4'te matris çarpım işlemine ait kaba kod verilmiştir. Parametre değerlerine ait verilerin farklı RAM'larda saklanması ve işlem yapılacak satırdaki tüm değerlerin aynı anda okunabilmesi ile işlemler aynı anda gerçeklenerek zamandan kazanım sağlanmaktadır (Şekil.5). Şekil 5(a)'dan da açıkça görüleceği üzere her parametrenin farklı RAM'larda saklanması ile matris çarpma işlemi daha hızlı olarak gerçeklenebilmektedir.



Şekil-2 : 2-2-1 yapısında YSA



Şekil-3: Jacobian matrisinin hesaplanması

```
for i = 1 : N
for j = 1 : N
for k = 1 : P
Jmul(i, j) = Jmul(i, j) + J(k, i) * J(k,j);
```

Şekil -4: Matris çarpım işlemi kaba kodu

```
for i = 1 : N
for k = 1 : P
        paralel matris çarpım bloğu
```

(a)

```
Jmul(i, 1) = Jmul(i, 1) + J(k, i) * J(k,1);
Jmul(i, 2) = Jmul(i, 2) + J(k, i) * J(k,2);
             |          |
             |          |
Jmul(i, N) = Jmul(i, N) + J(k, i) * J(k,N);
```

(b)

Şekil -5: (a) Paralel matris çarpım işlemi (b) Paralel matris çarpım bloğu

3.2.2. Paralel matris tersi alma işlemi

Bu çalışmada matrisin tersini almak için Gauss-Jordan yok etme yöntemi kullanılarak matris boyutu sınırlaması olmadan donanımsal gerçeklenmesi üzerinde çalışılmıştır (Şekil.6).

```
I = eye(n,n);
for i = 1 : n
d = a(i, i);
for j = 1 : n
a(i, j) = a(i, j) / d;
I(i, j) = I(i, j) / d;
for x = 1 : n
if x /= i
k = a(x, i);
for j = 1 : n
a(x, j) = a(x, j) - k * a(i, j);
I(x, j) = I(x, j) - k * I(i, j);
```

Şekil-6: Matris tersi alma işlemi

Her parametre için oluşturulan blok RAM'lar ile verilerin paralel okunabilmesiyle matrisin tersini alma işleminde de paralel bloklar oluşturulmuştur (Şekil 7).

```
Jinv = eye(N, N)
for i = 1 : N
d = Jmul(i, i);
paralel bölme bloğu

for x = 1 : N
if x /= i
k = Jmul (x, i);
Paralel çarpım bloğu
```

(a)

```
Jmul(i, 1) = Jmul(i, 1) / d;
Jinv(i, 1) = Jinv(i, 1) / d;
             |  |
             |  |
Jmul (i, N) = Jmul (i, N) / d;
Jinv(i, N) = Jinv(i, N) / d;
```

(b)

```
Jmul (x, 1) = Jmul (x, 1) - k * Jmul (i, 1);
Jinv(x, 1) = Jinv(x, 1) - k * Jinv(i, 1);
             |          |
        |          |
Jmul (x, N) = Jmul (x, N) - k * Jmul (i, N);
Jinv(x, N) = Jinv(x, N) - k * Jinv(i, N);
```

(c)

Şekil-7: (a) Paralel matris tersi alma işlemi (b) Paralel bölme bloğu (c) Paralel çarpım bloğu

4. Deneysel Sonuçlar

Önceki bölümde açıklanan LM ile YSA eğitiminin FPGA üzerinde gerçeklenmesi iki örnek üzerinde deneysel olarak test edilmiştir. Deneysel çalışmalar, ISE Webpack10.1 programı kullanılarak Xilinx Virtex 5 xc5vlx110-3ff1153 FPGA'sı üzerinde gerçeklenmiştir.

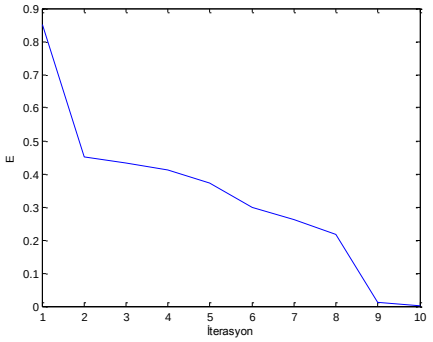
Örnek-1 :Bu örnekte EXOR problemini çözecek 2 giriş, 2 hücreli 1 gizli katman ve 1 çıkış hücresinden oluşan YSA eğitimi üzerine odaklanılmıştır. EXOR probleminde örnek sayısı P=4, ve parametre sayısı N= 9' a göre RAM blokları ve mimari oluşturulmuştur. Şekil.8 ve Tablo 1'de EXOR problemi için aynı başlangıç koşullarında 10 iterasyonda elde edilen uygunluk değerine ait FPGA ve yazılım çıktı sonuçlarının karşılaştırılması verilmiştir.

Örnek 2: Eşitlik 8'de verilen problemim çözümü için 1 giriş, 2 hücreli 1 gizli katman ve 1 çıkış hücresinden oluşan YSA eğitimi üzerinde çalışılmıştır. Eşitlik 8'de q=3.75 ve y(0)=0.6 seçilmiştir. Örnek için ağ eğitiminde 8 örnek kullanılmıştır (P=8) ve parametre sayısı N =7'ye göre RAM blokları ve mimari oluşturulmuştur.

$$y(i) = qy(i-1)\left[1 - y(i-1)\right] \qquad (8)$$

Şekil.9'da 8 giriş örneğiyle 30 iterasyonda eğitilmiş ağın eğitimde kullanılan 8 örnek ve eğitimde kullanılmayan 8 örnekle birlikte toplam 16 örnek üzerinde test edilmesi gösterilmiştir. Şekilden de görüleceği üzere başarılı bir eğitim gerçeklenmiştir.

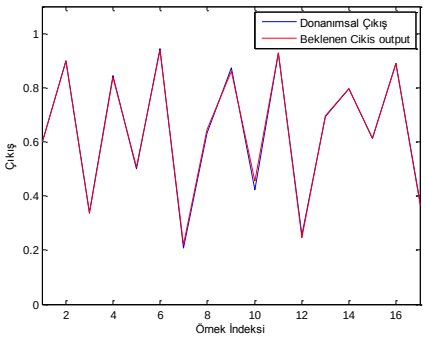
Tablo 2'de örnek 1 ve örnek 2'nin donanımsal gerçeklenmesinde kullanılan toplama, çarpma ve bölme modüllerinin toplam sayısı verilmiştir. Tablo 3'te de aynı örneklere ait sentez sonuçları ve parametrelerin saklanması için gerekli hafıza kullanımı gösterilmiştir.



Şekil-8: EXOR problemi için 10 iterasyonda uygunluk değeri çıktısı

Tablo-1: Örnek 1 için yazılımsal ve donanımsal uygunluk değerleri

	Yazılım	FPGA
1	0.8525466333690	0.85254651
2	0.4501576202882	0.45015827
3	0.4317160062259	0.43168321
4	0.4111825700951	0.41124606
5	0.3733348978269	0.37337276
6	0.2988104777587	0.29879546
7	0.2606668960795	0.2610005
8	0.2177783000194	0.2178748
9	0.01258936438906	0.01254566
10	0.000375131752227	0.00036696



Şekil-9: Örnek 2 eğitim sonucunda eğitim ve test verilerinden elde edilen beklenen ve ağ çıktıları

Tablo-2 : Örnek 1 ve Örnek 2 için kullanılan aritmetik işlem modül sayısı

	Örnek 1			Örnek 2		
	Toplama	Çarpma	Bölme	Toplama	Çarpma	Bölme
MLP	10	6	2	8	4	2
E	2	1	-	2	1	-
Upd.	43	48	18	39	42	14
Top.	55	55	20	49	47	16

Tablo-3 : Örnek 1 ve Örnek 2 için donanımsal gerçeklenmeye ait sentez sonuçları

Lojik eleman	Örnek 1			Örnek 2		
	Kullanılan	Mevcut	Kullanım oranı	Kullanılan	Mevcut	Kullanım oranı
Dilim saklayıcı sayısı	10540	69120	15%	8828	69120	12%
Dilim LUT sayısı	47510	69120	68%	40609	69120	58%
Blok RAM sayısı	18	148	12%	16	148	10%

5. Yorumlar

Çalışmada özellikle matematiksel işlemlerin FPGA üzerinde paralel olarak gerçeklenmesi amaçlanmıştır. Bu nedenle FPGA üzerinde YSA mimarisi Şekil 2'deki gibi, Jacobian matrisinin Şekil 3'deki gibi ve (1)'de verilen YSA parametrelerinin güncelleme işlemi ise Şekil 4-7'de gösterilen kaba kodlardaki gibi paralel olarak gerçeklenmiştir.

Önerilen mimari ile gerçeklenen Örnek 1 için Tablo 1 ve Şekil 8, Örnek 2 için ise Şekil 9'dan da açıkça görüleceği üzere LM algoritmasıyla YSA eğitiminin kayan noktalı sayı formatında FPGA üzerinde eğitimi başarı ile gerçeklenmiştir. Tablo 3'te sayıları verilen kullanılan aritmetik modüllerin fazla olmasına rağmen uygulamaların FPGA üzerinde başarıyla gerçeklendiği gösterilmektedir.

Çalışmada kullanılan örneklerde, YSA mimarisi için gizli katmandaki ve çıkıştaki hücre sayıları aynı olmasına rağmen giriş sayısındaki farklılıktan dolayı meydana gelen aritmetik modül kullanım ihtiyacı ve donanım tüketiminde artış sırası ile Tablo 2 ve Tablo 3'de görülmektedir.

Kaynakça

[1] **Wilamowski, B.M., Yu, H.,** 2010, Improved computation for Levenberg–Marquardt training, IEEE Transactions on Neural Networks , Cilt: 21, No: 6, 930-937.

[2] **Ferrari, S., Jensenius, M.,** 2008, A constrained optimization approach to preserving priorknowledge during incremental training, IEEE Trans. Neural Netw., Cilt: 19, No: 6, 996–1009.

[3] **Wilamowski, B.M., Chen, Y.,** 1999, Efficient algorithm for training neural Networks with one hidden layer, in Proc. of the International Joint Conference on Neural Networks, Cilt: 3, 1725-1728.

[4] **Dohnal ,J.,** 2004, Using of Levenberg-Marquardt method in identification by neural networks, In Student EEICT 2004. Student EEICT 2004. Brno: Ing. Zdeněk Novotný CSc., 2004, pp. 361 - 365, ISBN 80-214-2636-5 .

[5] **Khosravi, Z,M.H., Barghinia, S., Ansarimehr, P.,** 2006, New momentum adjustment technique for Levenberg-Marquardt neural network used in short term load forecasting, in Proc. of 21st International Power System Conference (PSC 2006), Tehran, Iran.

[6] **Chalhoub, N., Muller, F.,Auguin, M.,** 2006, FPGA-based generic neural network architecture, Industrial Embedded Systems, International Symposium on; Antibes Juan-Les-Pins,France,1-4, 18-20.

[7] **Çavuşlu, M. A., Karakuzu, C., Şahin,S.,** 2006, Neural network hardware implementation using FPGA, in proc. of ISEECE 2006 3rd International Symposium on Electrical, Electronic and Computer Engineering Symposium, Nicosia, TRNC, 287-290.

[8] **Elliot, D.L.,** 1993, A beter activation function for artificial neural networks, Technical Research Report T.R. 93-8, Institute for Systems Research, University of Maryland.

[9] **Lázaro, J., Arias, J., Astarloa, A., Bidarte,U., Zuloaga, A.,** 2007, Hardware architecture for a general regression neural network coprocessor, Neurocomputing, Volume 71, Issues 1-3, Pages 78-87.

[10] **Boubaker, M. Akil, M., Khalifa, K.B., Grandpierre, T., Bedoui, M.,** 2009, Implementation of an LVQ neural network with a variable size: algorithmic specification, architectural exploration and optimized implementation on FPGA devices, Neural

Computing & Applications, Cilt 19, Sayı 2, 283-297.

[11] **Çavuşlu, M.A., Karakuzu, C., Şahin, S., Yakut, M.,** 2011, Neural network training based on FPGA with floating point number format and it's performance, Neural Computing & Application, Cilt 20, Sayı 2,195-202.

[12] **Phan Quoc Dzung, Le Minh Phuong, Le Dinh Khoa, Truong Phuoc,** 2009, A new aproach for motor speed estimator using ANN based on FPGA, in Proc. of IFOST 2009 Oct.21 – Oct.23,131-136.

[13] **Mateo,F., Aliaga, R.J., Ferrando, N., Martı´nez, J.D., Herrero, V., Lerche, Ch.W. Colom, R.J.,Monzo´,J. M., Sebastia´, A., Gadea, R.,** 2009, High-precision position estimationin PET using artificial neural networks, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Volume 604, Issues 1–2, 366–369.

[14] **Talla J., Stehlik**, **J.,** 2008, FPGA based neural network position and speed estimator for switched reluctance motor drive, in Proceedings of the 7th conference on Circuits, systems, electronics, control and signal processing (CSECS'08), Stamatios Kartalopoulos, Andris Buikis, Nikos Mastorakis, and Luigi Vladareanu (Eds.). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 139-144.

[15] **Oniga, S., Tisan, A., Mic, D., Lung, C., Orha, I., Buchman, A., Vida-Ratiu, A.,** 2009, FPGA implementation of feed-forward neural networks for smart devices development, in Proc. of International Symposium on Signals, Circuits and Systems, ISSCS 2009, Iasi, Romania, 1-4.

[16] **Subadra, M., Marimuthu, N.S.,** 2008, Neuro-chip design for developing wearable medical diagnostic E-sniffer, in Proc. of TENCON 2008 - 2008 IEEE Region 10 Conference, Hyderabad, India, 1-6.

[17] **Estryk, D.S., Rios, G.E., Verrastro, C.,** 2007, FPGA neural networks implementation for nuclear pulses parameters estimation, in Proc. of 3rd Southern Conference on Programmable Logic, SPL '07, Mar del Plata, Argentine,7-12.

[18] **Savich, A. W., Moussa, M., Areibi, S.,** 2007, The impact of arithmetic representation on implementing MLP-BP on FPGAs: A study, IEEE Transactions on Neural Networks, Cilt: 18, Sayı: 1, 240 – 252.

[19] **Farmahini-Farahani, A., Fakhraie, S. M., Safari, S.,** 2008, Scalable architecture for on-chip neural network training using swarm intelligence, in Proc. of the Design, Automationand Test in Europe Conf. (DATE'08), Munich, Germany, 1340-1345.