M.A. ÇAVUŞLU, 2013

T.C
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANA BİLİM DALI

YAPAY SİNİR AĞLARI EĞİTİMİNİN GRADYEN TABANLI VE GLOBAL ARAMA ALGORİTMALARI İLE FPGA ÜZERİNDE DONANIMSAL GERÇEKLENMESİ

MEHMET ALİ ÇAVUŞLU

KASIM 2013

YÜKSEK LİSANS TEZİ

NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

T.C.
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANA BİLİM DALI

YAPAY SİNİR AĞLARI EĞİTİMİNİN GRADYEN TABANLI VE GLOBAL ARAMA ALGORİTMALARI İLE FPGA ÜZERİNDE DONANIMSAL GERÇEKLENMESİ

MEHMET ALİ ÇAVUŞLU

Yüksek Lisans Tezi

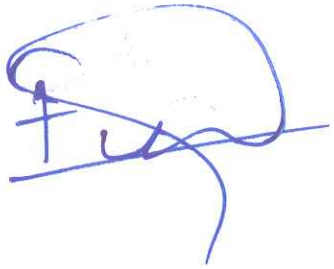
Danışman

Yrd. Doç. Dr. Fuat KARAKAYA

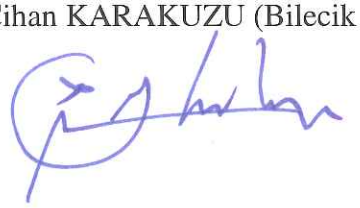
KASIM 2013

Mehmet Ali Çavuşlu tarafından **Yrd. Doç. Dr. Fuat Karakaya** danışmanlığında hazırlanan "**Yapay Sinir Ağları Eğitiminin Gradyen Tabanlı ve Global Arama Algoritmaları ile FPGA Üzerinde Donanımsal Gerçeklenmesi**" adlı bu çalışma jürimiz tarafından Niğde Üniversitesi Fen Bilimleri Enstitüsü Ana Bilim Dalı'nda Yüksek Lisans tezi olarak kabul edilmiştir.

Başkan : Yrd. Doç. Dr. Fuat KARAKAYA (Niğde Üniversitesi)



Üye : Doç. Dr. Cihan KARAKUZU (Bilecik Şey Edebali Üniversitesi)



Üye : Yrd. Doç. Dr. Suhap ŞAHİN (Kocaeli Üniversitesi)

ONAY:

Bu tez, Fen Bilimleri Enstitüsü Yönetim Kurulunca belirlenmiş olan yukarıdaki jüri üyeleri tarafından …./…./20.... tarihinde uygun görülmüş ve Enstitü Yönetim Kurulu'nun …./…./20.... tarih ve ..... sayılı kararıyla kabul edilmiştir.

...../...../20...

Doç. Dr. Osman SİVRİKAYA
MÜDÜR

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin bilimsel ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

Mehmet Ali ÇAVUŞLU

ÖZET

YAPAY SİNİR AĞLARI EĞİTİMİNİN GRADYEN TABANLI VE GLOBAL ARAMA ALGORİTMALARI İLE FPGA ÜZERİNDE DONANIMSAL GERÇEKLENMESİ

ÇAVUŞLU, Mehmet Ali
Niğde Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Ana Bilim Dalı
Danışman : Yrd. Doç. Dr. Fuat KARAKAYA
Kasım 2013, 80 sayfa

Yapay sinir ağları (YSA), sistem giriş ve çıkışları arasındaki karmaşık ilişkiyi etkili bir şekilde modelleyebilme yeteneği ile bilimsel çalışmalarda yaygın olarak kullanılmaktadır. Güncel çalışmalarda YSA eğitiminde gradyen tabanlı algoritmaların yanında global arama özelliğine sahip algoritmalar da kullanılmaktadır. Bu çalışmada, gradyen tabanlı algoritmalardan geriye yayılım (GY) ve Levenberg & Marquardt (LM) algoritmaları ile sezgisel arama özelliğine sahip algoritmalardan parçacık sürü optimizasyon (PSO) ve yapay arı koloni (YAK) algoritmaları kullanılarak YSA eğitimi FPGA üzerinde donanımsal olarak gerçeklenmiştir. Gerçeklemelerde sağlamış olduğu dinamiklik ve hassasiyetten ötürü IEEE 754 kayan noktalı sayı formatı kullanılmıştır. FPGA üzerinde YSA gerçeklemesinde en kritik aşama olan aktivasyon fonksiyonunun gerçeklenmesinde matematiksel yaklaşımlar tercih edilmiştir. Donanımsal gerçeklemeler dinamik sistem tanıma ve araç plaka bölgesi belirleme problemleri kullanılarak test edilmiştir. Eğitilen YSA'lar eğitim fazında ağa gösterilmeyen giriş-çıkış örnekleri ile test edilmiş ve her örnek için yukarıda bahsi geçen algoritmaların YSA eğitimindeki başarım oranları kıyaslamalı olarak verilmiştir.

Anahtar Sözcükler: FPGA, Yapay Sinir Ağları, Yapay Sinir Hücresi, Geriye Yayılım Algoritması,Levenberg & Marquardt Algoritması, Parçacık Sürü Optimizasyon Algoritması, Yapay Arı Koloni Algoritması, Kayan Noktalı Sayı

SUMMARY

HARDWARE IMPLEMENTATON OF ARTIFICIAL NEURAL NETWORK TRAINING USING GRADIENT BASED AND GLOBAL SEARCH ALGORİTHMS ON FPGA

ÇAVUŞLU, Mehmet Ali
Niğde University
Graduate School of Natural and Applied Science
Department of Electrical-Electronics Engineering

Supervisor : Assistant Professor Dr. Fuat KARAKAYA
November 2013, 80 pages

Artificial neural networks (ANNs) are commonly used in scientific studies due to their ability to effectively model complex relationship between input and output of a system. In recent studies global search algorithms are also utilized in addition to gradient based algorithms in ANN training. In this study, hardware implementation of ANN training on FPGA is realized using gradient based algorithms such as Back Propagation (BP) and Levenberg&Marquardt, and heuristic algorithms such as Particle Swarm Optimization (PSO) and Artificial Bee Colony (ABC). For implementation, floating point numbers are chosen as number format due to its dynamism and accuracy. Mathematical approachesare preferred for hardware implementation of activation functions, which is the most critical stage of ANN implementation on FPGA. The hardware implementations on FPGA are tested using dynamic system identification and license plate recognition problems. The trained ANNs are tested using input-output data sets which are not used in training and results for each example are given in a comparative manner.

Keywords: FPGA, Artificial Neural Networks, Artificial Neural Cell, Back Propagation Algorithm, Levenberg & Marquardt Algorithm, Particle Swarm Optimization Algorithm, Artificial Bee Colony Algorithm, Floating Point Number

ÖN SÖZ

Bu yüksek lisans çalışmasında yapay sinir ağının ve eğitiminin geriye yayılım algoritması, Levenberg & Marquardt algoritması, parçacık sürü optimizasyon algoritması ve yapay arı koloni algoritması ile gerçekleştirilmiştir. Çalışmada gerçeklemeler dinamik sistem tanıma ve araç plaka bölgesi belirleme problemleri kullanılarak yapılmıştır. Eğitilen YSA'lar eğitim fazında ağa gösterilmeyen giriş-çıkış örnekleri ile test edilmiş ve her örnek için yukarıda bahsi geçen algoritmaların YSA eğitimindeki başarım oranları kıyaslamalı olarak verilmiştir.

Bu tez çalışması sürecinde bana desteğini esirgemeyen, değerli danışmanım Sayın Yrd. Doç. Dr. Fuat Karakaya'ya; bugünlere gelmemde emeği olan sayın hocam Doç. Dr. Cihan Karakuzu'ya; tez çalışmalarıma verdiği destekten dolayı Dr. Serdar Özen ve Mehmet Muzaffer Kösten'e; eğitim hayatım boyunca bana göstermiş oldukları sabır ve desteklerinden ötürü aileme teşekkür ederim.

İÇİNDEKİLER DİZİNİ

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

KISALATMALAR

YSA **:** Yapay Sinir Ağları

YSH **:** Yapay Sinir Hücresi

GY **:** Geriye Yayılım

L&M **:** Levenberg & Marquardt

PSO **:**Parçacık Sürü Optimizasyon

YAK **:** Yapay Arı Kolonisi

ÇKA : Çok Katmanlı Algılayıcı

GA **:** Genetik Algoritma

BÖLÜM I

GİRİŞ

Giriş ve çıkış arasındaki doğrusal olmayan karmaşık ilişkiyi etkili bir şekilde modelleyebilen yapay sinir ağları (YSA) (Merchant, Peterson ve Kong 2006), birçok alanda çeşitli problemlerin çözümünde başarıyla kullanılmıştır (Karakuzu ve Öztürk 2000; Çavuşlu ve diğerleri 2008; Li ve Areibi 2004; Narendra ve Narendra 1990; Economou ve diğerleri 1994).En çok kullanılan türü çok katmanlı algılayıcılar (ÇKA) olan YSA'lar(Ferreira ve diğerleri 2006; Mandal, Sural ve Patra 2008), giriş-çıkış eşlemesinin, uygun bir modeline ait eğitim seti ile eğitilmelidir (Merchant, Peterson ve Kong 2006). YSA'ların eğitiminde genellikle geriye yayılım (GY) algoritması kullanılır(Ferrari ve Jensenius 2006).

GY algoritması, düşük eğitim verimi (Wilamowski ve Chen 1999) ve kötü yakınsama hızı gibi dezavantajlara sahiptir (Ferrari ve Jensenius 2008). Newton, Levenberg&Marquardt(LM) gibi ikinci dereceden türev bilgisi gerektiren algoritmalar öğrenme hızını belirgin bir şekilde artırmaktadır (Wilamowski ve Chen 1999). Newton algoritmasının hızı ve eğim düşüm (steepest descent)metodunun kararlılığını birleştiren LM algoritması (Ferrari ve Jensenius 2008) günümüzde ağ eğitiminde etkili olarak kullanılmaktadır (Dohnal 2004; Khosravi, Barghinia ve Ansarimehr 2006).

Son yıllardaağ eğitiminde küresel arama özelliğine sahip genetik algoritma (GA), yapay arı kolonisi (YAK), parçacık sürü optimizasyonu (PSO) gibi evrimsel algoritmalar, türev bilgisi gerektiren eğitim algoritmalarına alternatif olarak kullanılmaktadır(Karaboga 2007; Karaboga, Akay ve Ozturk 2007; Kumbhar ve Krishnan 2011; Li ve Chen 2006; Vilovic, Burum ve Milic 2009; Haridas ve Devi 2010).

Paralel işlem yapabilme ve veri akışı özelliklerine sahip FPGA'larla, ağır işlem yükü gerektiren gerçek zamanlı uygulamalar gerçeklenebilmektedir(Martinez ve diğerleri 2008). Son yıllarda birçok YSA uygulamasında, tercih edilen gömülü sistem platformu olarak FPGA ön plana çıkmıştır (Krips, Lammert ve Kummert 2002; Ossoinig ve diğerleri 1996; Sahin, Becerikli ve Yazici 2006; Zhu ve diğerleri 1999; Mousa, Areibi ve Nichols 2006).

FPGA üzerinde, eğitilmiş (parametreleri bilinen) bir yapay sinir ağı gerçekleme (Nedjah ve diğerleri 2009; Ferreira ve diğerleri 2006; Won 2007; Ferrer ve diğerleri 2004; Çavuşlu, Karakuzu ve Şahin 2006) ve eğitimi ile birlikte bir yapay sinir ağı gerçekleme çalışmaları (Savich, Moussa ve Areibi 2007; Çavuşlu ve diğerleri 2011; Farmahini-Farahani, Fakhraie ve Safari 2008; Çavuşlu, Karakuzu ve Karakaya 2012) literatürde sunulmuştur.

Bu çalışmalardan, Ferrer ve diğerleri, Savich ve diğerleri, Farmahini-Farahani ve diğerleri farklı bit uzunluklarında sabit noktalı sayı formatı kullanmışlardır(Ferrer ve diğerleri 2004; Savich, Moussa ve Areibi 2007; Farmahini-Farahani, Fakhraie ve Safari 2008). Nedjah ve diğerleri, Çavuşlu ve diğerleri, Moussa ve diğerleri kayan noktalı sayı formatı kullanmışlardır (Nedjah ve diğerleri 2009; Ferreira ve diğerleri 2006; Çavuşlu, Karakuzu ve Şahin 2006; Savich, Moussa ve Areibi 2007; Çavuşlu ve diğerleri 2011; Çavuşlu, Karakuzu ve Karakaya 2012).Tam sayı formatı kullanan çalışma Won'a aittir (Won 2007).

Aktivasyon fonksiyonu olarak daNedjah ve diğerleri, Won, Çavuşlu ve diğerleri, Savich ve diğerlerilogaritmik sigmoidal yaklaşımlarını kullanmışlardır(Nedjah ve diğerleri 2009; Won 2007; Çavuşlu, Karakuzu ve Şahin 2006; Savich, Moussa ve Areibi 2007; Çavuşlu ve diğerleri 2011; Çavuşlu, Karakuzu ve Karakaya 2012). Ferreira ve diğerleri; Ferrer ve diğerleri; Farmahini-Farahani ve diğerleri, Çavuşlu ve diğerleri aktivasyon fonksiyonu olarak tanjant hiperbolik aktivasyon fonksiyon yaklaşımlarını kullanmışlardır (Ferreira ve diğerleri 2006; Ferrer ve diğerleri 2004; Farmahini-Farahani, Fakhraie ve Safari 2008; Çavuşlu, Karakuzu ve Karakaya 2012).

Bu aktivasyon fonksiyonlarının donanımsal gerçeklenmesinde Won, Farmahini-Farahani ve diğerleri bakma tablosu yaklaşımı kullanmışlardır (Won 2007; Farmahini-Farahani, Fakhraie ve Safari 2008).Ferreirave diğerleri, Ferrer ve diğerleri, Savich ve diğerleri parçalı doğrusal yaklaşım kullanmışlardır (Ferreira ve diğerleri 2006; Ferrer ve diğerleri 2004; Savich, Moussa ve Areibi 2007). Nedjah ve diğerleri parabolik yaklaşım kullanılmıştır (Nedjah ve diğerleri 2009).Çavuşlu ve diğerleri fonksiyonelyaklaşım kullanmıştır (Çavuşlu, Karakuzu ve Şahin 2006; Çavuşlu ve diğerleri 2011; Çavuşlu, Karakuzu ve Karakaya 2012).

Ağ eğitimi için Savich ve diğerleri, Çavuşlu ve diğerleri geriye yayılım algoritmasını kullanmıştır (Savich, Moussa ve Areibi 2007; Çavuşlu ve diğerleri 2011). Farmahini-Farahani ve diğerleri, Çavuşlu ve diğerleri ağ eğitiminde PSO algoritmasını kullanmıştır (Farmahini-Farahani, Fakhraie ve Safari 2008; Çavuşlu, Karakuzu ve Karakaya 2012).

L&M algoritması ve YAK algoritması kullanılarak ağ eğitiminin FPGA üzerinde donanımsal gerçeklenmesine ait çalışmalara literatürde rastlanmamıştır.

Bu tezde, FPGA üzerinde YSA'nın ve GY, LM, PSO ve YAK tabanlı eğitiminin 32 bit kayan noktalı sayı formatında gerçeklenmesi sistem tanıma problemleri ve plaka tanıma sistemi üzerinde örneklerle anlatılmıştır. Bu çalışmada nöral hücre aktivasyon fonksiyonları için, fonksiyonel yaklaşım kullanılmıştır. Bu yaklaşımın farkı, parçalı doğrusal ve parabolik yaklaşımlarda kullanılan, toplama ve çarpma modüllerine ek olarak bölme modülünü kullanmasıdır. Avantajı ise bakma tablosu yöntemindeki gibi hafıza gerektirmemesi, parçalı doğrusal yaklaşımdaki gibi kontrol ifadelerine ihtiyaç duymamasıdır. Çalışma Xilinx Kintex 7 xc7k325tffg900-2 FPGA'sı üzerinde gerçeklenmiştir.

Bu tezin organizasyonu şöyledir: Bölüm 2'de yapay sinir ağları ve eğitim algoritmaları tanıtılmış, Bölüm 3'te çok katmanlı algılayıcı ve eğitim algoritmalarınınFPGA üzerinde donanımsal gerçeklenmesi, 4. bölümde eğitim algoritmalarının donanımsal gerçeklenmesine ilişkin deneysel sonuçlar verilmişve 5. bölümde sonuçlar ve yorumlar verilmiştir.

BÖLÜM II

YAPAY SİNİR AĞLARI ve EĞİTİM ALGORİTMALARI

İnsan beyninin fonksiyonları ve çalışma şeklinden esinlenerek yapılan mühendislik çalışmalarında yapay sinir ağlarının temelleri oluşturulmuştur (Haykın 1999). Yapılan bu çalışmalarda beynin davranışlarını kabaca modelleyebilmek amacıyla çeşitli yapay hücre ve ağ modelleri geliştirilmiştir.

İlk çalışmalar nöronların matematiksel olarak modellenmesi üzerine yoğunlaşmıştır. Bu çalışmaların ortaya çıkardığı bulgular, her bir nöronun komşu nörondan bazı bilgiler aldığı ve bu bilgilerin biyolojik sinir hücresi dinamiğinin öngördüğü biçimde bir çıktıya dönüştürüldüğü şeklindeydi.

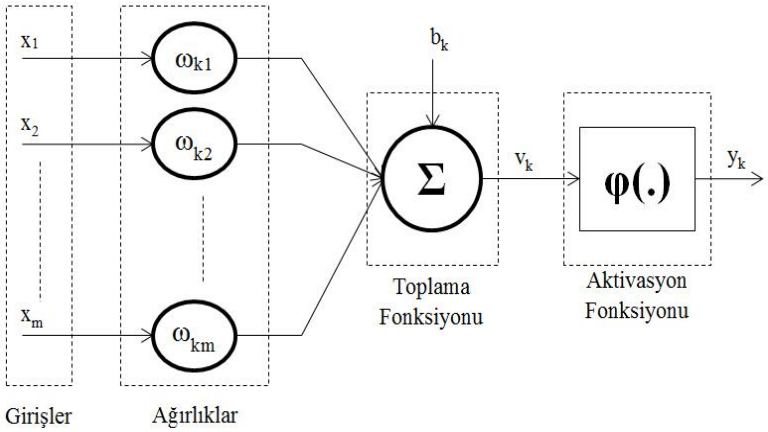
Yapay Sinir Ağları, Yapay Sinir Hücresi (YSH) olarak da adlandırılan bir veya birden çok nöronunçeşitli şekillerde bağlanmasıyla oluşan genel bir hesaplama/modelleme sistemleridir.Bu sistemler genellikle katmanlar şeklinde düzenlenmektedir ve her katmandaki nöronlardabilgi aktarımı paralel olarak gerçeklenmektedir.

2.1.Yapay Sinir Hücresi

Yapay sinir hücresine (YSH) ait genel yapı Şekil 2.1.'de verilmiştir.YSH'lertek başlarına ele alındıklarında üç temel işlevi olan işlemciler olarak düşünülebilmektedir. Birinci işlev, diğer YSH'lerden ya da dış ortamdan hücreye giren bilgilerin bağlantılara ait ağırlık değerleri ile çarpılır. İkinci işlevde ise ağırlıklandırılmış YSH girişleri toplama fonksiyonunda o hücreye ait eşik değeri ile toplanır (Denklem 2.1). Üçüncü işlevde toplama fonksiyonu sonucunda elde edilen değer, hücreye ait aktivasyon fonksiyonundan geçirilir ve aktivasyon fonksiyonu çıktısı YSH çıkışına aktarılır (Denklem 2.2) (Yu ve Deni 1999). Denklem (2.1)'de ω_{ki} , $k.$YSH'nin$i.$ girişine ilişkin ağırlık katsayısını,. x_i YSH'nin$i.$ giriş değerini, b_k ise $k.$YSH'nin eşik değerini,m toplam giriş sayısını ve v_k toplama fonksiyonu sonucunu ifade etmektedir (Öztemel 2003).

$$v_k = \sum_{i=1}^{m} \omega_{ki} x_i + b_k \tag{2.1}$$

$$y_k = \varphi(v_k) \tag{2.2}$$



Şekil 2.1. Yapay sinir hücresi modeli

2.1.1. Aktivasyon fonksiyonları

Hücre modellerinde, hücrenin gerçekleştireceği işleve göre çeşitli tipte aktivasyon fonksiyonları kullanılabilir. Aktivasyon fonksiyonları sabit parametreli ya da uyarlanabilir parametreli seçilebilir. Aşağıda, hücre modellerinde yaygın olarak kullanılan çeşitli aktivasyon fonksiyonları tanıtılmıştır (Haykın 1999).

Doğrusal aktivasyon fonksiyonu: Doğrusal aktivasyon fonksiyonu matematiksel olarak Denklem (2.3)'deki gibi ifade edilebilir. Denklem (2.3)'de α sabit bir sayıdır ve toplama fonksiyonu çıkışında elde edilen değerin çıkışa aktarılma oranını belirlemektedir.

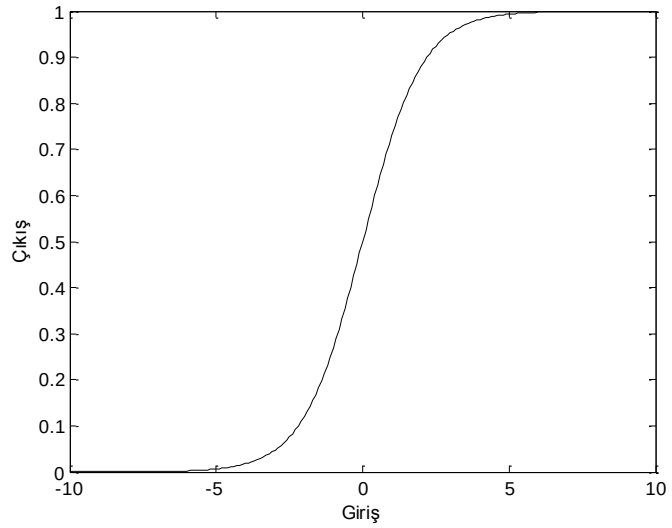
$$y = \alpha v \tag{2.3}$$

Sigmoid aktivasyon fonksiyonu: Sigmoid fonksiyonu doğrusal olmayan problemlerin çözümünde kullanılan YSA'larda tercih edilmesi ile birlikte türev bilgisine ihtiyaç duyan algoritmalar için de avantaj sağlamaktadır.

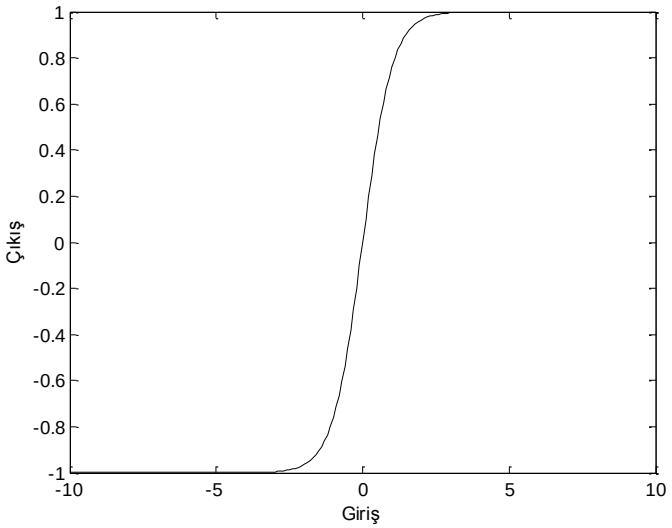
Sigmoid fonksiyonu tek yönlü ve çift yönlü olmak üzere iki şekilde ifade edilmektedir. Logaritmik sigmoidal olarak da adlandırılan tek yönlü sigmoid fonksiyonu [0 1] aralığında çıkış üretmektedir (Şekil 2.2) ve Denklem (2.4)'deki gibi ifade edilmektedir. Tanjant hiperbolik olarak da adlandırılan çift yönlü sigmoid fonksiyonu [-1 1] aralığında çıkış üretmektedir (Şekil 2.3.) ve Denklem (2.5)'deki gibi ifade edilmektedir:

$$\varphi(x) = \frac{1}{1+e^{-x}}, \tag{2.4}$$

$$\varphi(x) = \frac{1-e^{-x}}{1+e^{-x}}. \tag{2.5}$$



Şekil 2.2. Tek yönlü sigmoid aktivasyon fonksiyonu



Şekil 2.3. Çift yönlü sigmoid aktivasyon fonksiyonu

Eşik aktivasyon fonksiyonu: McCulloch-Pitts modeli olarak bilinen eşik aktivasyon fonksiyonlu hücreler, mantıksal çıkış verir ve sınıflandırıcı ağlarda tercih edilir. Eşik aktivasyon fonksiyonu Denklem (2.6)'daki gibi tanımlanır:

$$y(v) = \begin{cases} 1, & v \geq 0 \\ -1, & v < 0 \end{cases}. \tag{2.6}$$

2.2.Çok Katmanlı Algılayıcı

Yapay Sinir Hücrelerinin bağlantı şekillerine, öğrenme kurallarına ve aktivasyon fonksiyonlarına göre çeşitli Yapay Sinir Ağları geliştirilmiştir. Bu yapılardan bazıları(Haykın 1999) :

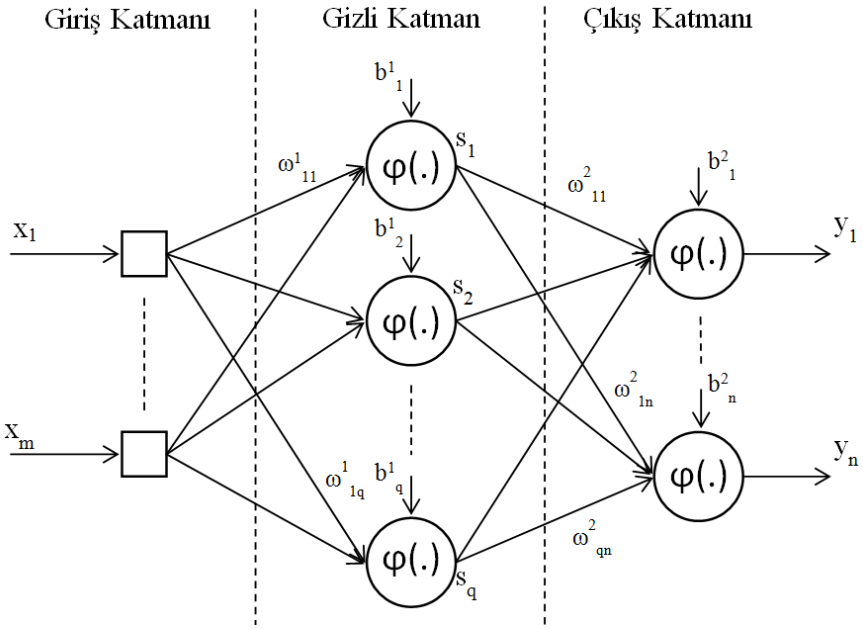
- Çok Katmanlı Algılayıcı (ÇKA) (Multi LayerPerceptron - MLP)
- Radyal Tabanlı Ağ (Radial-Basis Network - RBF)
- Öz Düzenlemeli Harita (Self-OrganizingMap - SOM)
- Geri beslemeli (recurrent) sinir ağları'dır.

ÇKA'da, hücreler katmanlar şeklinde düzenlenir ve bir katmandaki hücrelerin çıkışları bir sonraki katmana ağırlıklandırılarak giriş olarak verilir. ÇKA giriş katmanı, gizli katman ve çıkış katmanı olmak üzere 3 katmandan meydana gelir. Şekil 2.4'te m girişli, tek gizli katman ve bu katmanda q adet hücre ile birlikte çıkış katmanından n adet hücreden oluşan ÇKA mimarisi gösterilmektedir. ÇKA mimarisinde giriş değerlerinin ağırlıklandırılmasından sonra gizli katmandaki her bir hücredeki çıkışa aktarım işlemli Denklem (2.8)'de gösterilmiştir. Denklem (2.8)'de u_k^1, 1. katmandaki ağırlıklandırılmış girişlerin gizli katmandaki $k.$ hücreye ait toplam değerini gösterir. m toplam giriş sayısını, x giriş değerini, ω ağırlık değerini göstermektedir. Ağırlık değerlerinin gösteriminde üst indis parametresi katman numarasını göstermektedir. Alt indis değerlerinden ilk parametre ilgili ağrılık giriş indisini ve ikinci parametre ise ağırlıklandırılmış giriş değerinin hangi hücreye ait olduğunu göstermektedir. Gizli katmanda elde edilen çıkış değerleri ise çıkış katmanında bulunana hücrelere giriş olarak verilmekte ve bu değerlerin ağırlıklandırma işleminden sonra çıkışa aktarımı ise

Denklem (2.9)'da verilmiştir. Denklem (2.9)'da n çıkış katmanındaki hücre sayısını göstermektedir.

$$\left.\begin{aligned} u_k^1 &= \sum_{i=1}^{m} \omega_{ik}^1 x_i + b_k \\ s_k &= \varphi\left(u_k^1\right) \end{aligned}\right\} \quad k = 1, \cdots, q \tag{2.8}$$

$$\left.\begin{aligned} u_k^2 &= \sum_{i=1}^{q} \omega_{ik}^2 s_i + b_k^2 \\ y_k &= \varphi\left(u_k^2\right) \end{aligned}\right\} \quad k = 1, \cdots, n \tag{2.9}$$



Şekil 2.4. 3 katmanlı ÇKA mimarisi

2.3.Yapay Sinir Ağlarının Eğitilmesi

Yapay sinir ağları, öğrenme sürecinden sonra bilgiyi toplama, hücreler arasındaki bağlantı ağırlıkları ile bu bilgiyi saklama ve genelleme yeteneğine sahip paralel dağıtık bir işlemcidir. Öğrenme süreci, arzu edilen amaca ulaşmak için YSA ağırlıklarının yenilenmesini sağlayan öğrenme algoritmalarını ihtiva eder.

Öğrenme algoritmaları öğrenme denen olguyu matematiğin kuralları ile ölçülebilir büyüklüklere dönüştürerek başarım ölçütünün oluşturulmasına ve bu ölçütün zaman içerisinde artırılmasını sağlayacak parametre değişikliklerinin hesaplanmasına dayanır.

Öğrenme algoritmaları YSA çıkışında elde edilen hata bilgisini kullanarak parametre değişikliklerini belirlerler. Denklem (2.10)'da YSA çıkışında elde edilen hata değeri gösterilmektedir. Denklem (2.10)'da d_k ağ eğitimi esnasında giriş olarak verilen veri setine ilişkin $k.$ çıkış hücresine ait beklenen değerini göstermektedir. e_k ise giriş veri setine ilişkin $k.$ çıkış hücresinde oluşan hata değerini göstermektedir.

$$e_k = d_k - y_k \tag{2.10}$$

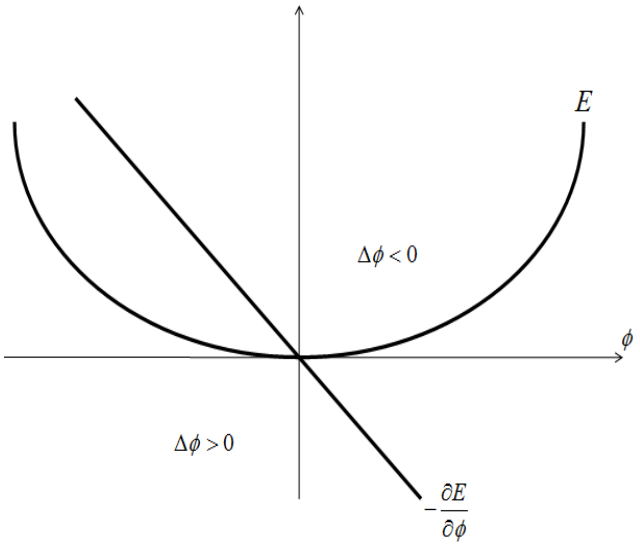
Bu çalışmada ÇKA eğitiminde eğim bilgisi kullanan geriye yayılım algoritması ile Levenberg & Marquardt algoritması ve küresel arama özelliği olan parçacık sürü optimizasyon (particle swarm optimization) algoritması ile yapay arı kolonisi (artificial bee colony) algoritması parametre güncelleme işlemlerini yapmak amacı ile kullanılmıştır. Bu algoritmalar aşağıda kısaca tanıtılmıştır.

2.3.1. Geriye yayılım algoritması

Geriye yayılım algoritması, ağa gösterilen o anki veri setinden elde edilen hata değeri ile eğim düşüm bilgisini hesaplayarak parametre güncelleme işlemini yapmaktadır (Şekil 2.5). Denklem (2.11)'de ağa eğitimi esnasında giriş olarak verilen veri setine ilişkin ağ çıkışında elde edilen hataların karelerinin toplamı ile elde edilen ölçüt fonksiyonunun (E) en küçük değeri aldığı noktanın Denklem (2.12) ile verilen kural ile yinelemeli olarak bulunabilmesine dayalıdır. Denklem (2.12)'de J Jacobian matrisini, ϕ ise güncellenecek ağ parametresini göstermektedir (Haykın 1999).

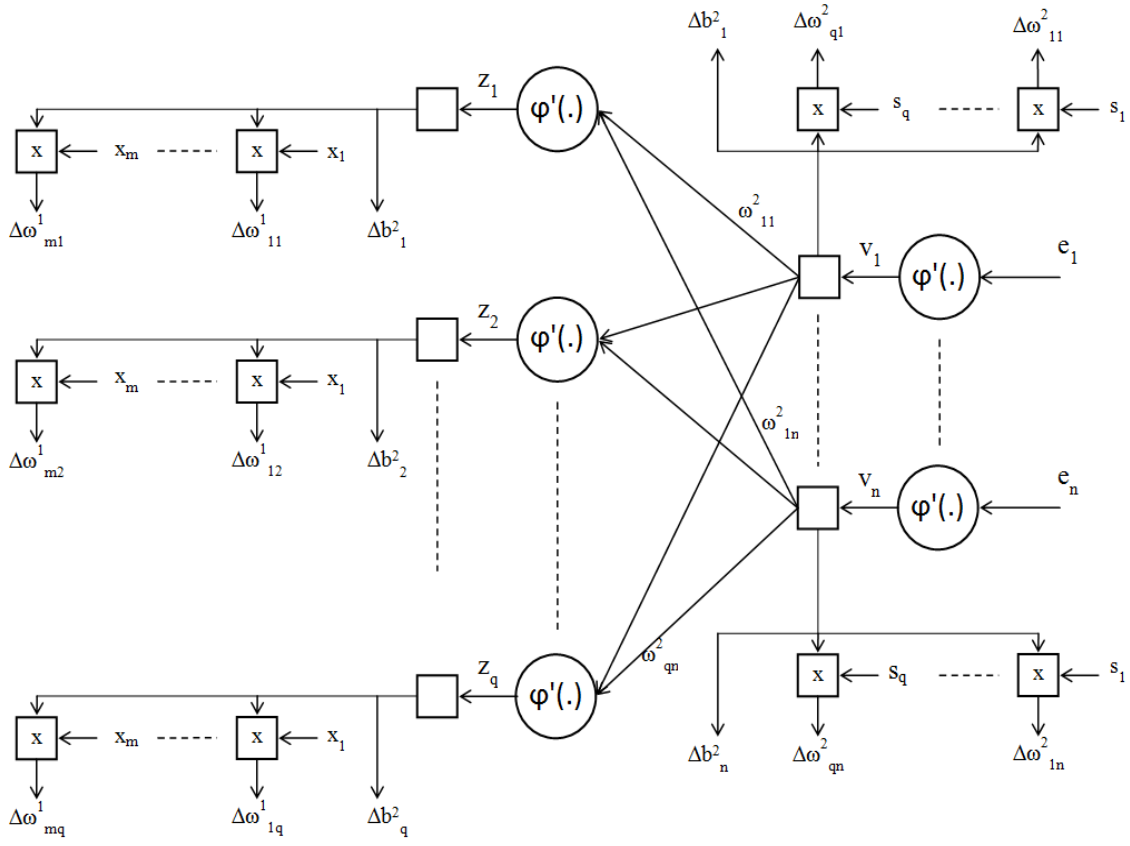
$$E = \frac{1}{2}\sum_{k=1}^{n} e_k^2 \tag{2.11}$$

$$\Delta\phi = -\frac{\partial E}{\partial \phi} = -\frac{\partial J}{\partial \phi}\mathbf{e} \tag{2.12}$$



Şekil 2.5. Eğim düşümü grafik gösterimi

ÇKA için ağ parametre güncelleme işlemi Şekil 2.6'da verilen öbek şeması ile gösterilmiştir.



Şekil 2.6. Geriye yayılım algoritması ile ÇKA parametre güncelleme öbek şeması

Şekil (2.6)'da gösterilen **v** değerleri Denklem (2.13)'deki gibi hesaplanır. Çıkış katmanındaki eşik parametrelerinin güncelleme değeri Denklem (2.14)'deki gibi hesaplanır. Gizli katmanda bulunan hücrelerin çıkış değerlerine ait ağırlıklandırma parametrelerinin güncelleme değeri Denklem (2.15)'deki gibi hesaplanır. Şekil (2.6)'da gösterilen **z** değerleri Denklem (2.16)'daki gibi hesaplanır. Gizli katmanındaki eşik parametrelerinin güncelleme değeri Denklem (2.17)'deki gibi hesaplanır. Girişlere ait ağırlıklandırma parametrelerinin güncelleme değeri Denklem (2.18)'deki gibi hesaplanır.

$$v_k = e_k \varphi_k^{'}(.); \quad k = 1,\cdots,n \tag{2.13}$$

$$\Delta b_k^2 = v_k; \quad k = 1,\cdots,n \tag{2.14}$$

$$\Delta \omega_{ik}^2 = s_i v_k; \quad k = 1,\cdots,n; i = 1,\cdots,q \tag{2.15}$$

$$z_i = \varphi_k^{'}(.) \sum_{k=1}^{n} v_k \omega_{ik}^2; \quad i = 1,\cdots,q \tag{2.16}$$

$$\Delta b_i^1 = z_i; \quad i = 1,\cdots,q \tag{2.17}$$

$$\Delta \omega_{ki}^1 = x_k z_i; \quad i = 1,\cdots,q; k = 1,\cdots,m \tag{2.18}$$

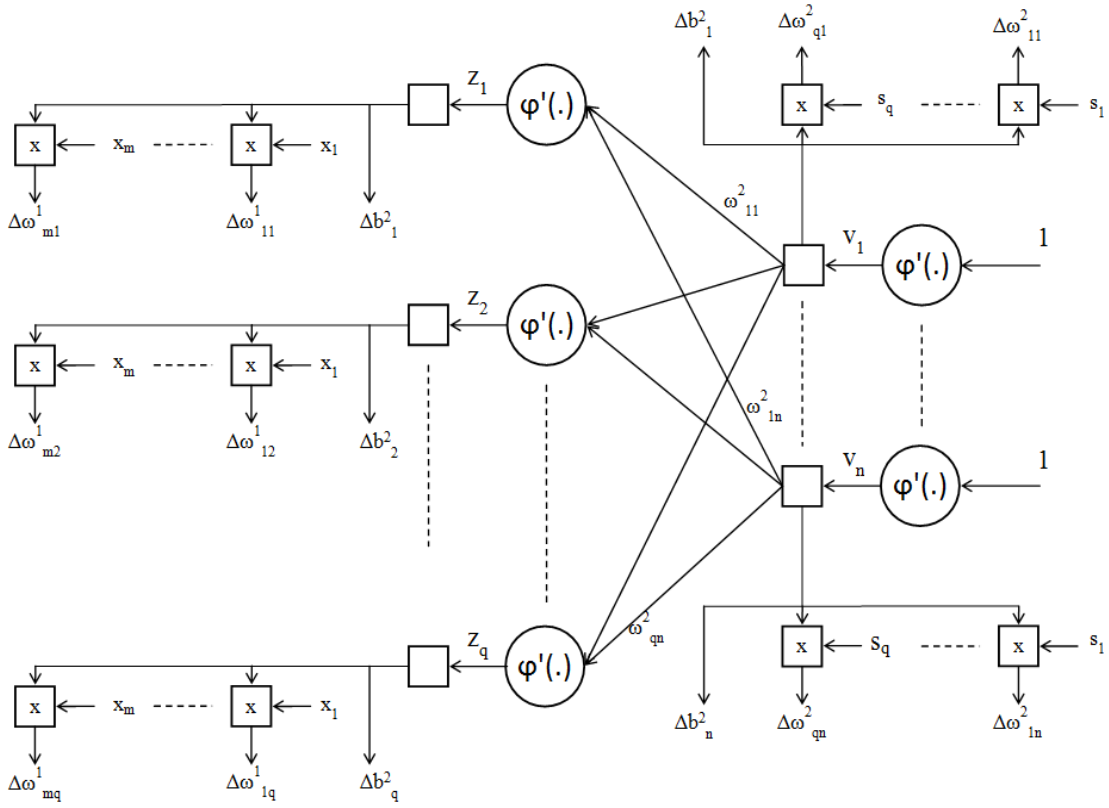
2.3.2. Levenberg & Marquardt algoritması

Eğim düşüm ve Newton algoritmalarından türetilerek, LM algoritmasına ait parametre güncelleme işlemi Denklem(2.19)'da verilmiştir. Denklem (2.19)'da **ω** ağırlık vektörü, I birim matris, μ kombinasyon katsayısıdır. J, $\left[\left(P \times n\right), N\right]$ boyutunda Jacobian matrisini, **e** $\left[\left(P \times n\right), 1\right]$ boyutunda hata vektörünü göstermektedir. P, eğitim örnek sayısını, n çıkış sayısını ve N ağırlık sayısını göstermektedir (Wilamowski ve Chen 1999).

$$\Delta \boldsymbol{\omega} = (J^T J + \mu I) J^T \mathbf{e} \tag{2.19}$$

Levenberg & Marquardt algoritması parametre güncelleme işlemlerini, tüm giriş örnek değerleri için oluşturduğu hata vektörünü ve Jacobian matrisini kullanarak yapmaktadır. Denklem (2.20)'de Jacobian matrisinin elde edilmesi gösterilmektedir. Jacobian matrisi

parametrelerinin oluşumu ise Şekil (2.7)'de gösterilmiştir. Hata vektörü ise Denklem (2.21)'deki gibi elde edilir.



Şekil 2.7. LM algoritması ile ÇKA parametre güncelleme öbek şeması

$$J = \begin{bmatrix} \frac{\partial e_{11}}{\partial \omega_1} & \frac{\partial e_{11}}{\partial \omega_2} & \cdots & \frac{\partial e_{11}}{\partial \omega_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{1n}}{\partial \omega_1} & \frac{\partial e_{1n}}{\partial \omega_2} & \cdots & \frac{\partial e_{1n}}{\partial \omega_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{P1}}{\partial \omega_1} & \frac{\partial e_{P1}}{\partial \omega_2} & \cdots & \frac{\partial e_{P1}}{\partial \omega_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{Pn}}{\partial \omega_1} & \frac{\partial e_{Pn}}{\partial \omega_2} & \cdots & \frac{\partial e_{Pn}}{\partial \omega_N} \end{bmatrix} \tag{2.20}$$

$$\mathbf{e} = \begin{bmatrix} e_{11} \\ \cdots \\ e_{1n} \\ \cdots \\ e_{P1} \\ \cdots \\ e_{Pn} \end{bmatrix} \tag{2.21}$$

Denklem (2.19)'da μ ayarlanabilir bir parametredir (Denklem (2.22)). Eğer bu parametre çok büyükse yöntem eğim düşüm metodu gibi davranmaktadır. Eğer çok küçükse Newton metodu gibi davranmaktadır (Hashemipoor, Suratgar ve Fard 2010).

$$\mu(t) = \begin{cases} \mu(t)k & E(t) > E(t-1) \\ \mu(t)/k & E(t) \le E(t-1) \end{cases} \tag{2.22}$$

2.3.3. Parçacık sürü optimizasyon algoritması

Parçacık sürüsü optimizasyon (Particle Swarm Optimization) algoritması (PSO); kuş sürülerinin davranışlarından esinlenilerek geliştirilmiş popülasyon tabanlı rastgele arama algoritmasıdır(Kennedy ve Eberhart 1995).

Algoritma başlangıçta rastgele atanmış bireylerle (parçacık) başlatılır. Her bir iterasyonda, parçacıkların hızları ve pozisyonları güncellenir. Sürüdeki her parçacık problem için bir çözüm adayı olarak değerlendirilebilir. N elemana sahip sürünün *i*. parçacığı (2.23)'de verilen bir vektör olarak tanımlanır.

$$\vec{p}_i = \left[p_{i1}, \quad p_{i2}, \quad \cdots, \quad p_{iN} \right] \tag{2.23}$$

Her parçacık arama uzayında, o iterasyona kadar kendisinin en iyi pozisyonu (yerel en iyi, p_{yei} (2.24)) ve tüm popülasyon için o ana kadar en iyi (küresel en iyi, p_{kei} (2.25)) pozisyona sahip parçacığın pozisyonuna bağlı olarak konum değiştirir.

$$\vec{p}_{yei} = \left[p_{yei1}, \quad p_{yei2}, \quad \cdots, \quad p_{yeiN} \right] \tag{2.24}$$

$$\vec{p}_{kei} = \left[p_{kei1}, \quad p_{kei2}, \quad \cdots, \quad p_{keiN} \right] \tag{2.25}$$

$i.$ parçacığın pozisyonundaki değişim hızı parçacık hızı (v_i) olarak adlandırılır(2.26).

$$\vec{v}_i = \left[v_{kei1}, \quad v_{kei2}, \quad \cdots, \quad v_{keiN} \right] \tag{2.26}$$

Parçacık hızlarının güncellenmesi için literatürde farklı yöntemler mevcuttur. Bu çalışmada hızların güncellenmesi (2.27)'de verilen yöntem kullanılmıştır.

$$v_i(n+1) = \xi \left[v_i(n) + \alpha_1 r_1 \left(p_{yei,i} - p_i(n) \right) + \alpha_2 r_2 \left(p_{kei,i} - p_i(n) \right) \right] \tag{2.27}$$

Denklem (2.27)'de α_1 ve α_2 öğrenme sabitleri, r_1 ve r_2 [0-1) aralığında düzgün dağılımlı rastgele sayılar; ξ sınırlama faktörüdür. Hızlarının belirlenmesinden sonra parçacıklar (2.28)'e göre güncellenir.

$$p_i(n+1) = p_i(n) + v_i(n) \tag{2.28}$$

2.3.4. Yapay arı kolonisi algoritması

Yapay Arı Koloni (Artificial Bee Colony, ABC) algoritması da arıların yiyecek arama davranışını modelleyerek geliştirmiştir (Karaboğa 2005).

YAK algoritması en fazla nektara sahip kaynağın yerini bulmaya çalışarak uzaydaki çözümlerden problemin çözümünü bulmaya çalışmaktadır(Akay ve Karaboğa 2009). Algoritma, her kaynak için bir arı görevlendirmektedir. Bu nedenle arı sayısı kaynak sayısına eşittir. YAK algoritmasına ait adımlar aşağıdaki gibidir (Karaboğa ve Akay 2009).

1. Başlangıç popülasyonu oluştur.
2. Tekrarla
3. İşçi arıları kaynaklara gönder ve nektar hesapla
4. Gözcü arıları kaynaklara gönder ve nektar hesapla
5. .Yeni kaynak için kâşif arıları gönder.

6. En iyi kaynağı hafızaya kaydet
7. Koşul şartları sağlanmamışsa 2. Adıma dön aksi taktirde algoritmayı sonlandır.

İşçi arılar, bulunduğu kaynaktan yeni bir kaynak belirler ve bu yeni kaynağın eski kaynaktan daha iyi olup olmadığı hakkında bilgi edinir. Yeni kaynağın eski kaynaktan iyi olması durumunda bu kaynağı hafızaya alır. Tanımlanan bu hareket (2.29)'da verilmiştir. Denklem (2.29)'da x o anki kaynağı, V ise güncellenmiş kaynağı göstermektedir. i ve j ise birbirine eşit olmayan rastgele tam sayılardır. θ ise rastgele gerçel sayıdır.

$$V_{ij} = x_{ij} + \theta_{ij} \times \left(x_{ij} - x_{kj} \right) \tag{2.27}$$

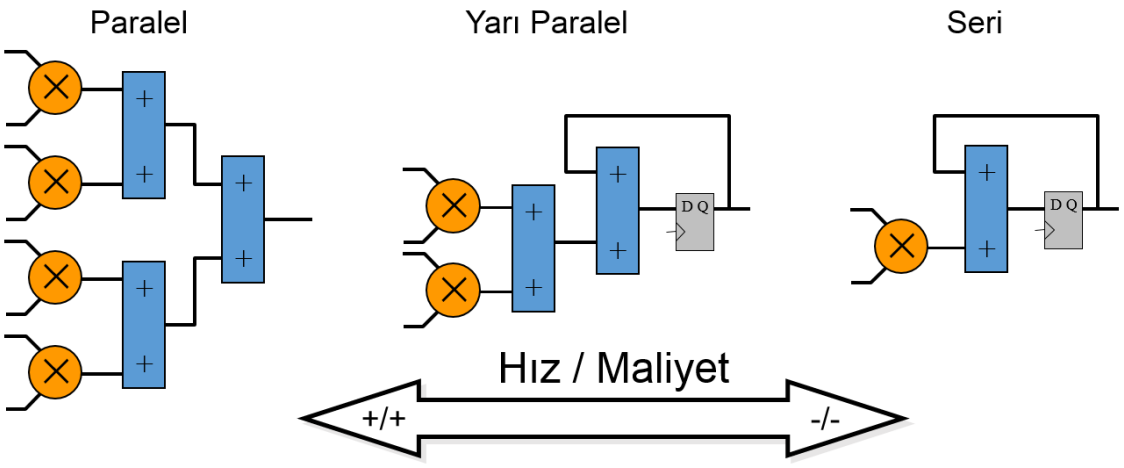
BÖLÜM III

ÇOK KATMANLI ALGILAYICI ve EĞİTİM ALOGİRMALARININ FPGA ÜZERİNDE DONANIMSAL GERÇEKLENMESİ

Bu çalışmada çok katmanlı algılayıcı FPGA üzerinde donanımsal olarak gerçeklenmiş ve aynı zamanda Bölüm II'de verilen eğitim algoritmaları ile ağ parametreleri donanımsal olarak belirlenmiştir.

3.1 Yapay Sinir Hücresinin Donanımsal Gerçeklenmesi

Paralel mimariye sahip bir yapay sinir hücresinin (YSH) donanımsal gerçeklenmesi işlem hızında artış sağlarken donanım kaynaklarının kullanımında artırmaktadır. Şekil 3.1.'de paralelden seriye veri işleme tekniklerinin hız/maliyet değişimleri gösterilmektedir. Paralel gerçeklenen bir mimari de hız artmakta fakat çarpıcı ve toplayıcı sayısında meydana gelen artış nedeniyle donanım maliyeti de artmaktadır. Seri gerçeklenen bir mimaride aynı toplayıcı ve çarpıcının sıralı şekilde kullanılmasından dolayı maliyetin azalmasıyla birlikte hız da azalmaktadır.



Şekil 3.1. Veri işleme hız / maliyet grafiği (Gallagher 2004)

FPGA'da YSA'nın donanımsal gerçeklenmesinde, veri gösterimi ve aktivasyon fonksiyonunun önceden belirlenmesi gerekmektedir. Bu tezde kullanılan aktivasyon fonksiyonları ve sayı formatı Bölüm 3.1.1 ve Bölüm 3.1.2'de tanıtılmıştır.

3.1.1. Aktivasyon fonksiyonu

YSH modellerinde, hücrenin gerçekleştireceği işleve göre çeşitli tipte aktivasyon fonksiyonları kullanılabilir. Aktivasyon fonksiyonları sabit parametreli ya da uyarlanabilir parametreli seçilebilir. Bu tezde, yaygın olarak kullanılan tanjant hiperbolik ve logaritmik sigmoidal aktivasyon fonksiyonlarındaki üssel ifadenin FPGA'da doğrudan gerçeklenememesi sebebiyle, bu fonksiyonların yaklaşımları FPGA'de donanımsal olarak gerçeklenmiştir(Çavuşlu, Karakuzu 2011). Bölüm 3.1.1.1 ve Bölüm 3.1.1.2'de bu yaklaşımların donanımsal gerçeklenmeleri kısaca tanıtılacaktır.

3.1.1.1. Logaritmik sigmoidal aktivasyon fonksiyonu

Logaritmik sigmoidal fonksiyonu [0 1] aralığında çıkış veren, yapay sinir ağlarında çok tercih edilen bir aktivasyon fonksiyonudur. Logaritmik sigmoidal fonksiyonunun matematiksel ifadesi Denklem (3.1)'de verilmiştir.

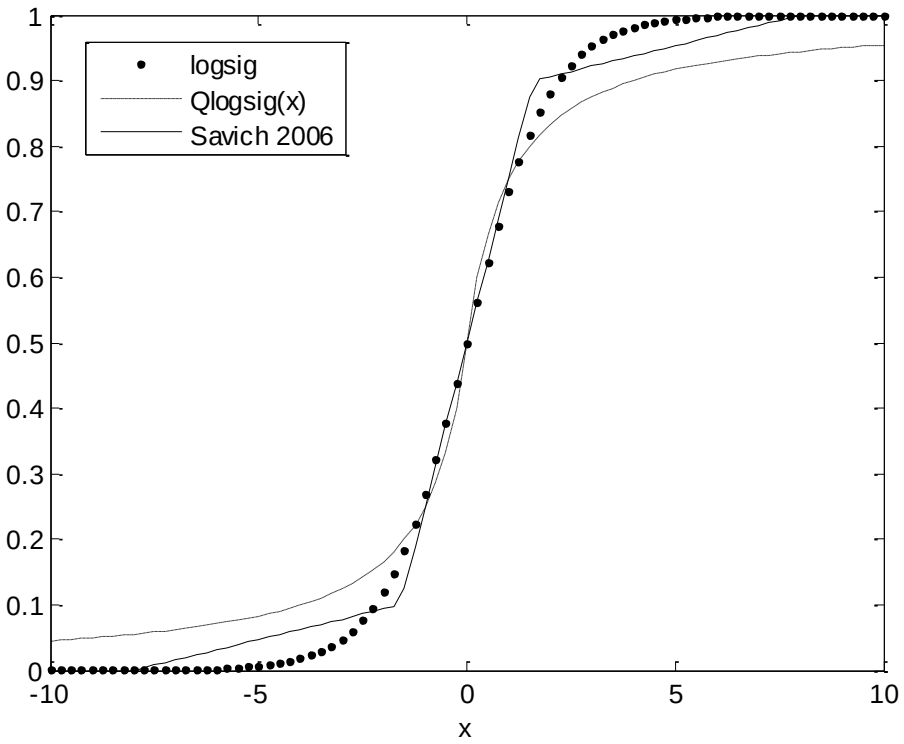
$$\log sig\left(x\right) = \frac{1}{1+e^{-x}} \tag{3.1}$$

Logaritmik sigmoidal ifadeyi donanımsal gerçeklemek için, Denklem (3.2)'de verilen yaklaşım kullanılmıştır (Elliot 1993).

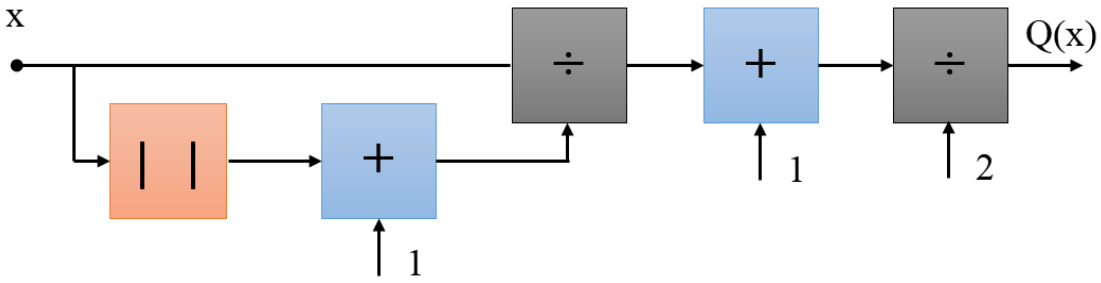
$$Q_{logsig}\left(x\right) = \frac{1}{2}\left[1+\frac{x}{1+\left|x\right|}\right] \tag{3.2}$$

Şekil 3.2'de (3.1) ve (3.2)'de verilen logaritmik sigmoidal ve yaklaşımı ile Savich ve diğerlerinin kullandığı logaritmik sigmoidal doğrusal parçalı yaklaşımlarının [-10 10] aralığında davranışları karşılaştırmalı olarak gösterilmiştir. Şekil 3.3'te logaritmik sigmoidal fonksiyonunun matematiksel yaklaşımına ait blok diyagram, Şekil 3.4'de ise bu yaklaşımın gerçeklendiği VHDL kodu verilmiştir. Görüleceği üzere yaklaşımın gerçeklenebilmesi için 1 mutlak değer, 2 toplama ve 2 bölme modülü gerekmektedir. Mutlak değer alma işlemi, kayan noktalı sayılarda sayının işaret bitinin '0' yapılmasıyla sabit noktalı sayılarda ise 2'ye tümleyen işlemi ile gerçeklenir. 2'ye bölme işlemi ise kayan noktalı sayılarda sayının üs (e) kısmının 1 azaltılmasıyla, sabit noktalı sayılarda ise 1 bit sağa kaydırma işlemi ile gerçeklenir(Çavuşlu, Karakuzu 2011). Şekil 3.4'den

anlaşılacağı üzere, toplama, çarpma ve bölme işlemleri için önceki çalışmalarımızda oluşturulan sırasıyla "*add*", "*mul*" ve "*divide*" kütüphaneleri kullanılmıştır.



Şekil 3.2. logisg(x), $Q_{logsig}(x)$ ve Savich ve diğerleri 2006'da kullanılan parçalı doğrusal fonksiyonlarının kıyaslaması



Şekil 3.3. Logaritmik sigmoidal fonksiyonu yaklaşımının FPGA'da gerçekleme blok yapısı

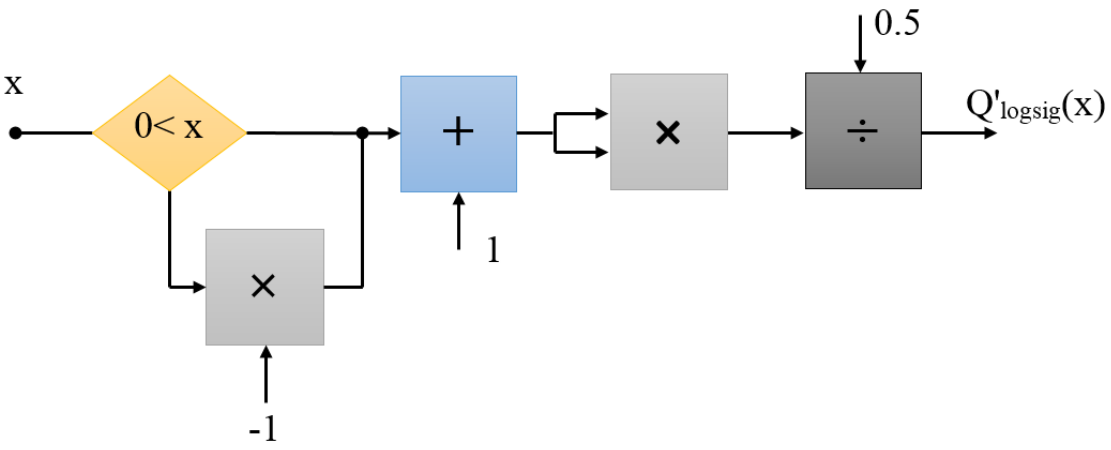
```
case kont is
   when "00000" =>
toplam_1 <= add(bir, '0' & giris(30 downto 0));
kont <= "00001";

when "00001" =>
bolme_sonuc <= divide(giris, toplam_1)
kont <= "00010";

when "00010" =>
     toplam2 <= add(bir, bolme1_sonuc);
kont <= "00011";
```

Şekil 3.4. Logaritmik sigmoidal fonksiyonu yaklaşımının FPGA'da gerçekleme VHDL kodu (Çavuşlu, Karakuzu 2011).

Gradyen tabanlı eğitimde kullanılması açısından, logaritmik sigmoidal ve yaklaşımının türevleri sırasıyla Denklem (3.3) ve Denklem (3.4)'de verilmiştir. Denklem (3.4)'de verilen ifadenin FPGA'da gerçeklenme blok yapısı Şekil 3.5'de, bu blok yapının VHDL dili program kodu da Şekil 3.6'da verilmiştir. Şekil 3.7'de ise orijinal logsig, önerilen yaklaşım ve Savich ve diğerleri 2006'da kullanılan yaklaşımın türevleri karşılaştırmalı olarak gösterilmiştir (Çavuşlu, Karakuzu 2011).

$$\log sig'(x) = \log sig(x)\left(1 - \log sig(x)\right) \tag{3.3}$$

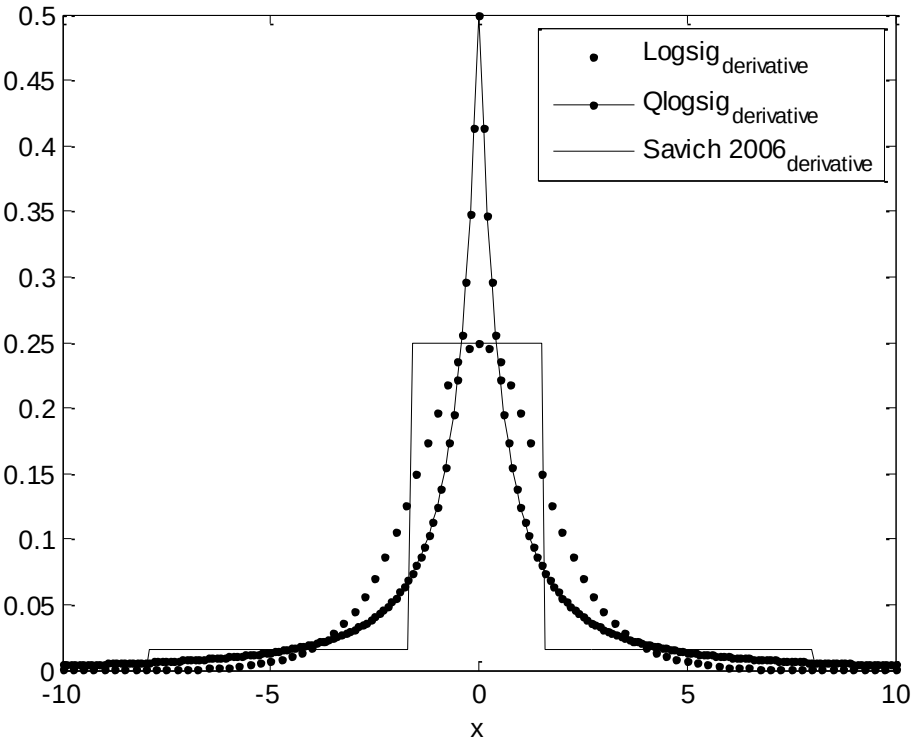
$$Q'_{\log sig}(x) = \begin{cases} \dfrac{0.5}{(1-x)^2}, & x < 0 \\ \dfrac{0.5}{(1+x)^2}, & x \geq 0 \end{cases} \tag{3.4}$$



Şekil 3.5. Logsig yaklaşım fonksiyonu türevinin FPGA'de gerçekleme blok yapısı

```
case kont is
when "00000" =>
if giris(31) = '1' then
toplam_1 <= add(bir, (not giris(31)) & giris(30 downto 0));
kont <= "00001";
else
toplam_1 <= add(bir, giris);
kont <= "00001";
end if;

  when "00001" =>
carpma_1  <= mul(toplam_1, toplam_1);
     kont <= "00010";

when "00010" =>
cikis <= divide(yarim, carpma_1)

when others => null;
end case;
```

Şekil 3.6. Logsig yaklaşım fonksiyonu türevinin FPGA'de gerçekleme VHDL kodu (Çavuşlu, Karakuzu 2011).



Şekil 3.7. logisg(x), Qlogsig(x) ve Savich ve diğerlerinin kullandığı parçalı doğrusal fonksiyonlarının türevlerinin kıyaslaması (Çavuşlu, Karakuzu 2011).

3.1.1.2. Tanjant hiperbolik

Tanjant hiperbolik fonksiyonu, türevi alınabilir, sürekli ve doğrusal olmayan bir fonksiyon olması nedeniyle doğrusal olmayan problemlerin çözümünde kullanılan YSA'larda tercih edilir. Tanjant hiperbolik fonksiyonunun matematiksel ifadesi Denklem (3.5)'de verilmiştir.

$$tanh(x) = \frac{1-e^{-x}}{1+e^{-x}} \tag{3.5}$$

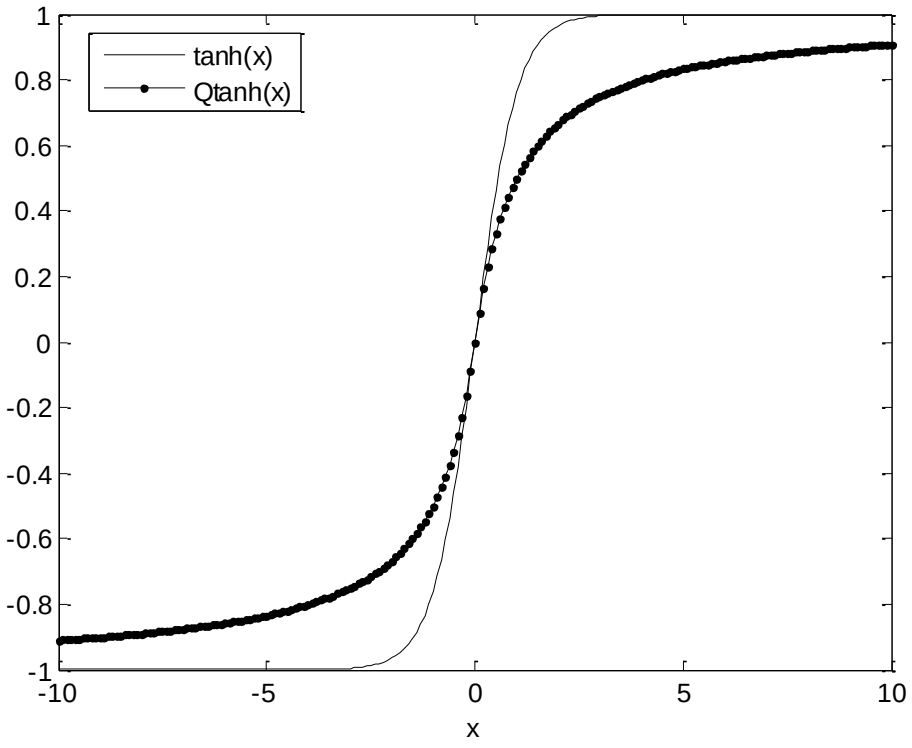
Eşitlikteki üstel ifade donanımsal gerçeklenemediğinden Denklem (3.5) yerine, Denklem (3.6)'da verilen matematiksel yaklaşım kullanılmıştır (Elliot 1993).

$$Q_{\tanh}(x) = \frac{x}{1+|x|} \tag{3.6}$$

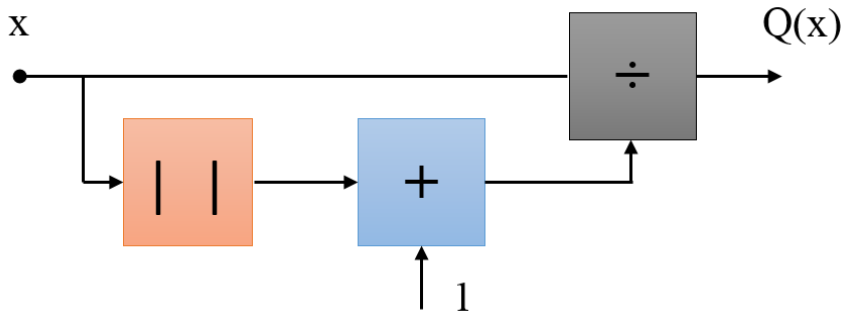
Şekil 3.8'de *tanh(x)* ve $Q_{tanh}(x)$ fonksiyonlarının [-10 10] aralığında davranışları verilmiştir. Şekil 3.9'da ise Denklem (3.6)'da verilen yaklaşımın gerçekleme blok yapısı verilmiştir. Gradyen tabanlı eğitimde kullanılması açısından, tanjant hiperbolik ve yaklaşımının türevleri Denklem (3.7) ve Denklem (3.8)'de verildiği gibidir. Şekil 3.10'da bu yaklaşımın türevinin orijinalinin türevi ile karşılaştırmalı davranışı verilmiştir. Şekil 3.11'de ise Denklem (3.8)'i gerçekleme blok yapısı verilmiştir (Çavuşlu, Karakuzu 2011)..

$$\tanh'(x) = 1 - \tanh^2(x) \tag{3.7}$$

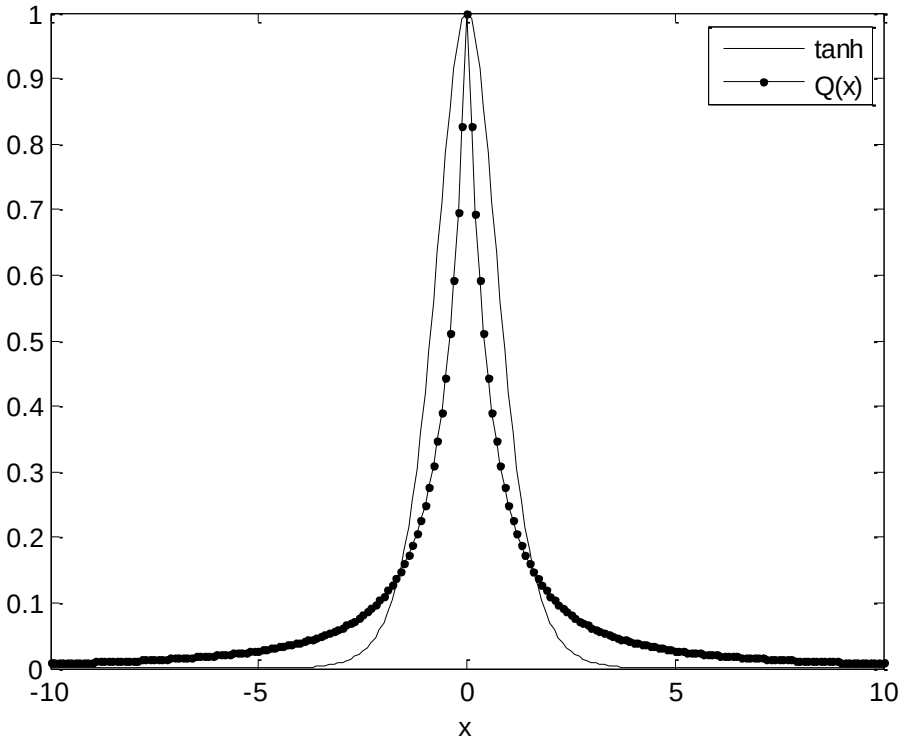
$$Q\tanh'(x) = \begin{cases} \dfrac{1}{(1-x)^2}, & x < 0 \\ \dfrac{1}{(1+x)^2}, & x \geq 0 \end{cases} \tag{3.8}$$



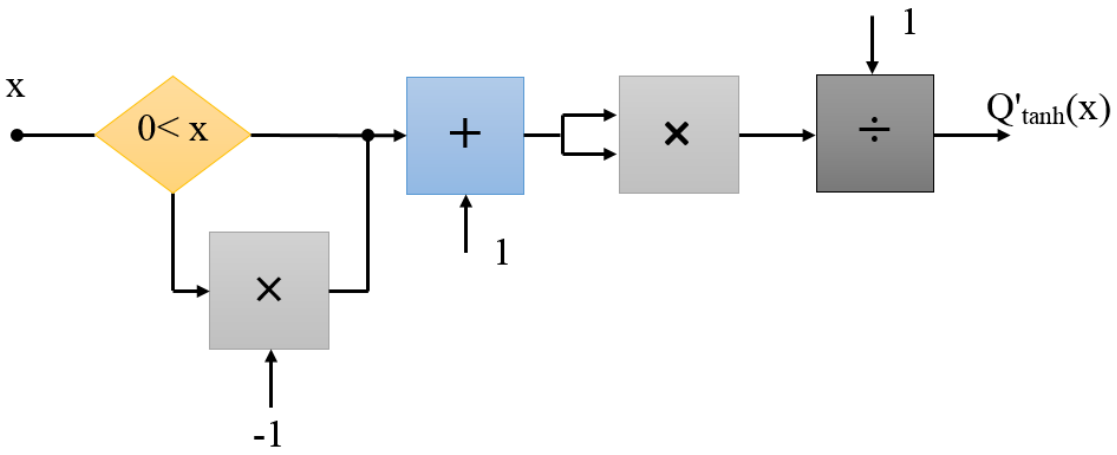
Şekil 3.8. tanh(x) ve Qtanh(x) fonksiyonlarının kıyaslaması (Çavuşlu, Karakuzu 2011).



Şekil 3.9. Tanjant hiperbolik fonksiyonunun matematiksel yaklaşımının FPGA'da gerçekleme blok yapısı



Şekil 3.10. tanh(x) ve Qtanh(x) fonksiyonları türevlerinin kıyaslaması (Çavuşlu, Karakuzu 2011).



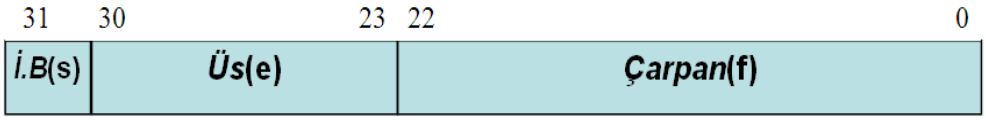
Şekil 3.11. Tanjant hiperbolik yaklaşımı fonksiyonu türevinin FPGA'da gerçekleme blok yapısı

3.1.2 Sayı formatı

Sayısal ortamda aritmetik işlemleri yapabilmek için kullanılan sayı formatlarından duyarlılık bakımından en göze çarpanı, kayan noktalı sayılar ve sabit noktalı sayılardır. Bu tezde, kullanıcıya sağladığı hassasiyet ve dinamiklik sebebiyle IEEE 754 formatındaki tek duyarlıklı kayan noktalı sayılar ile çalışılmıştır.

3.1.2.1 Kayan noktalı sayılar

Kayan noktalı sayılar, sayısal ortamda gerçek sayıların bir gösterim biçimidir. Hassas işlem yapma özelliğine sahip kayan noktalı sayılar, dinamiklik özelliği ile de geniş sayı gösterim aralığı imkânı sunmaktadırlar. En çok kullanılan kayan noktalı sayı gösterim standardı IEEE 754 standardıdır. Bu çalışmada tek duyarlı (32 bit) IEEE 754 standardı kullanılmış olup, bu gösterim formatı Şekil 3.12'de verilmiştir.



Şekil 3.12. IEEE 754 Kayan-noktalı sayı gösterimi

Denklem (3.9)'da gerçel bir sayının kayan-noktalı sayılarda gösterimi verilmiştir. Denklem (3.9)'da s işaret bitini temsil eder. İşaret biti "0" ise sayının pozitif, "1" ise

negatif olduğunu belirtir. e üs değerini ve f çarpan değerini gösterirken, n üs değerini tutan bit sayısıdır. Çarpan değeri daima sıfır ile bir arasında olmalıdır.

Denklem (3.9)'da tanımlanan *bias* değeri Denklem (3.10)'daki gibi hesaplanır. e değeri ise Denklem (3.11)'deki gibi hesaplanır.

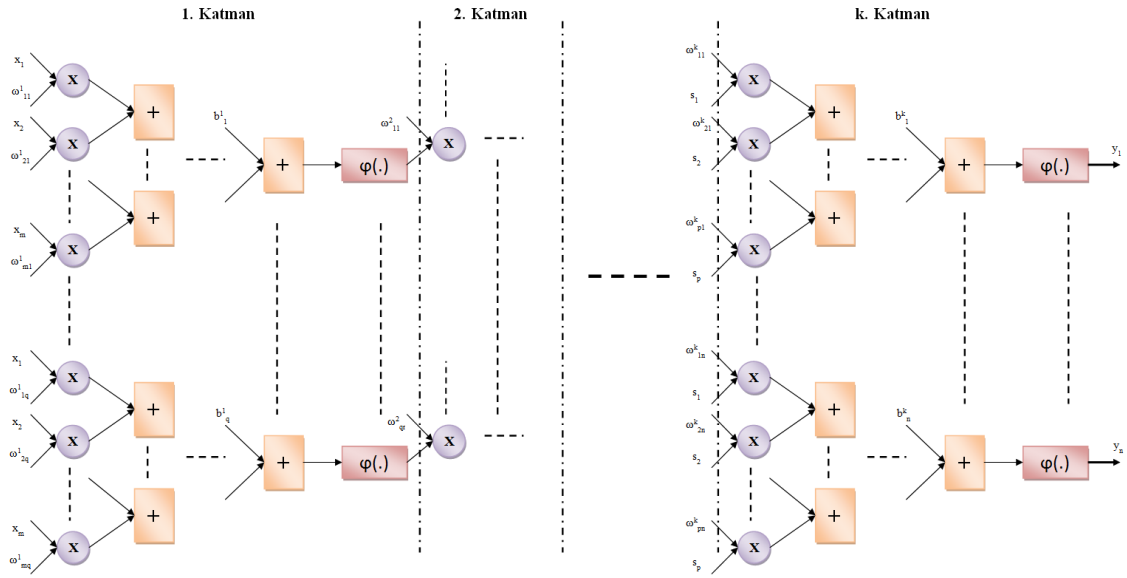
$$Sayı = (-1)^s \, 2^{e-bias} \, (1.f) \tag{3.9}$$

$$bias = 2^{n-1} - 1 \tag{3.10}$$

$$e = bias + floor(\log_2^{Sayı}) \tag{3.11}$$

3.2. Çok Katmanlı Algılayıcı Tipi Yapay Sinir Ağının Donanımsal Gerçeklenmesi

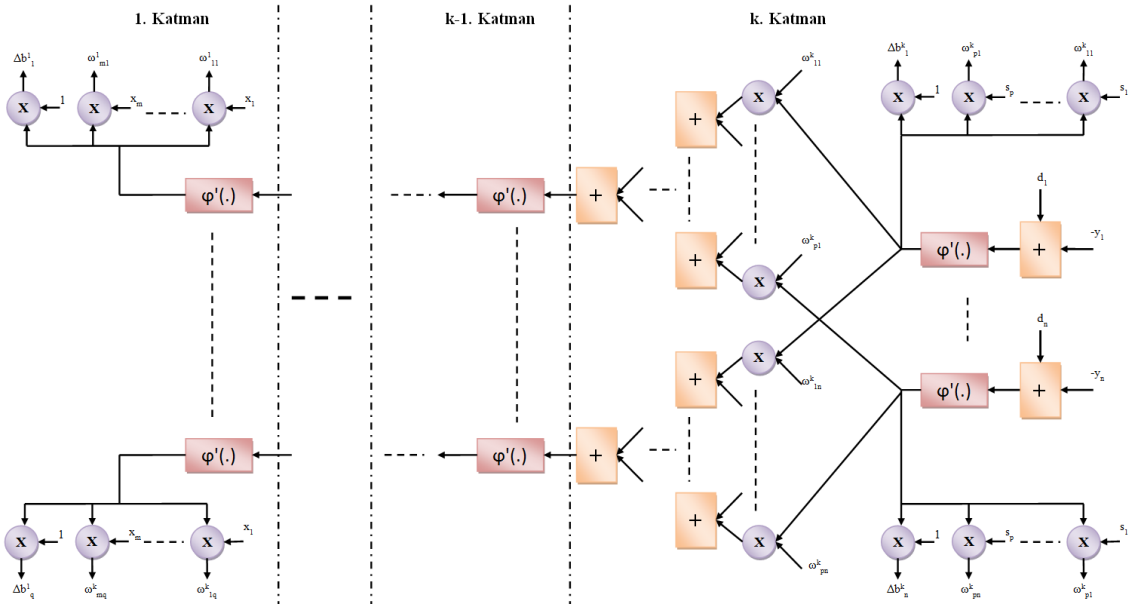
Çok Katmanlı Algılayıcılar, YSH'lerin farklı şekillerde bağlanmasından meydana gelmektedir. Şekil 3.13'te paralel mimariye sahip k adet katmandan meydana gelen m girişli, n çıkışlı bir ÇKA gösterilmektedir. Şekil 3.13'den de görüleceği üzere 1. katmanda tüm çarpıcılar aynı anda işlem sonuçlarını üretebilmektedir. YSA'ların doğasında var olan paralel veri işleme özelliğinin gerçeklenebilmesi için Şekil 3.13'te gösterilen mimariye benzer bir yapının tasarlanması gerekmektedir. Geleneksel işlemcilerle YSA'ların paralel veri işleme özelliği gerçeklenememektedir. ASIC mimarilerde paralel veri işleme özelliğine sahip YSA'ların tasarımı maliyetli ve uzun zaman gerektirmektedir. Paralel veri işleme özelliğine sahip FPGA'larla YSA'ların paralel veri işleme gerçeklenebilmektedir. Bu çalışmada kullanılan olan ÇKA tipi YSA mimarisi Şekil 3.13'teki gibi paralel mimariye sahiptir.



Şekil 3.13. Paralel ÇKA tipi YSA Mimarisi

3.3 Geriye Yayılım Algoritmasının Donanımsal Gerçeklenmesi

Bölüm 2.3.1'de anlatılan geriye yayılım algoritmasının paralel mimaride gerçeklenmesi Şekil 3.14'de gösterilmiştir. Şekilden de görüleceği üzere k. katmanda çıkış hücresine ait beklenen değerden çıkış hücresinin o anki iterasyonda elde edilen çıkış değeri çıkarılarak hücreye ait hata değeri hesaplanır. Elde edilen hata değeri ilgili çıkış hücresine ait aktivasyon fonksiyonlarının türevleri ile çarpılır. Çarpım sonuçları ve $k-1$. katmanda elde edilen hücre çıkışları kullanılarak k. katmana ait ağırlık ve eşik değerlerine ait güncelleme değerleri bulunur. Her bir hücrede elde edilen çarpım sonuçları, $k-1$. katmanda bulunan hücrelerin ağırlıkları ile çarpıldıktan sonra toplanır. Elde edilen bu toplam değer k . katmandaki hücrelerde olduğu gibi aktivasyon fonksiyonun türevi ile çarpılır. k . katmana benzer şekilde ağırlık ve eşik değerleri güncelleme işlemleri 1. katmana kadar yapılır. 1. katmanda güncelleme işlemi yapılırken hücre çıkış değerleri yerine ağ giriş değerleri kullanılmaktadır.



Şekil 3.14. Paralel mimariye sahip Geriye Yayılım Algoritması

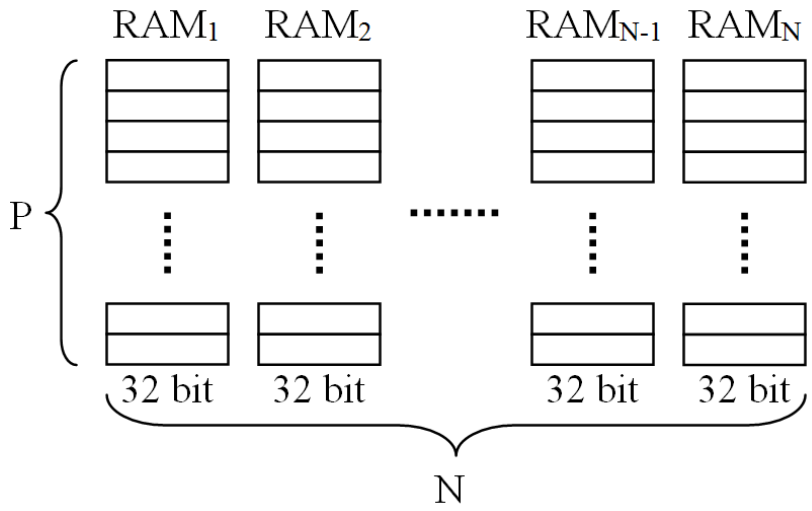
3.4. Levenberg & Marquardt Algoritmasının Donanımsal Gerçeklenmesi

YSA'nın eğitiminin Levenberg& Marquardt Algoritması ile FPGA'da donanımsal gerçeklenmesi Jacobian matrisinin oluşturulması, parametrelerin güncellenmesi aşamalarından meydana gelmektedir.

Jacobian vektörü aşağıdaki (3.12)'deki gibi oluşturulur.

$$J = [\Delta\omega_{11}^{1} \quad \cdots \quad \Delta\omega_{1q}^{1} \quad \cdots \quad \Delta\omega_{m1}^{1} \quad \cdots \quad \Delta\omega_{mq}^{1} \quad \Delta b_{1}^{1} \quad \cdots \quad \Delta b_{q}^{1} \quad \cdots \quad \Delta\omega_{11}^{k} \quad \cdots \quad \Delta b_{n}^{k}] \qquad (3.12)$$

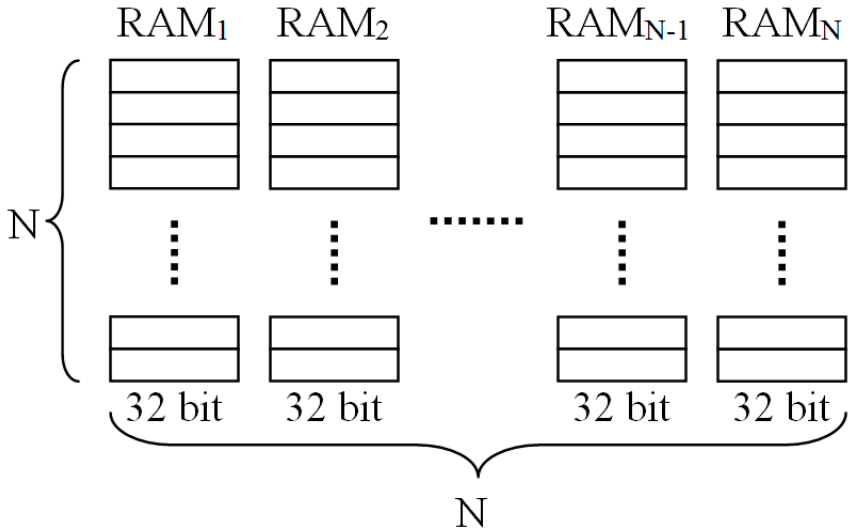
Levenberg & Matquardt algoritması kullanarak ağ eğitimini paralel yapabilmek amacı ile matris sonuçları Jacobian matrisi giriş örnek uzunluğunda (P), ağ parametre sayısı kadar (N) blok ramlara yazılmaktadır. Yani tüm parametrelere ait değerler, kendi vektöründe tutulmaktadır. RAM'larin derinlikleri 32 bit kayan noktalı sayı formatında işlem yapılacağından dolayı 32 bit'tir (Şekil 3.15).



Şekil 3.15. Jacobian matrisi için oluşturulan blok ram yapısı

3.4.1. Devrik Jacobian matrisi ile Jacobian matrisinin çarpımı

Parametrelerin güncellemesi işleminde (2.19)'daki eşitlik adım adım uygulanmaktadır. Öncelikle parametre değerlerinim saklanması için N uzunluğunda, 32 bit derinliğinde N adet blok ram oluşturulmuştur (Şekil 3.16).



Şekil 3.16. Matris çarpımı sonucunu saklamak için oluşturulan blok ram yapısı

Şekil 3.17'dedevrik Jacobian matrisinin, Jacobian matrisi ile çarpımına ait sözde kod verilmiştir.

```
for i = 1 : N
for j = 1 : N
for k = 1 : P

        Jmul(i, j) = Jmul(i, j) + J(k, i) * J(k,j);

end
end
```

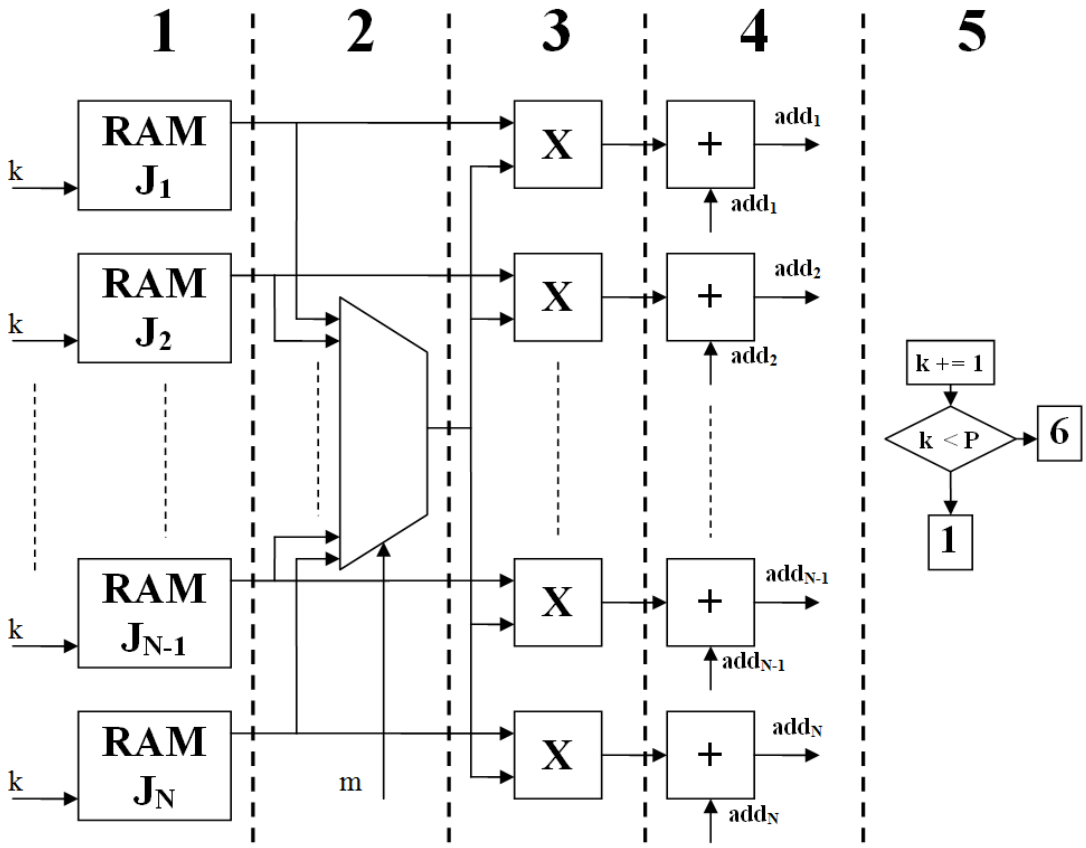
Şekil 3.17. Jacobian matrisinin, Jacobian matrisi ile çarpımına ait sözde kod

Jacobian matrisinde her parametre değerlerinin farklı RAM'de saklanması ile işlem yapılacak satırdaki tüm değerler aynı anda okunabilmektedir ve işlemler aynı anda gerçeklenerek zamandan kazanım sağlanmaktadır (Şekil 3.18). Şekil 3.18'den de görüleceği üzere döngü sayısı N^2P 'den NP 'ye inmiştir.

```
for i = 1 : N
for k = 1 : P

        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);

end
end
```

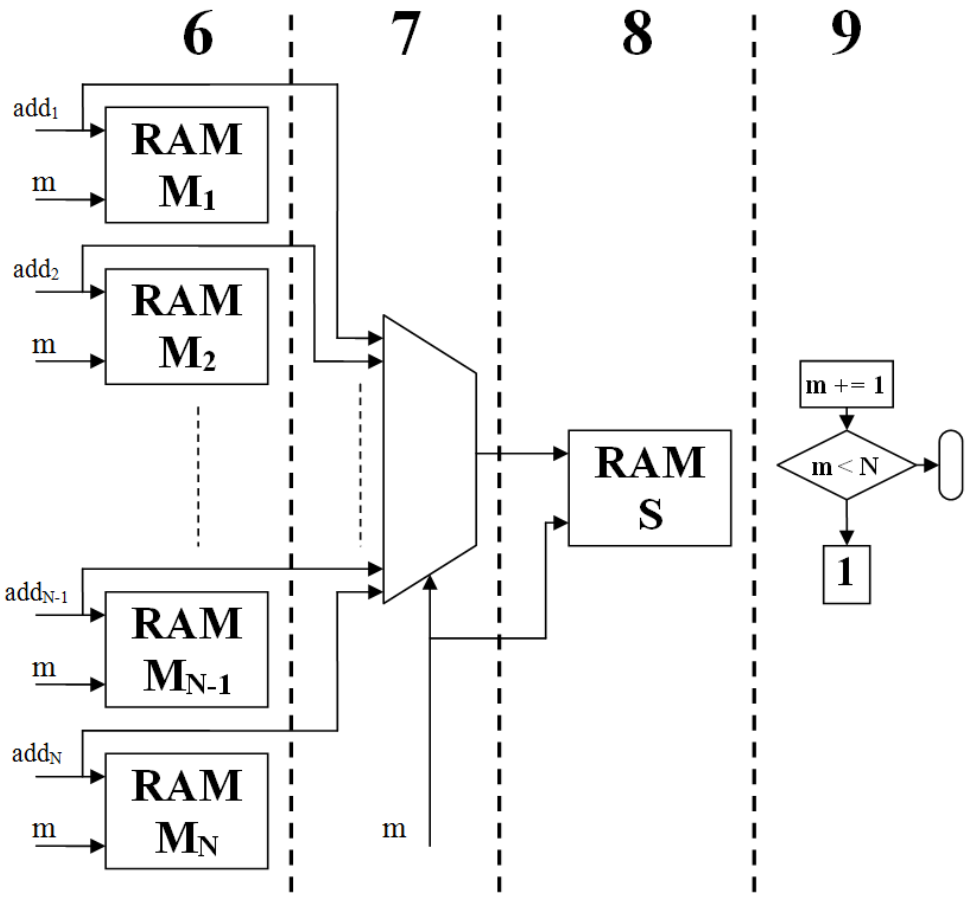
Şekil 3.18. Jacobian matrisinin, Jacobian matrisi ile çarpımına ait parlelleştirilmiş sözde kodu

Donanımsal gerçeklemede birinci adımda ilk olarak, Jacobian matrisinde her parametre için oluşturulan blok RAM'lardan değerler sırasıyla okunmaktadır. 2. adımda bu değerler, arasından çarpma işleminde kullanılacak olan $J(k,m)$ değeri seçilmektedir. 3. adımda seçilen bu değer, 1. adımda okuduğumuz değerlerle çarpılmaktadır. 4. Adımda, 3. adımda elde edilen çarpma değeri bir önceki k değerinde elde edilen toplam değeri ile toplanmaktadır. Bu toplam değeri $k=0$ için 0 olmaktadır. 5. adımda k değeri bir artırılmakta eğer k değeri P değerinden küçükse 1. adıma geçilmekte aksi takdirde 6. adıma geçilmektedir (Şekil 3.19).



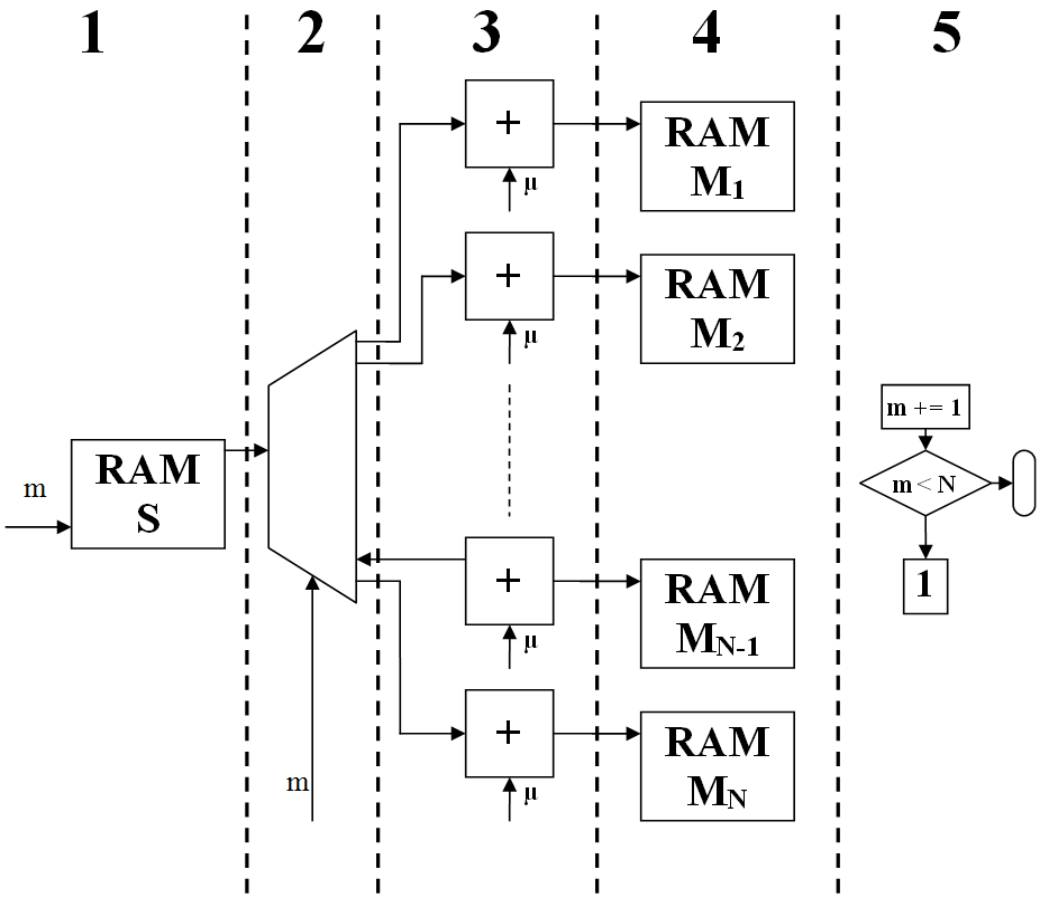
Şekil.3.19. Donanımsak matris çarpımı blok diyagramı-1

6. adımda 4. adımda elde edilen toplam değeri çarpım değerlerini tutmak için oluşturulan RAM'lara yazılmaktadır. μI değerinin matris çarpımına olan etkisi sadece matris köşegenlerine μ değerini eklenmesi ile elde edilebilmektedir. Bu nedenle 7. adımda matrisin (m,m) köşegen değeri bulunmakta, 8. Adımda ise bu değer RAM'a yazılmaktadır. 9. adımda m değeri bir artırılmaktadır. Bu değer N değerinden küçük olursa 1. adıma tekrar dönülür. Aksi takdirde işlem sonlandırılır (Şekil.3.20).



Şekil.3.20. Donanımsak matris çarpımı blok diyagramı-2

Matris çarpımı işleminde RAM'a yazılan köşegen değerleri 1. adımda RAM'dan okunmaktadır. 2. adımda okunan değer, ilgili köşegen değerine yönlendirilir; 3. adımda da bu değer μ parametresi eklenir. 4. aşamada elde edilen sonuçlar m. RAM'in m. adresine yazılmaktadır. 5. adımda ise m değeri bir artırılmaktadır. Bu değer N değerinden küçük olursa 1. adıma tekrar dönülür. Aksi takdirde işlem sonlandırılır (Şekil.3.21).



Şekil 3.21. Matris çarpım değerlerine μ parametresinin eklenmesi

3.4.2 Matris tersi alma

Literatürde matrisin tersini almak için adjoint, LU, QR, Hermitian, Analitik, Blockwise, Gauss-Jordan yok etme yöntemleri önerilmiştir. Bu çalışmada Gauss-Jordan yok etme yöntemi kullanılarak matris boyutu sınırlaması olmadan donanımsal gerçeklenmesi üzerinde çalışılmıştır.

3.4.2.1. Gauss-Jordan yok etme yöntemi

Bu metotta amaç bir matrisin tersini almak için tersi olan matris ile çarpımını, birim matrise dönüştürmektir. A matrisinin tersi (3.13)'deki gibi bulunabilir.

$$A \times B = I \tag{3.13}$$

B matrisi, A matrisinin tersidir ve (3.14)'deki gibi yazılabilir.

$$I = B \times A \tag{3.14}$$

Yönteme ilişkin sözde kod Şekil 3.22'de verilmiştir.

```
b = eye(n,n);

for i = 1 : n
   d = a(i, i);
for j = 1 : n
      a(i, j) = a(i, j) / d;
      b(i, j) = b(i, j) / d;
end
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);
            b(x, j) = b(x, j) - k * b(i, j);
end
end
end
end
```

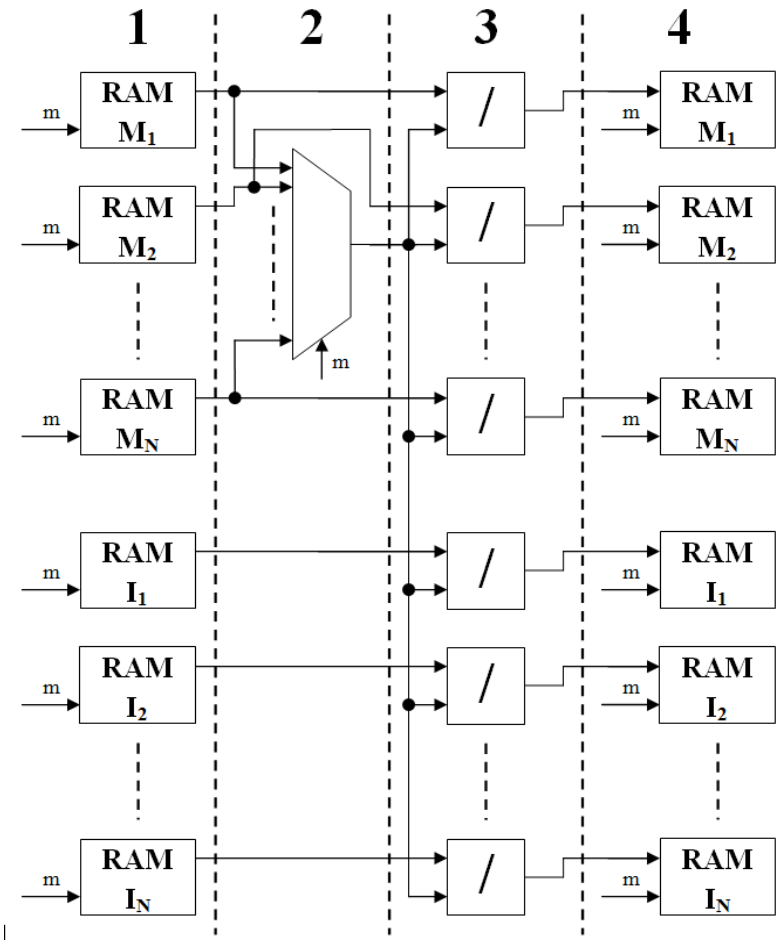
Şekil 3.22. Gauss-Jordan Yok Etme Yöntemi ile matris tersi alma sözde kodu

RAM'larin paralel oluşturulması ile işlemler daha hızlı yapılabilmektedir. Şekil 3.23'te paralel mimari ile Gauss-Jordan Yok Etme Yöntemi ile matris tersi alma sözde kodu verilmiştir.

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

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

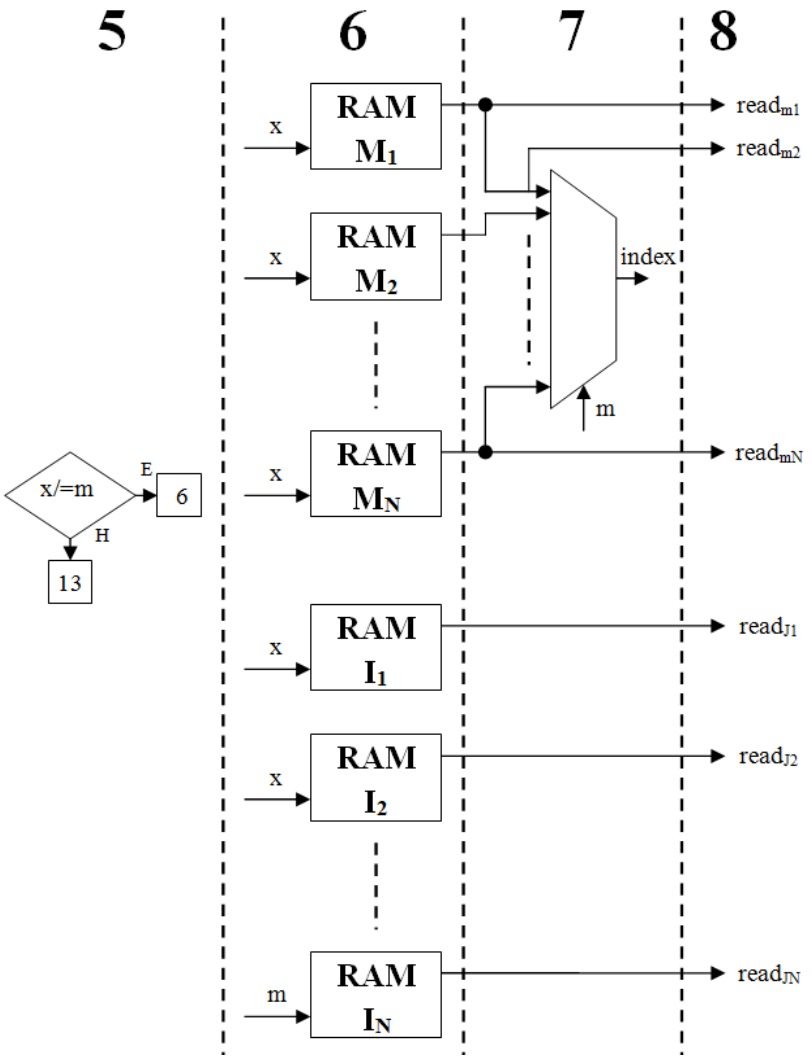
Şekil 3.23. Gauss-Jordan Yok Etme Yöntemi ile paralel mimaride matris tersi alma sözde kodu

Matrisin tersini alma işleminde 1. aşamada ilk olarak bir önceki aşamada elde ettiğimiz ve RAM_M'larda sakladığımız μ parametresi eklenmiş çarpım değerleri ve başlangıçta birim matris olarak oluşturulan RAM_J'lerden *m.* adresteki değerlerler okunmaktadır. 2. aşamada m. RAM_M'deki deki değer seçilmektedir. 3. aşamada, 1. aşamada okunan tüm değerler 2. aşamada elde edilen değere bölünmektedir. Bölme sonuçları 4. aşamada tekrar RAM_M ve RAM_J'lerde *m.* adreslerdeki yerlerine yazılmaktadır (Şekil.3.24).



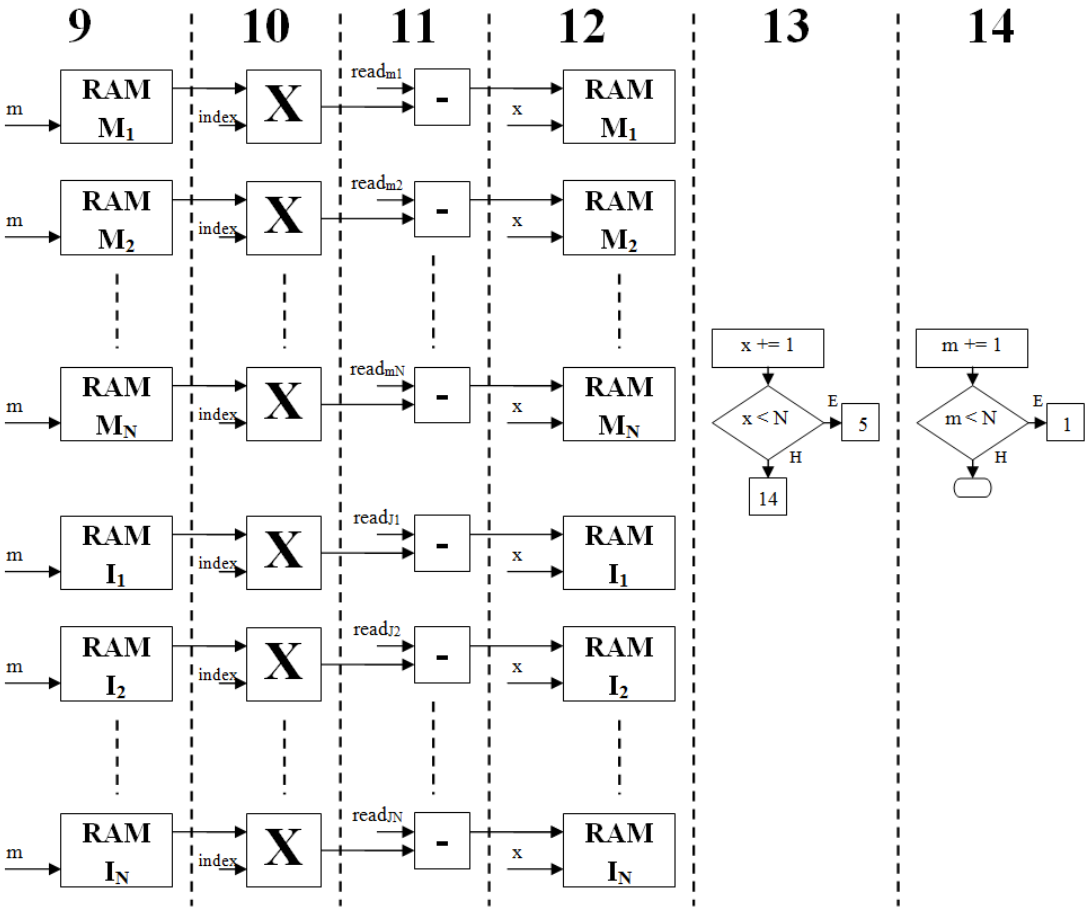
Şekil.3.24. Matrisin tersi alma işlemi-1

5. adımda *x* değerinin *m* değerine eşit olup olmadığı kontrol edilmektedir. Eğer eşit ise *x* değeri 1 artırılmak üzere 13. adıma geçilmektedir. Aksi takdirde 6. adıma geçilmektedir. 6. adımda çarpım değerlerinin ve ters (invers) değerlerinin tutulduğu matris değerleri RAM'lardan *x.* değerler okunmaktadır. 7. adımda ise çarpım değerlerinin tutulduğu RAM'lardan okunan değerlerden *m.* RAM'a ait değer indeks değeri olarak seçilmektedir. 8. adımda ise RAM'lardan okunan tüm değerler hafızada tutulmaktadır (Şekil.3.25).



Şekil.3.25. Matrisin tersi alma işlemi-2

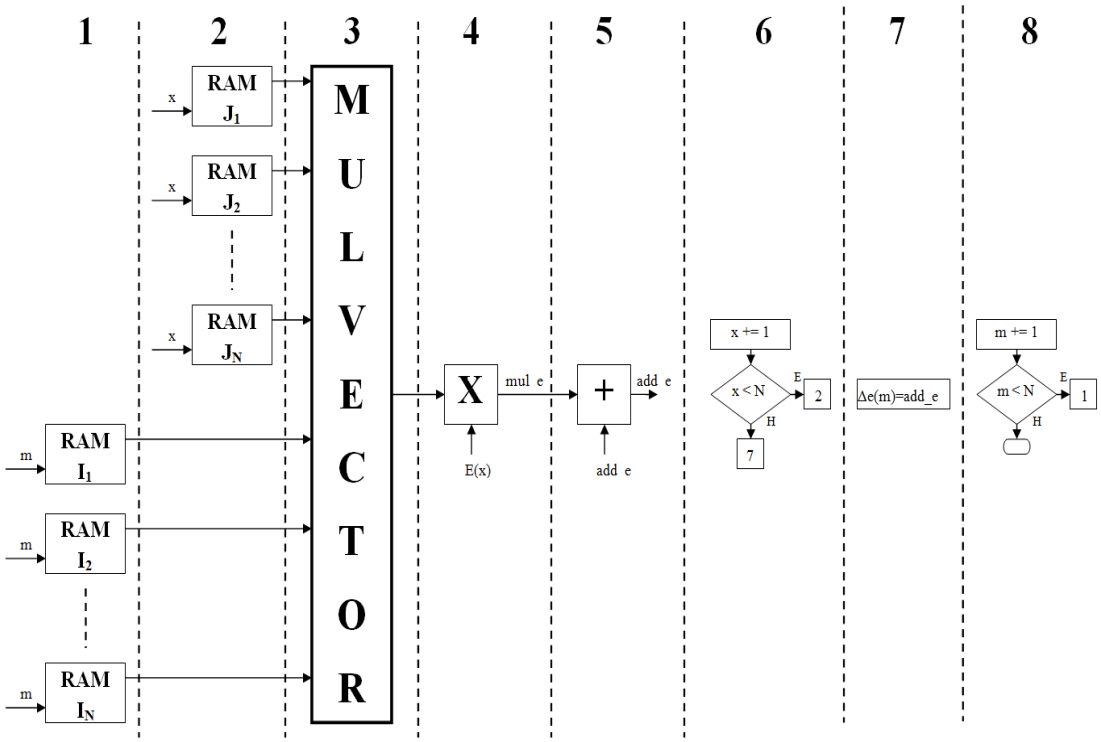
9. adımda ise çarpım değerlerinin ve tersinin değerlerinin tutulduğu matris değerleri RAM'lardan *x.* değerler okunmaktadır. 10. Adımda okunan bu değerler indeks değeri ile çarpılmaktadır. 11. adımda, 8. adımda hafızada sakladığımız bu değerlerden 10 adımda elde edilen çarpma değerleri çıkartılmaktadır. 12. adımda ise çıkarma işlemi sonucu RAM M ve J'lerdeki *x.* yerlerine yazılmaktadır. 13. adımda *x* değeri 1 artırılmaktadır. *x* değeri parametre sayısından az ise 5. adıma dönülür aksi takdirde 14. adıma geçilir. 14. Adımda ise *m.* değeri 1 artırılır. Eğer bu değer parametre sayısından küçük ise 1. Adıma dönülür aksi takdirde işlem sonlandırılır (Şekil.3.26).



Şekil.3.26. Matrisin tersi alma işlemi-3

3.4.3 Matris tersi ile devrik Jacobian matrisinin ve hata matrisinin çarpımı

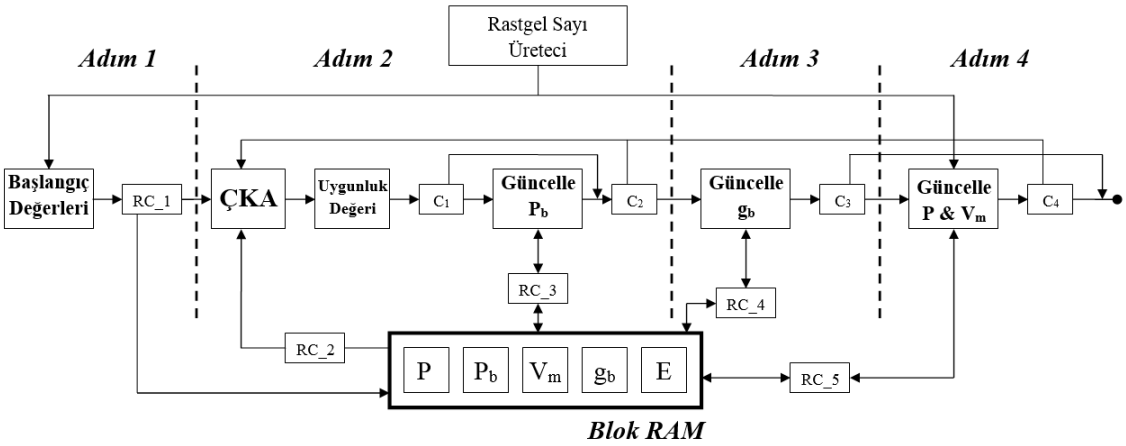
1. adımda ters matris değerlerinin tutulduğu RAM'lardan *m.* değerler okunmaktadır. 2. aşamada ise Jacobian matris parametrelerinin tutulduğu RAM'lardan *x.* değerler okunmaktadır. 3. aşamada 1. ve 2. aşamada okunan değerler 3. Aşamada vektör çarpımı işlemine tabi tutulmaktadır. 4. aşamada ise vektör çarpımının sonucu YSA'da *x.* giriş değerleri için elde edilen hata değeri ile çarpılmaktadır. 5. adımda çarpım sonucu bir önceki adımda elde edilen çarpım sonuç değerlerine eklenir. 6. adımda *x* değeri 1 artırılmaktadır. *x* değeri örnek sayısından az ise 2. adıma donudur aksi takdirde 7. adıma geçilir. 7. Adımda tüm giriş örnekleri için hata ile matris çarpım sonuçlarının toplamı m. parametreye ait güncelleme değeri olarak saklanmaktadır. 8. adımda m. değeri 1 artırılır. Eğer bu değer parametre sayısından küçük ise 1. Adıma dönülür aksi takdirde işlem sonlandırılır (Şekil.3.27).



Şekil.3.27. Matris tersi ile devrik Jacobian matris çarpımı ile hata vektörünün çarpımı

3.5. Parçacık Sürü Optimizasyon Algoritmasının Donanımsal Gerçeklenmesi

FPGA üzerinde Parçacık Sürü Optimizasyon algoritması tabanlı YSA eğitimi blok eğitim Şekil.3.28'da verilmiştir. Şekilden de görüleceği üzere PSO algoritması ile YSA eğitimi, 4 aşamada gerçeklenmektedir.



Şekil 3.28. PSO ile FPGA üzerinde YSA eğitimi blok yapısı

3.5.1. Başlangıç değerlerinin atanması

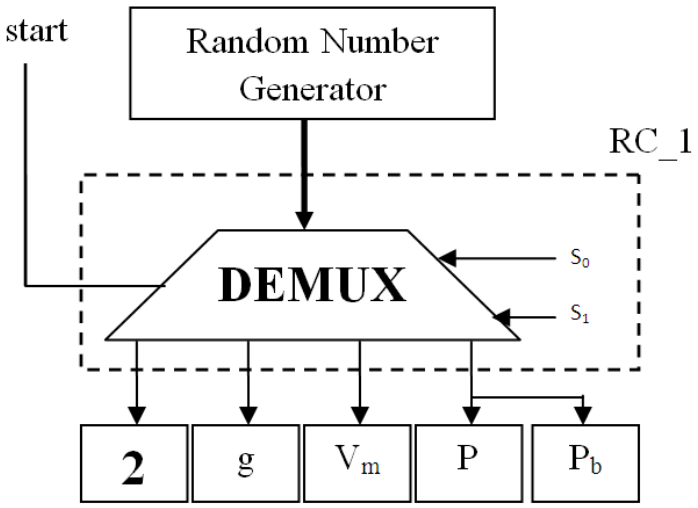
FPGA tabanlı gerçeklemede sürü (P), yerel en iyi parçacıklar (P_b), küresel en iyi parçacıklar (g_b), parçacık hızları (V_m) ve uygunluk değerleri (E_n) blok ramlarda saklanmaktadır. Uygulamada P, P_b, V_m matrisleri için $N \times D$ uzunluğunda, g_b vektörü için $1 \times D$, E_n vektörü için $1 \times N$ uzunluğunda blok RAM'lar oluşturulmuştur. Blok RAM'larin derinlikleri kullanılacak olan sayı formatının bit uzunluğu ile aynıdır. N parçacık sayısını (sürü büyüklüğü) ve D optimize edilecek parametre sayısını gösterir. Her bir vektör ve matrise ait farklı blok RAM oluşturulmasıyla parametrelerin okuma yazma işlemleri paralel yapılabilmektedir (Çavuşlu, Karakuzu ve Karakaya 2012).

PSO'daki parçacık ve hızlarının başlangıç değerleri 32 bit kayan noktalı sayı formatında [0-1) aralığında (3.15) ile verilen denklemle belirlenmiştir (Brysbaert 1991).

$$X_{n+1} = (aX_n + b) \bmod c \tag{3.15}$$

Sayı üretimi X_0 tohum değeri ile başlar. Eşitlikte, a ve b sabit sayılardır. Son terim ise c sayısına göre mod alma işlemini gösterir. Çalışmada rastgele sayı üreteci, programın yüklenmesi ile sayı üretmeye başlamaktadır. Her eğitimde başlangıç parametrelerinin farklı olması için, blok ramlara sayı yazma işlemi kullanıcı tarafından harici bir girişin set edilmesi ile başlamaktadır (Şekil.3.29).

Şekil.3.29'da da görüleceği üzere başlangıç değerlerinin rastgele üretilip RAM'a yazılması işlemi, kontrol girişlerine (s_0, s_1) bağlı olarak yapılmaktadır. Sırasıyla üretilen değerler ilk olarak P ve P_b değerlerine (bu değerler başlangıçta aynı olmalıdır), daha sonra sırasıyla V_m ve g_b için oluşturulan bloklara yazılmaktadır. Yazma işlemi bittikten sonra 2. aşamaya geçilmektedir.



Şekil 3.29. Başlangıç değerlerinin RAM'e yazılmasını gösteren blok şema (Çavuşlu, Karakuzu ve Karakaya 2012.)

3.5.2 YSA ve yerel en iyi parçacıkların belirlenmesi

Bu aşamada yapılan işlemler kısaca şöyledir: Parçacık elemanları RAM'dan okunduktan sonra YSA parametreleri olarak atanır ve bu parametrelerle her giriş için YSA çıkışındaki hata bulunur. Bu hatalar kullanılarak her iterasyonda parçacığın uygunluk değeri hesaplanır. Her parçacık için elde edilen uygunluk değerleri kendine ait eski değerden küçük olması durumunda ilgili parçacık yerel en iyi olarak atanır. Bu işlemler tüm parçacıklar için yapılarak 3. aşamaya geçilir.

Bu aşamada kullanılan en önemli blok ÇKA'dır. Bu blok YSA'nın donanımsal gerçeklenmesini gösterir. Şekil.3.28'de RC_2 işlem bloğu, *i*. parçacığa ait vektörü RAM'dan okur. YSA'nın donanımsal gerçeklenmesine ait detaylı bilgiler Bölüm 3.1 ve Bölüm 3.2'de verilmiştir.

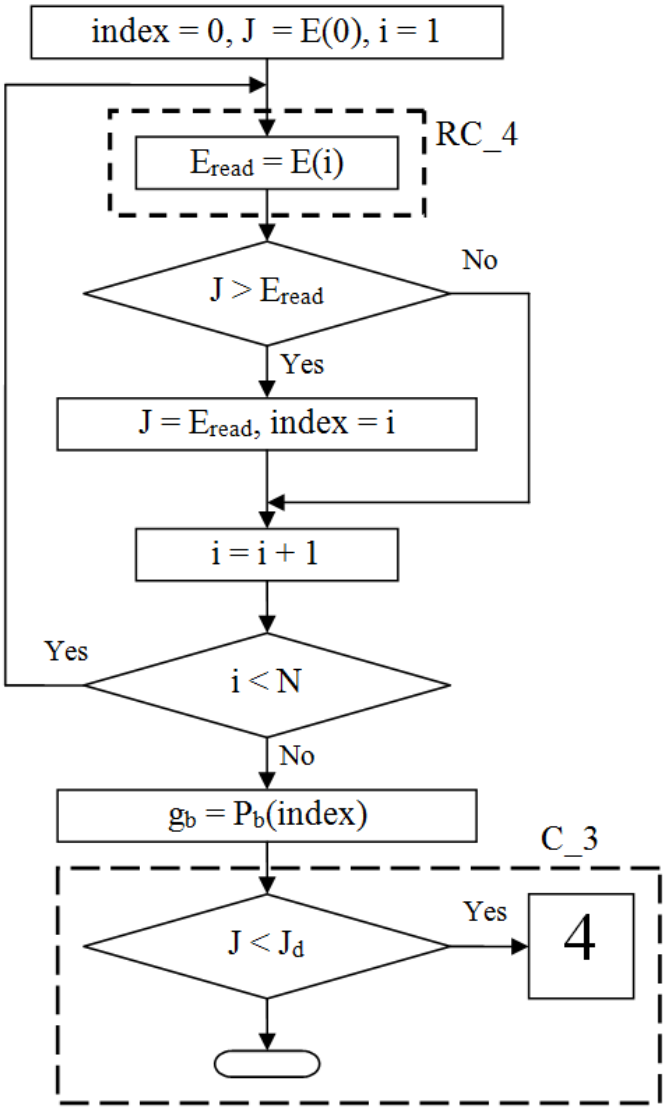
E_n uygunluk değeri hesaplama işlemi Şekil.3.30'da gösterilen "uygunluk değeri" adlı blokta yapılır. Şekil.3.30'da C_1 işlem bloğu elde edilen E_n değerini, daha önceki elde edilen E_{n_old} değeri ile karşılaştırır. Eğer yeni değer eski değerden küçük ise o parçacık P_b 'de ilgili yerine yazılır ve E_n değeri de E_{n_old} değerine atanır. Aksi halde C_2 bloğuna geçiş yapılır. C_2 bloğu ise tüm parçacıklar için aynı işlemlerin yapılıp yapılmadığını kontrol eder. Şekil.3.28'de RC_3 bloğu, eğer güncelleme işlemi

yapılacaksa sürüdeki $i.$ parçacık değerlerini ramdan okuyarak, yerel en iyideki yerine yazar. Aynı zamanda o parçacığa ait uygunluk değerini de günceller.

3.5.3. Küresel en iyi parçacıkların belirlenmesi

Tüm parçacıklar için uygunluk değeri hesaplanıp yerel en iyiler P_b içinde güncellendikten sonra küresel en iyi parçacık belirlenir. E_n içinde tutulan uygunluk değerleri arasından minimum olanının indeksinin işaret ettiği P_b'in ilgili satır vektörü küresel en iyi (g_b) olarak atanır. Bu işlem Şekil.3.30'de blok olarak verilmiştir. Şekildeki C_3 işlem bloğu eğer minimum ölçüt değeri sonlandırma kriteri (J_d) değerinin altında ise eğitimi sonlandır aksi takdirde 4. aşamaya geçilir. RC_4 işlem bloğu öncelikle uygunluk değerlerinin okunması işlemini yapar. Daha sonra sürüdeki en iyi pozisyona sahip parçacığa ait yerel en iyi değerlerini okuyarak, küresel en iyi değerine yazma işlemini yapar (Çavuşlu, Karakuzu ve Karakaya 2012.)

.

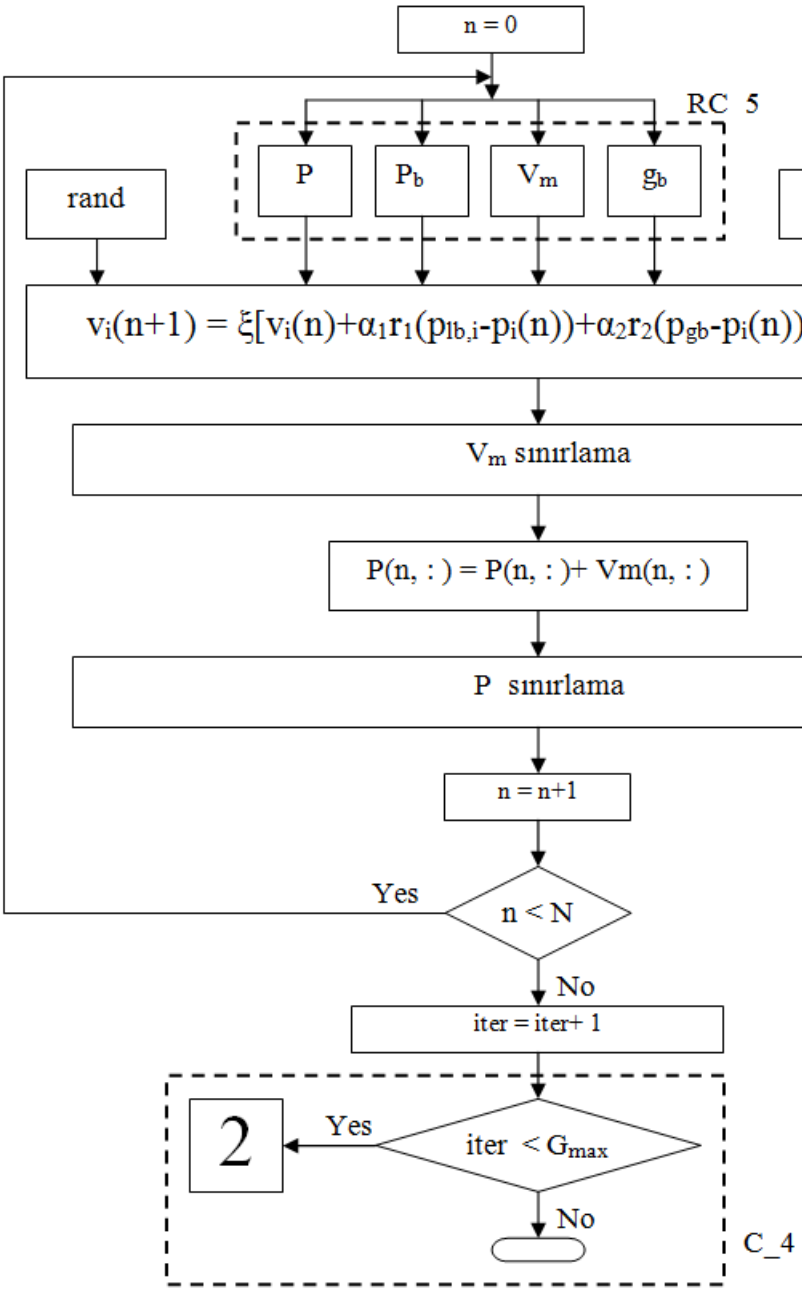


Şekil 3.30. Küresel en iyi parçacığı bulma akış diyagramı

3.5.4 Güncelleme

Bu aşamada yapılan işlemlerin akış şeması detaylı olarak Şekil 3.31'de verilmiştir. Şemadan da görüleceği üzere, RAM'den aynı anda alınan P, P_b, V_m ve g_b güncelleme işlemine tabi tutulur. P ve V_m için güncellemeler yapıldıktan sonra, güncel parçacıklar ve hızları aynı anda RAM'e yazılmaktadır. Tüm parçacıklara ait parametrelerin güncelleme işlemi yapıldıktan sonra iterasyon sayısı bir artırılır ve 2 nolu işlem bloğuna geri dönülür. Eğer iterasyon sayısı maksimum jenerasyon sayısı ($G_{\max}$) değerinden fazla olursa işlem sonlandırılır. Aksi takdirde 2 nolu aşamaya geçiş yapılır (Çavuşlu, Karakuzu ve Karakaya 2012.)

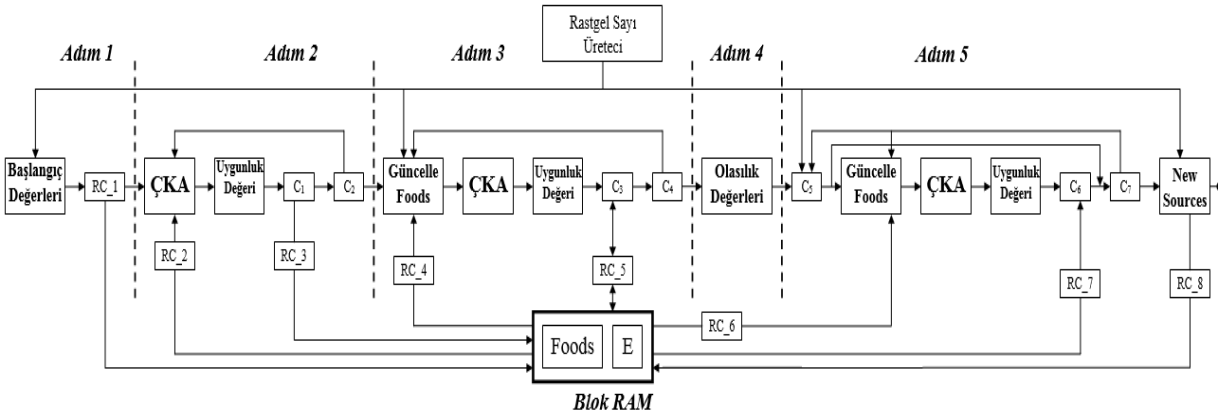
.



Şekil 3.31. Güncelleme akış diyagramı

3.6. Yapay Arı Koloni Algoritmasının Donanımsal Gerçeklenmesi

FPGA üzerinde Yapay Arı Koloni algoritması tabanlı YSA eğitimi blok eğitim Şekil.3.32'de verilmiştir. Şekilden de görüleceği üzere YAK algoritması ile YSA eğitimi, 5 aşamada gerçeklenmektedir.



Şekil 3.32. YAK ile FPGA üzerinde YSA eğitimi blok yapısı

3.6.1. Başlangıç değerlerinin atanması

FPGA tabanlı gerçeklemede yiyecek kaynakları ($Foods$) ve uygunluk değerleri (E_n) blok ramlarda saklanmaktadır. Uygulamada $Foods$ matrisleri için $N \times D$ uzunluğunda, E_n vektörü için $1 \times N$ uzunluğunda blok RAM'lar oluşturulmuştur. Blok RAM'larin derinlikleri kullanılacak olan sayı formatının bit uzunluğu ile aynıdır. N yiyecek kaynakları sayısını ve D optimize edilecek parametre sayısını gösterir. Her bir vektör ve matrise ait farklı blok RAM oluşturulmasıyla parametrelerin okuma yazma işlemleri paralel yapılabilmektedir.

PSO algoritmasında olduğu gibi yiyecekler başlangıç değerleri 32 bit kayan noktalı sayı formatında [0-1) aralığında (3.15) ile verilen denklemle belirlenmiştir.

Şekil.3.32'den de görüleceği üzere başlangıç değerlerinin rastgele üretilip RAM'a yazılması işlemi, RC_1 işlem bloğunda yapılmaktadır. Bu blokta, kaynak sayısı ile parametre sayısı kadar üretilen rastgele sayıların ram bloğuna yazılması işlemi yapılmaktadır. Yazma işlemi bittikten sonra ise 2. aşamaya geçilmektedir.

3.6.2 Başlangıç uygunluk fonksiyonu değerlerinin elde edilmesi

Bu aşamada kullanılan en önemli blok ÇKA'dır. Bu blok YSA'nın donanımsal gerçeklenmesini gösterir. Şekil.3.32'de RC_2 işlem bloğu, i. kaynağa ait vektörü RAM'dan okur. Okunana bu değerler YSA parametreleri olarak atanmaktadır. YSA'nın donanımsal gerçeklenmesine ait detaylı bilgiler Bölüm 3.1 ve Bölüm 3.2'de verilmiştir.

Tüm örnek giriş değerleri için E_n uygunluk değeri hesaplama işlemi Şekil.3.34'de gösterilen "uygunluk değeri" adlı blokta yapılır. Şekil.3.34'de C_1 işlem bloğu elde edilen E_n değerinin blok rama yazılması için RC_3 işlem bloğuna gönderir. RC_3 işlem bloğunda E_n değerini ilgili blok rama yazılma işlemi gerçekleştirilir ve C_2 işlem bloğuna geçiş yapılır. C_2 bloğu ise tüm kaynaklar için aynı işlemlerin yapılıp yapılmadığını kontrol eder.

3.5.3. İşçi arı yiyecek kaynaklarını güncellenmesi

Bu aşamada yiyecek kaynaklarından $i.$ kaynağa ait parametreler ve farklı bir kaynağın rastgele seçilmiş r_s indeksli parametresi RC_4 işlem bloğu ile blok ramdan okunur. $i.$ kaynağa ait r_s indeksli parametre, diğer kaynağa ait parametre yardımıyla (2.26) referans alınarak "Güncelle foods" işlem bloğunda güncellenir. Elde edilen yeni güncel $i.$ kaynak parametreleri YSA parametreleri olarak atanır. Tüm örnek giriş değerleri için E_n uygunluk değeri hesaplandıktan sonra C_3 işlem bloğu ile $i.$ kaynağa ait hafızadaki uygunluk değeri RC_5 işlem bloğu ile okunur. Yeni uygunluk değeri ile hafızadan okunan uygunluk değeri karşılaştırılır. Eğer yeni uygunluk değeri hafızadan okunan değerden küçükse yeni uygunluk değeri ve güncellenmiş r_s parametresi RC_5 işlem bloğuyla tekrar hafızaya yazılır. Aksi takdirde $i.$ kaynağa ait sınama değeri bir artırılır ve C_4 işlem bloğuna geçilir. C_4 bloğu ise tüm kaynaklar için aynı işlemlerin yapılıp yapılmadığını kontrol eder.

3.6.4 Olasılıksal değerlerin hesaplanması

Bu aşamada, en büyük uygunluk değeri tespit edilir. Daha sonra tüm kaynaklara ait tüm uygunluk değerleri en büyük uygunluk değerine bölünerek olasılıksal değerler elde edilir.

3.6.5 İşçi arı yiyecek kaynaklarını güncellenmesi

Bu aşamada C_5 işlem bloğunda $i.$ kaynağa ait olasılık değeri, o an ki rastgele üretilen sayı değeri ile karşılaştırılır. Eğer olasılık değeri rast gele üretilen sayıdan küçükse C_7 işlem bloğunda geçilir. Aksi takdirde parametre güncelleme için "Güncelle Foods"

işlem bloğuna geçilir. Bu işlem bloğunda yiyecek kaynaklarından $i.$ kaynağa ait parametreler ve farklı bir kaynağın rastgele seçilmiş r_s indeksli parametresi RC_6 işlem bloğu ile blok ramdan okunur. $i.$ kaynağa ait r_s indeksli parametre, diğer kaynağa ait parametre yardımıyla (2.26) referans alınarak güncellenir. Elde edilen yeni güncel $i.$ kaynak parametreleri YSA parametreleri olarak atanır. Tüm örnek giriş değerleri için E_n uygunluk değeri hesaplandıktan sonra C_6 işlem bloğu ile $i.$ kaynağa ait hafızadaki uygunluk değeri RC_7 işlem bloğu ile okunur. Yeni uygunluk değeri ile hafızadan okunan uygunluk değeri karşılaştırılır. Eğer yeni uygunluk değeri hafızadan okunan değerden küçükse yeni uygunluk değeri ve güncellenmiş r_s parametresi RC_5 işlem bloğuyla tekrar hafızaya yazılır. Aksi takdirde $i.$ kaynağa ait sınama değeri bir artırılır ve C7 işlem bloğuna geçilir. C_7 bloğu ise toplam işlem sayısının kaynak sayısından büyük olma durumunu kontrol eder. Büyük olması durumunda "New Source" işlem bloğuna geçilir. Aksi takdirde C_5 işlem bloğuna geçiş yapılır. New Source işlem bloğunda ise sınama değerleri belirlenen limitin üzerinde olan kaynaklar için yeni parametre değerleri oluşturulur. RC_8 işlem bloğu ile bu parametreler ramdaki yerlerine yazılır.

BÖLÜM IV

ÇKA ve EĞİTİM ALGORİTMALARININ FPGA ÜZERİNDE DONANIMSAL GERÇEKLENMESİNE İLİŞKİN DENEYSEL SONUÇLAR

Bölüm III'te anlatılan GY, LM, PSO ve YAK ile YSA eğitiminin FPGA üzerinde gerçeklenmesi iki örnek sistem tanıma ve plaka tanıma problemi üzerinde deneysel olarak test edilmiştir. Deneysel çalışmalar, ISE Design Suite 14.6 programı kullanılarak Xilinx Kintex 7 xc7k325tffg900-2 FPGA'sı üzerinde gerçeklenmiştir.

Örnek 1:Dinamik sistem tanıma problemi sistemi temsil edecek olan parametrelerin uygun bir şekilde ayarlanması işlemidir. Bu ayarlama işlemi, tanınacak sistemin gerçek çıkışı ile tanıma için seçilen modelin çıkışı arasındaki hata üzerine kurulmuş başarım işlevinin en iyilemesi (optimizasyonu) ile gerçeklenir. Bu örnek için Denklem (4.1)'de verilen sistem tanımlaması yapılmıştır (Narendra ve Parthasarathy 1990).

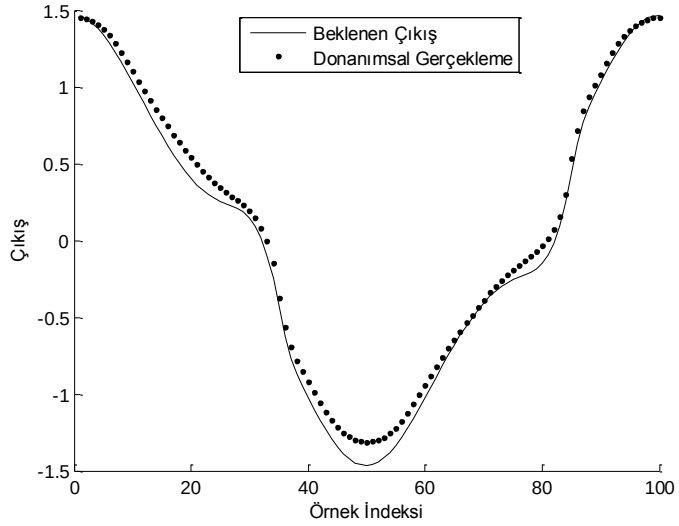
$$y[k+1]=\frac{y[k]}{1+y^2[k]}+u^3[k] \tag{4.1}$$

Denklem (4.1)'de $u[k]$ Denklem (4.2)'deki gibi tanımlanmıştır.

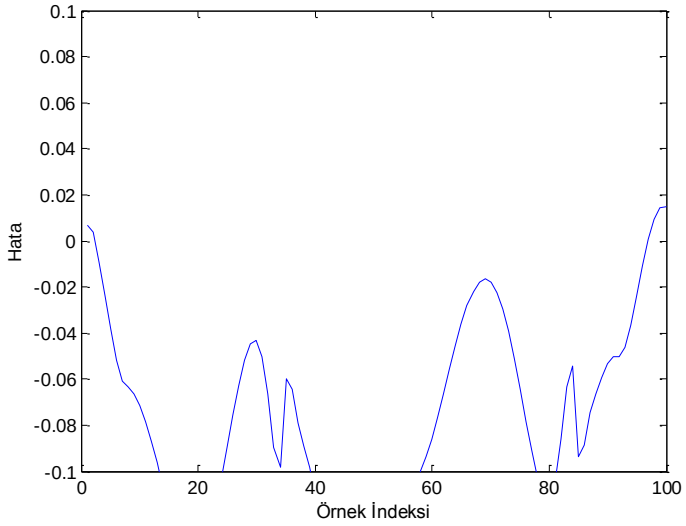
$$u[k]=\cos\frac{2\pi k}{100}, \quad k=1,2,\cdots,100 \tag{4.2}$$

Sistem tanıma için girişleri $u[k]$ ve $y[k]$ olan, gizli katmanında 3 ve çıkış katmanında 1 hücresi olan ÇKA kullanılmıştır. Gizli katman hücreleri için aktivasyon fonksiyonu olarak logaritmik sigmoidal fonksiyonu, çıkış hücresi için lineer aktivasyon fonksiyonu kullanılmıştır.

Şekil 4.1a'da Örnek 1 için geriye yayılım algoritması ile dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.1b'de Örnek 1 için geriye yayılım algoritması ile dinamik sistem tanımanın donanımsal gerçeklemesine ait hataları göstermektedir.



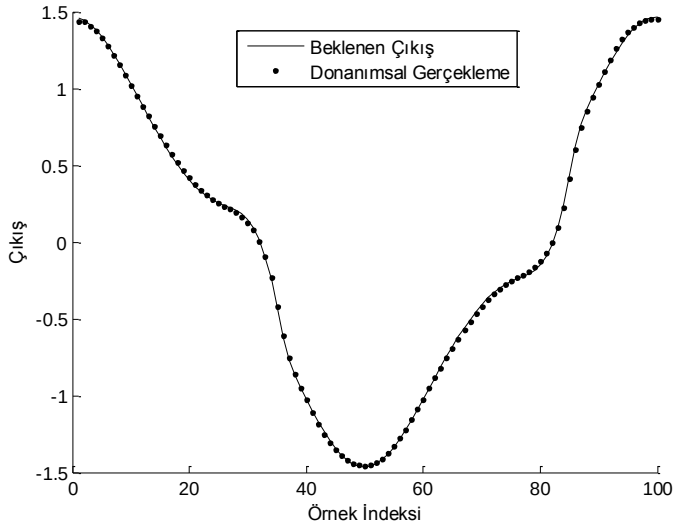
(a)



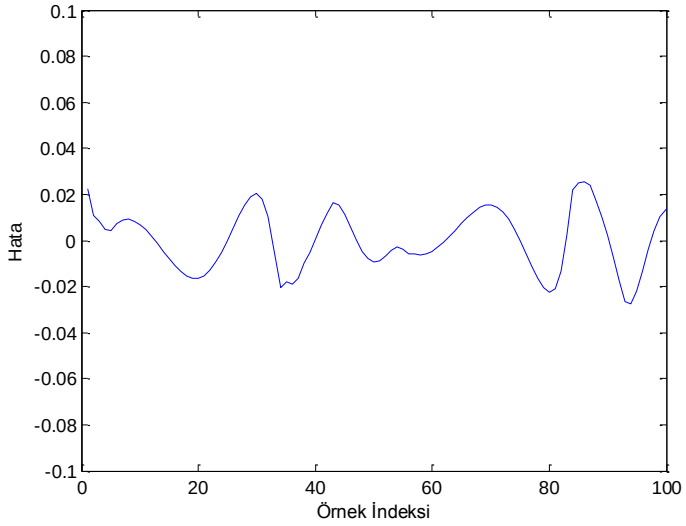
(b)

Şekil 4.1. Örnek 1 için geriye yayılım algoritması kullanılarak eğitilen ağa ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.2a'da Örnek 1 için LM algoritması ile dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.2b'de Örnek 1 için LM algoritması ile dinamik sistem tanımanın donanımsal gerçeklemesine ait hataları göstermektedir.



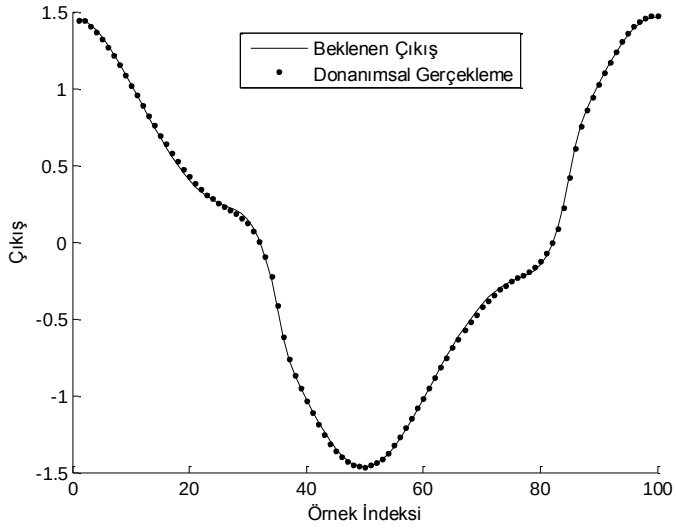
(a)



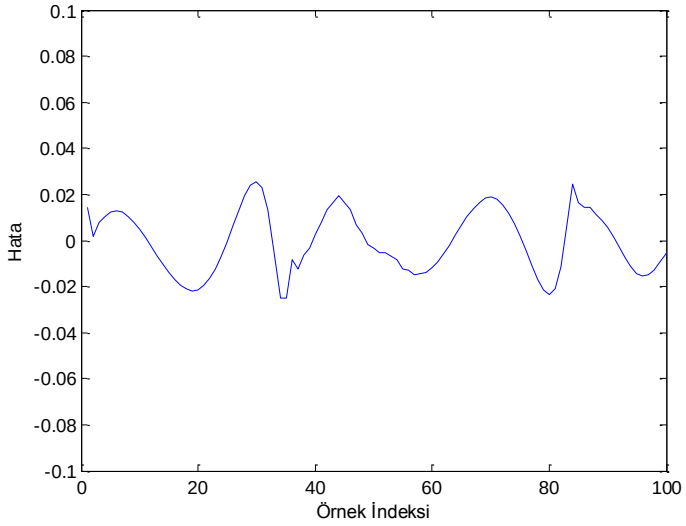
(b)

Şekil 4.2. Örnek 1 için LM algoritması kullanılarak eğitilen ağa ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.3a'da Örnek 1 için PSO algoritması ile dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.3b'de Örnek 1 için PSO algoritması ile dinamik sistem tanımanın donanımsal gerçeklemesine ait hataları göstermektedir.



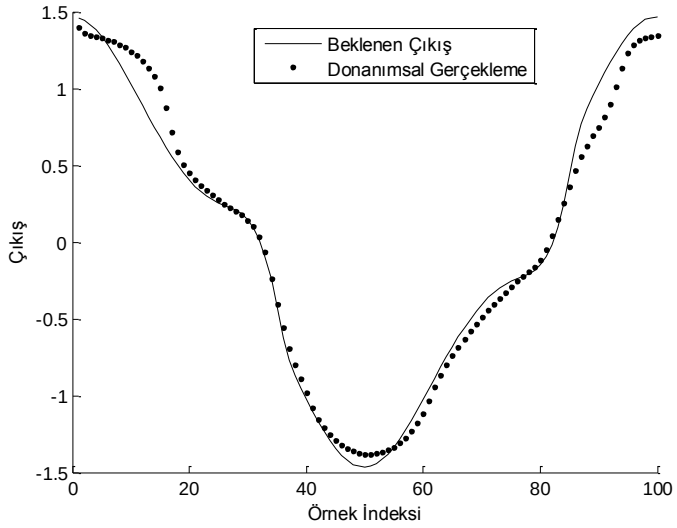
(a)



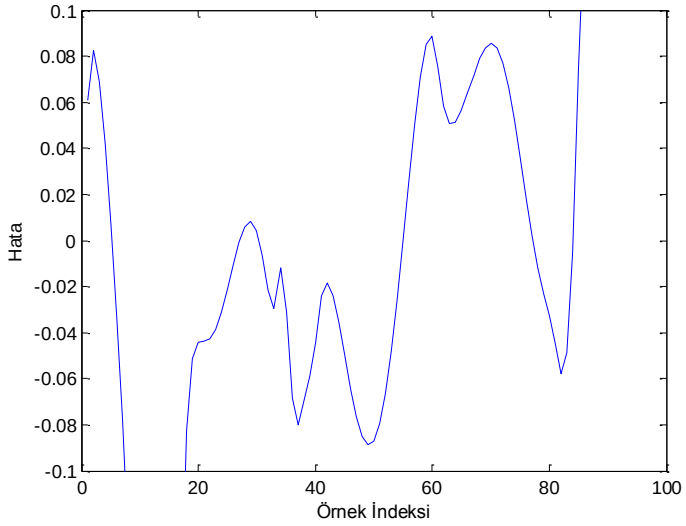
(b)

Şekil 4.3. Örnek 1 için PSO algoritması kullanılarak eğitilen ağa ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.4a'da Örnek 1 için YAK algoritması ile dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.4b'de Örnek 1 için YAK algoritması ile dinamik sistem tanımanın donanımsal gerçeklemesine ait hataları göstermektedir.



(a)



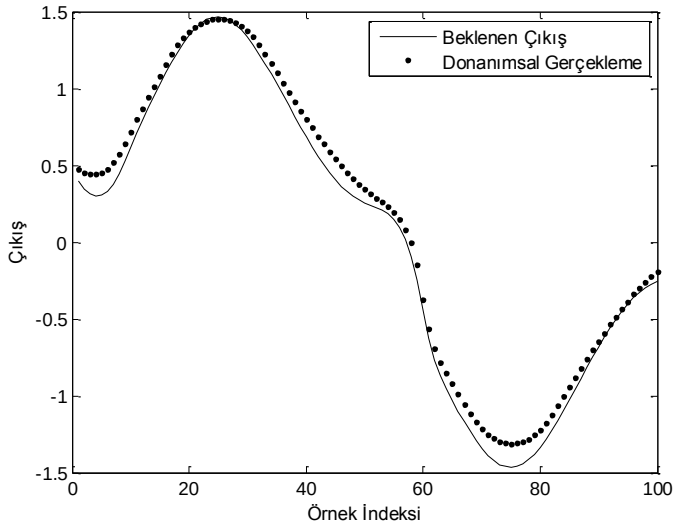
(b)

Şekil 4.4. Örnek 1 için YAK algoritması kullanılarak eğitilen ağa ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

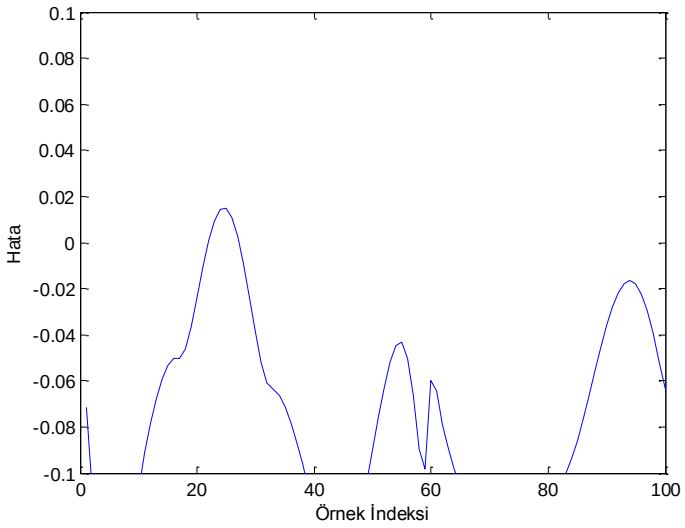
Denklem (4.2)'de verilen giriş dizisine göre eğitilen YSA'nın test işlemleri için Denklem (4.3)'te verilen giriş dizisi kullanılmıştır.

$$u[k] = \sin\frac{2\pi k}{100}, \quad k = 1,2,\cdots,100 \tag{4.3}$$

Şekil 4.5a'da Örnek 1 için Denklem (4.2)'de verilen giriş dizisi ile geriye yayılım algoritması ile eğitilmiş ağın Denklem (4.3)'te verilen giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.5b'de, geriye yayılım algoritması ile eğitilmiş ağın Denklem (4.3)'te verilen giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal gerçekleme hataları gösterilmektedir.



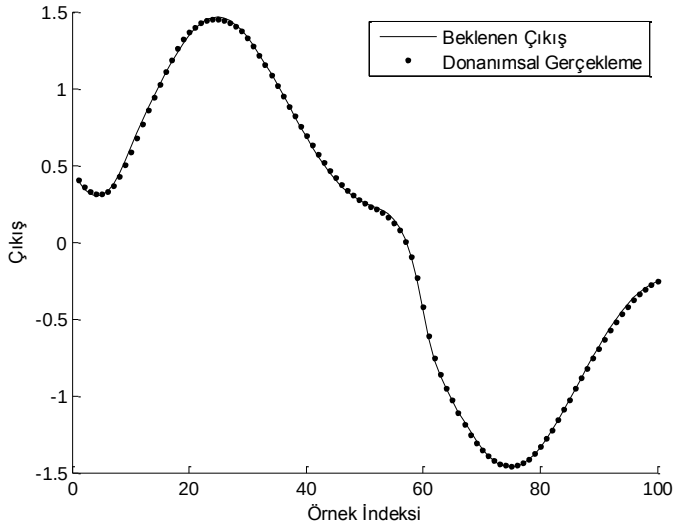
(a)



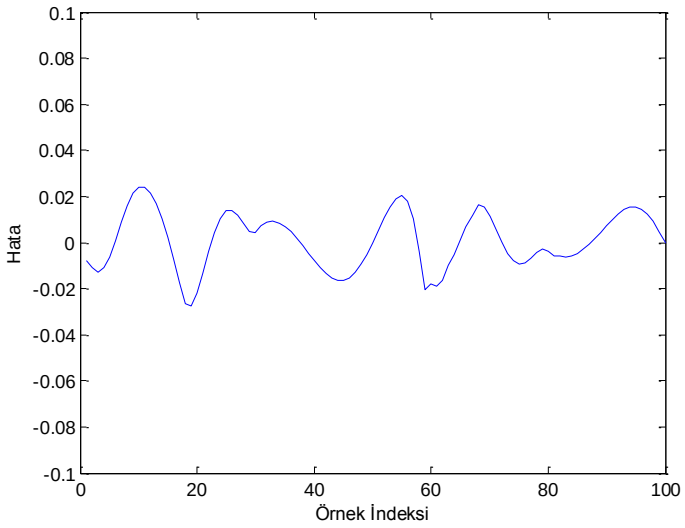
(b)

Şekil 4.5. Örnek 1 için geriye yayılım algoritması kullanılarak eğitilen ağa gösterilmeyen giriş değerlerine ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.6a'da Örnek 1 için Denklem (4.2)'de verilen giriş dizisi ile LM ile eğitilmiş ağın Denklem (4.3)'te verilen giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.6b'de, LM algoritması ile eğitilmiş ağın Denklem (4.3)'te verilen giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal gerçekleme hataları gösterilmektedir.



(a)

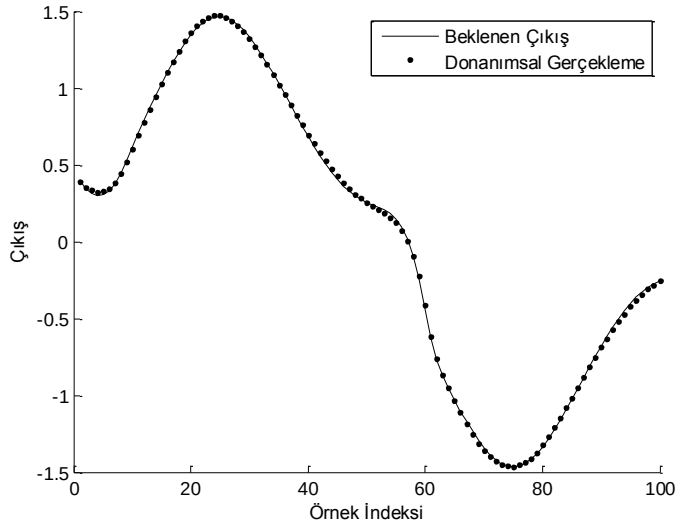


(b)

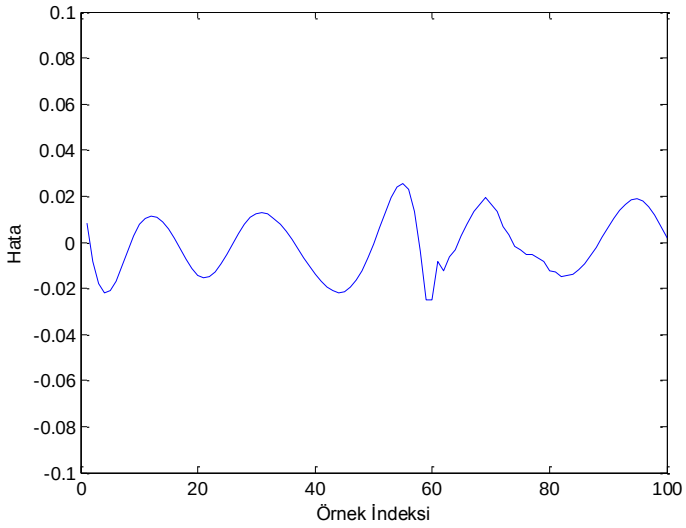
Şekil 4.6. Örnek 1 için LM algoritması kullanılarak eğitilen ağa gösterilmeyen giriş değerlerine ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.7a'da Örnek 1 için Denklem (4.2)'de verilen giriş dizisi ile PSO ile eğitilmiş ağın Denklem (4.3)'te verilen giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal

çıktıları göstermektedir. Şekil 4.7b'de, PSO algoritması ile eğitilmiş ağın Denklem (4.3)'te verilen giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal gerçekleme hataları gösterilmektedir.



(a)

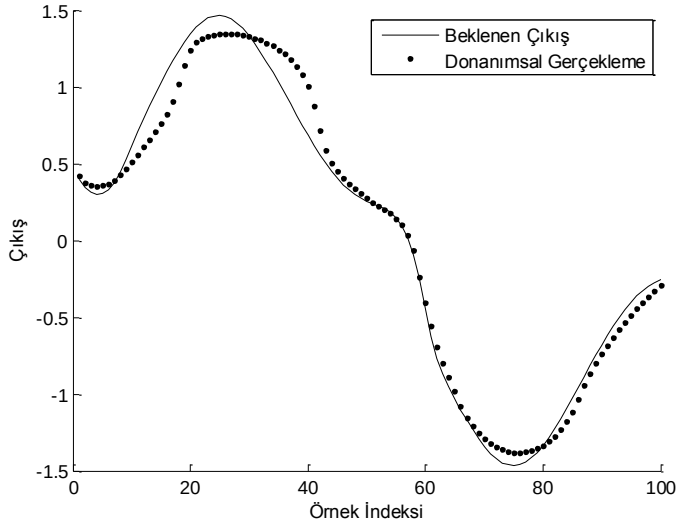


(b)

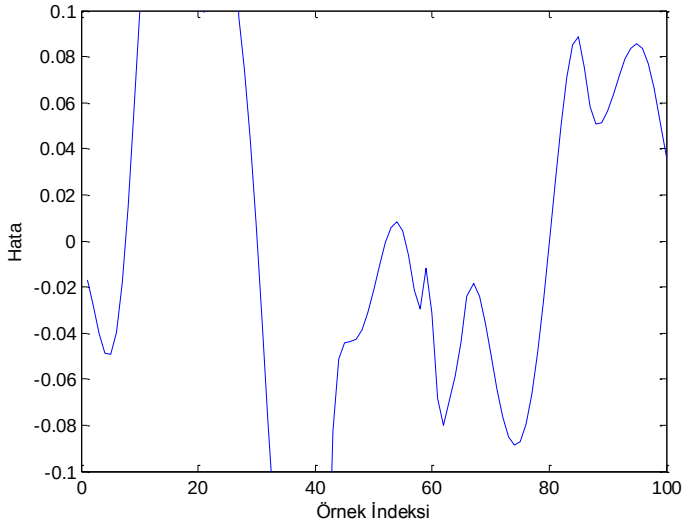
Şekil 4.7. Örnek 1 için PSO algoritması kullanılarak eğitilen ağa gösterilmeyen giriş değerlerine ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.8a'da Örnek 1 için Denklem (4.2)'de verilen giriş dizisi ile YAK ile eğitilmiş ağın Denklem (4.3)'te verilen giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.8b'de, YAK algoritması ile eğitilmiş ağın

Denklem (4.3)'te verilen giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal gerçekleme hataları gösterilmektedir.



(a)



(b)

Şekil 4.8. Örnek 1 için YAK algoritması kullanılarak eğitilen ağa gösterilmeyen giriş değerlerine ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Çizelge 4.1'de ağa gösterilen giriş dizisine göre eğitilen ağın eğitimde kullanılan algoritmalara göre uygunluk değerleri ile ağa gösterilmeyen giriş dizisi için elde edilen uygunluk değerleri verilmiştir. Uygunluk değerleri 10 farklı eğitim sonucunda elde edilen uygunluk değerlerinin ortalaması olarak verilmiştir. Şekil 4.9'da eğitim dataları

üzerinde elde edilen hata değerleri tek grafikte gösterilmiştir. Şekil 4.10'da ise test dataları üzerinde elde edilen hata değerleri tek grafikte gösterilmiştir.

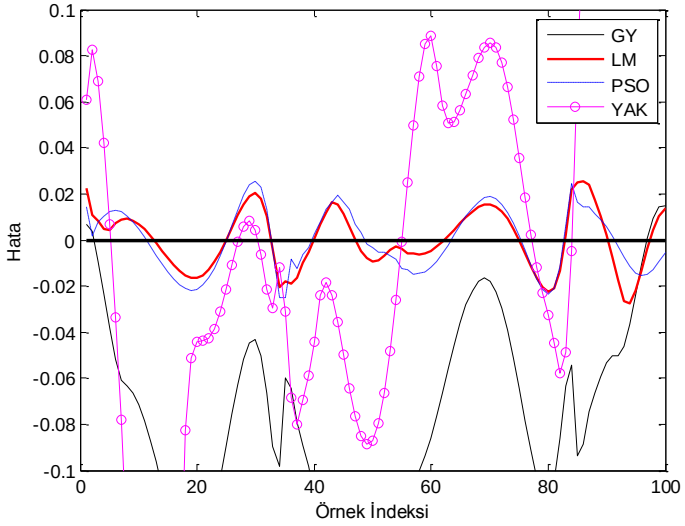
Çizelge 4.2'de GY, LM, PSO ve YAK algoritma ile eğitilmiş YSA'nın FPGA üzerinde gerçeklenmesine ilişkin sentez sonuçları verilmiştir.

Çizelge 4.1. Örnek 1 için eğitim ve test verilerine göre ağ uygunluk değerleri

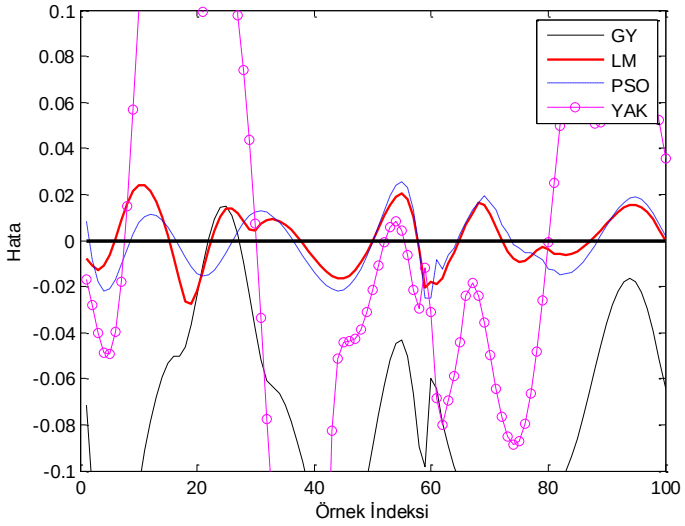
	Eğitim	**Test**	**İterasyon**
GY	0.3830	0.4211	2000
LM	0.0081	0.0073	20
PSO	0.0090	0.0082	2000
YAK	0.7295	0.7086	2000

Çizelge 4.2. Örnek 1 için sentez sonuçları

		Slice Reg.	**Slice LUTs**	**Block RAM**	**DSP48E1s**
	Mevcut	407600	203800	445	840
BP	Kullanılan	3503	21396	2	82
	Yüzde	0	10	0	9
LM	Kullanılan	19314	54858	22	150
	Yüzde	4	26	4	17
PSO	Kullanılan	5587	19701	8	29
	Yüzde	1	9	1	3
YAK	Kullanılan	5639	19786	4	26
	Yüzde	1	9	0	3



Şekil 4.9. Örnek 1 için eğitim datalarında elde edilen hata grafiği



Şekil 4.10. Örnek 1 için test verilerinde elde edilen hata grafiği

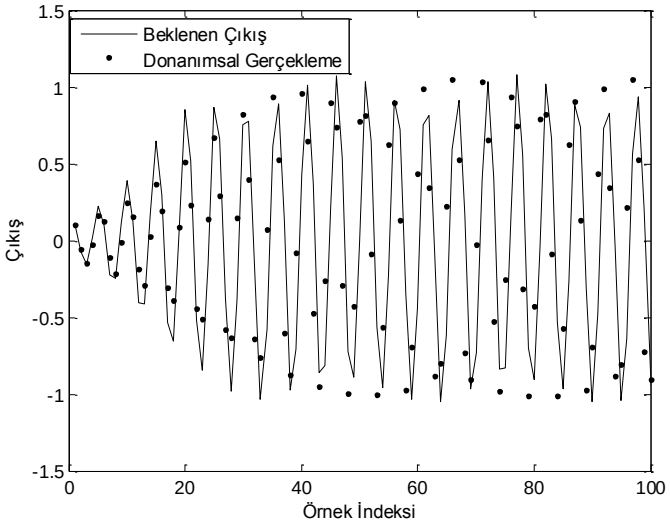
Örnek 2:Bu örnek için (4.4)'de verilen sistem tanımlaması yapılmıştır(Chen ve Billings 1992).

$$\begin{aligned} y[k+1] = {} & -1.17059\,y[k-1] + 0.606861\,y[k] + 0.679190\,y^2[k]\,y[k-1] \\ & - 0.136235\,y^4[k]\,y[k-1] + 0.165646\,y^3[k]\,y[k-1] \\ & \text{-}0.00711966\,y^6[k]\,y[k-1] - 0.114719\,y^5[k]\,y[k-1] \\ & \text{-}0.0314354\,y[k]\,y[k-1] + 0.0134682\,y^3[k] \end{aligned} \tag{4.4}$$

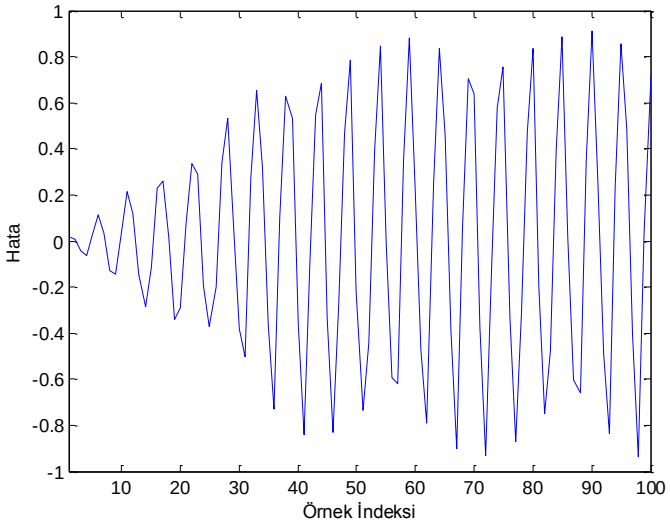
Sistem tanıma için girişleri $y[k]$ ve $y[k-1]$olan, gizli katmanında 3 ve çıkış katmanında 1 hücresi olan ÇKA kullanılmıştır. Gizli katman hücreleri için aktivasyon

fonksiyonu olarak tanjant hiperbolik fonksiyonu, çıkış hücresi için lineer aktivasyon fonksiyonu kullanılmıştır. Giriş değerlerinin elde edilmesinde başlangıç değerleri $y[0] = y[-1] = 0.1$ seçilmiştir.

Şekil 4.11a'da Örnek 2 için geriye yayılım algoritması ile dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.11b'de Örnek 2 için geriye yayılım algoritması ile dinamik sistem tanımanın donanımsal gerçeklemesine ait hataları göstermektedir.



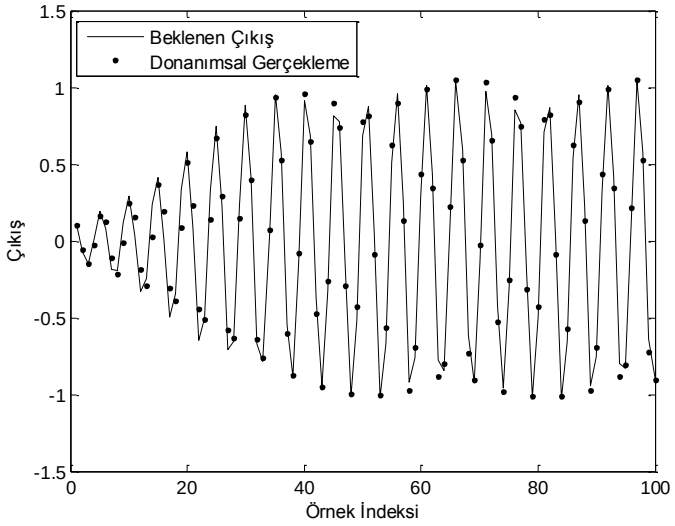
(a)



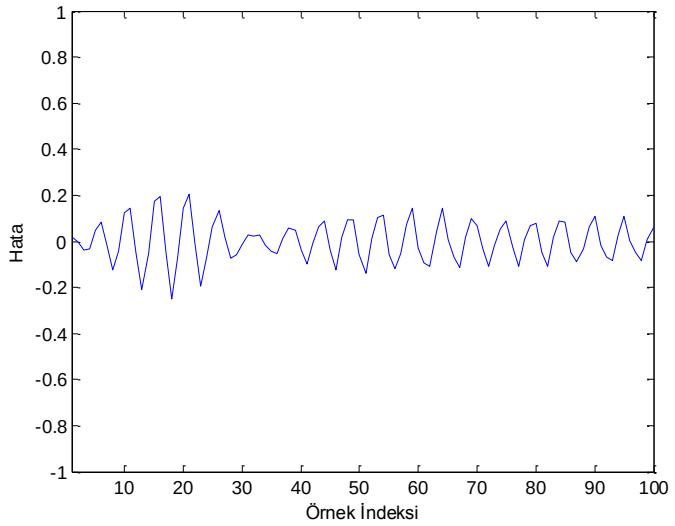
(b)

Şekil 4.11. Örnek 2 için geriye yayılım algoritması kullanılarak eğitilen ağa ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.12a'da Örnek 2 için LM algoritması ile dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.12b'de Örnek 2 için LM algoritması ile dinamik sistem tanımanın donanımsal gerçeklemesine ait hataları göstermektedir.



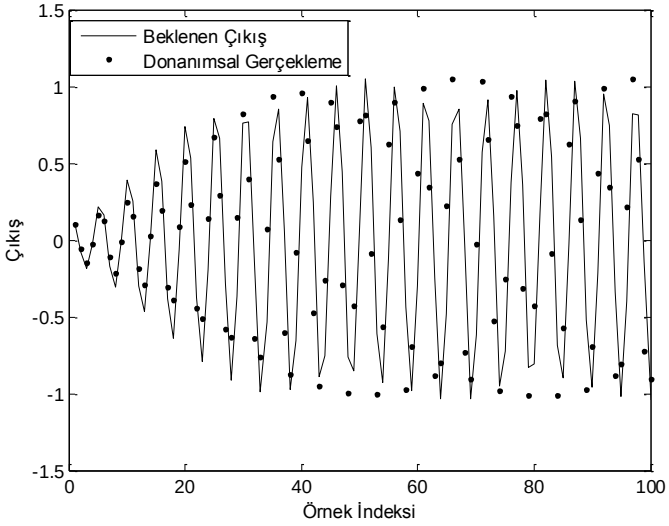
(a)



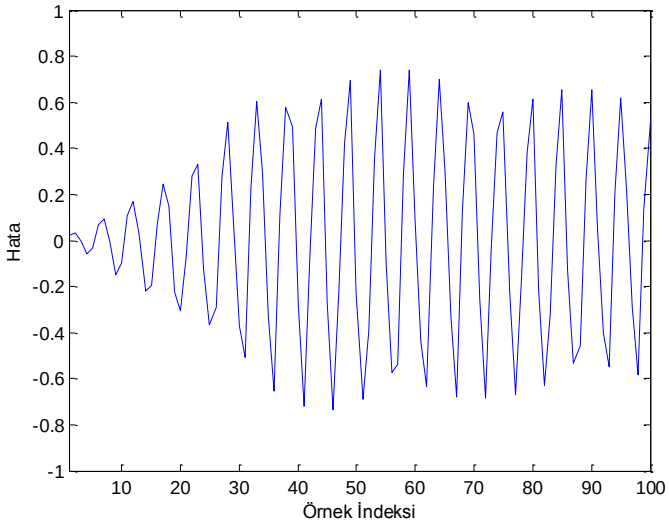
(b)

Şekil 4.12. Örnek 2 için LM algoritması kullanılarak eğitilen ağa ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.13a'da Örnek 2 için PSO algoritması ile dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.13b'de Örnek 2 için PSO algoritması ile dinamik sistem tanımanın donanımsal gerçeklemesine ait hataları göstermektedir.



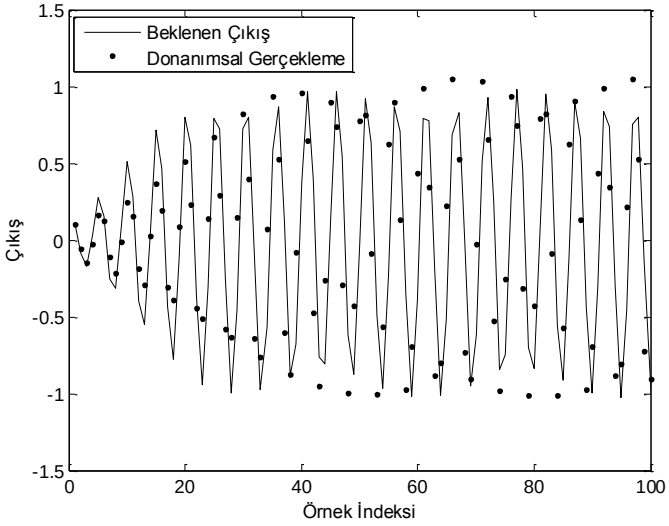
(a)



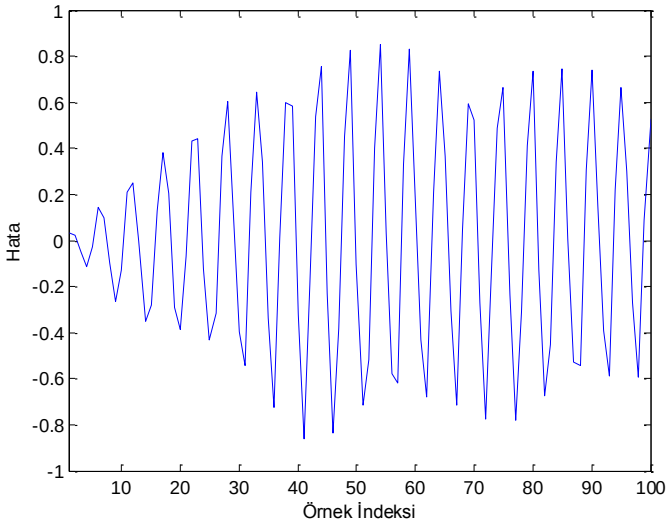
(b)

Şekil 4.13. Örnek 2 için PSO algoritması kullanılarak eğitilen ağa ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.14a'da Örnek 2 için YAK algoritması ile dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.14b'de Örnek 2 için YAK algoritması ile dinamik sistem tanımanın donanımsal gerçeklemesine ait hataları göstermektedir.



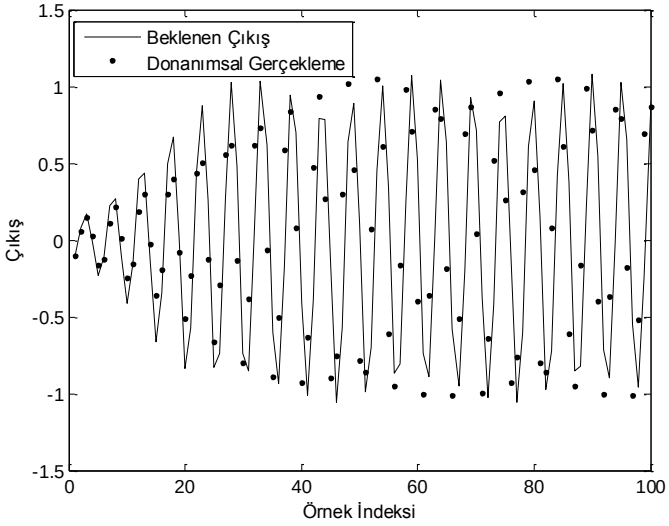
(a)



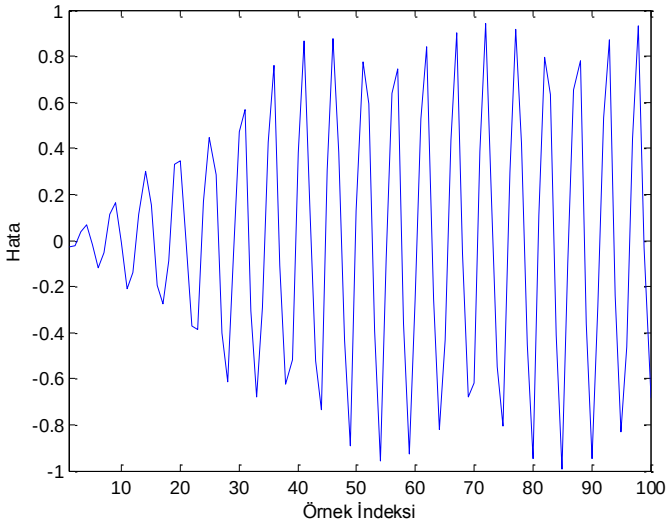
(b)

Şekil 4.14. Örnek 2 için YAK algoritması kullanılarak eğitilen ağa ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.15a'da Örnek 2 için $y[0]=y[-1]=0.1$ başlangıç değerleri ile oluşturulan giriş dizisi ile geriye yayılım algoritması ile eğitilmiş ağın $y[0]=y[-1]=-0.1$ başlangıç değerleri ile oluşturulan giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.15b'de, geriye yayılım algoritması ile eğitilmiş ağın $y[0]=y[-1]=-0.1$ başlangıç değerleri ile oluşturulan giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal gerçekleme hataları gösterilmektedir.



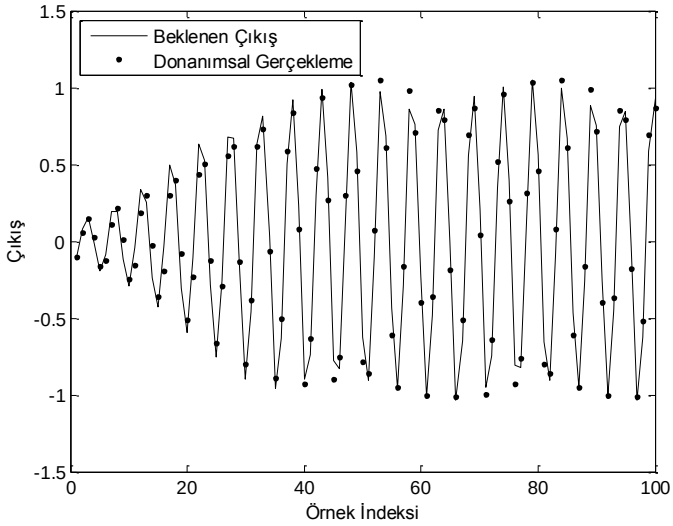
(a)



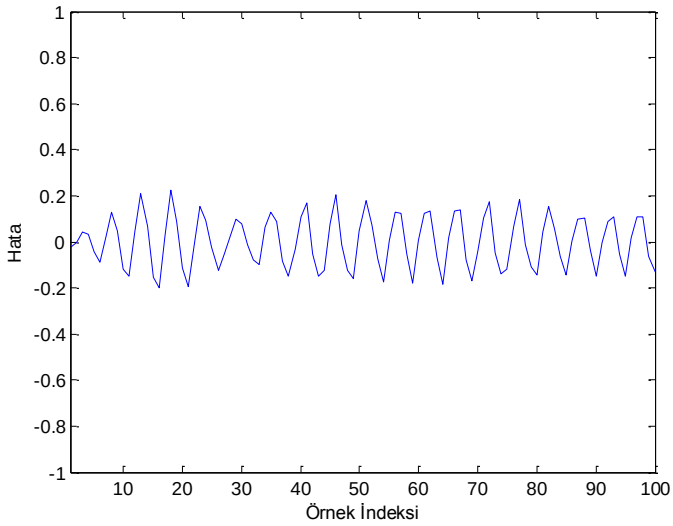
(b)

Şekil 4.15. Örnek 2 için geriye yayılım algoritması kullanılarak eğitilen ağa gösterilmeyen giriş değerlerine ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.16a'da Örnek 2 için $y[0]=y[-1]=0.1$ başlangıç değerleri ile oluşturulan giriş dizisi ile LM algoritması ile eğitilmiş ağın $y[0]=y[-1]=-0.1$ başlangıç değerleri ile oluşturulan giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.16b'de, LM algoritması ile eğitilmiş ağın $y[0]=y[-1]=-0.1$ başlangıç değerleri ile oluşturulan giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal gerçekleme hataları gösterilmektedir.



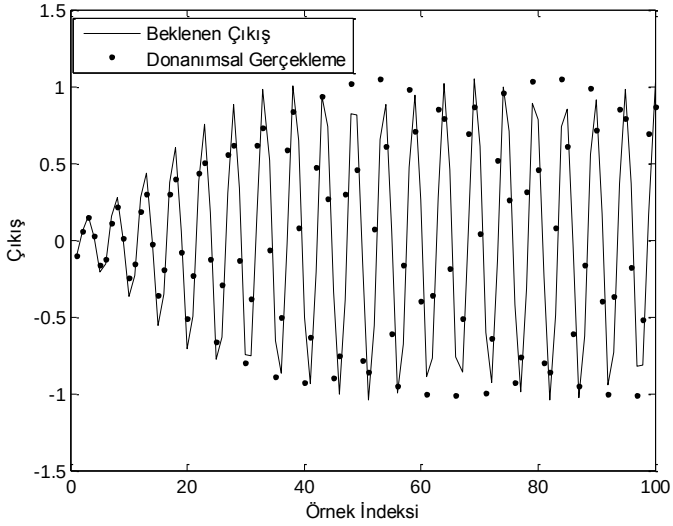
(a)



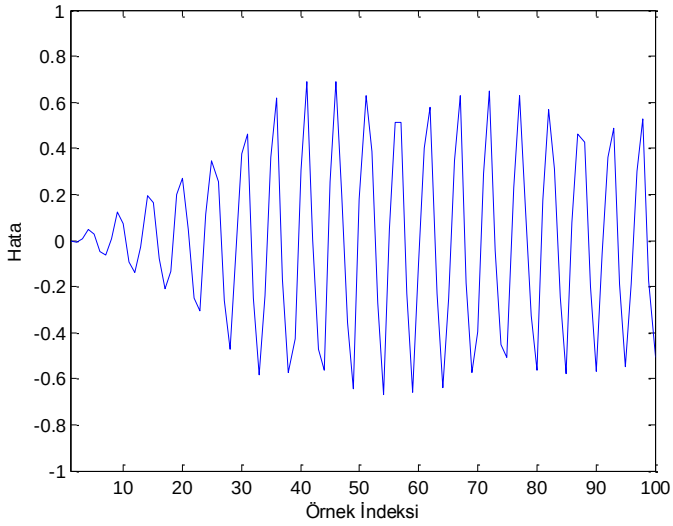
(b)

Şekil 4.16. Örnek 2 için LM algoritması kullanılarak eğitilen ağa gösterilmeyen giriş değerlerine ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.17a'da Örnek 2 için $y[0] = y[-1] = 0.1$ başlangıç değerleri ile oluşturulan giriş dizisi ile PSO algoritması ile eğitilmiş ağın $y[0] = y[-1] = -0.1$ başlangıç değerleri ile oluşturulan giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.17b'de, PSO algoritması ile eğitilmiş ağın $y[0] = y[-1] = -0.1$ başlangıç değerleri ile oluşturulan giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal gerçekleme hataları gösterilmektedir.



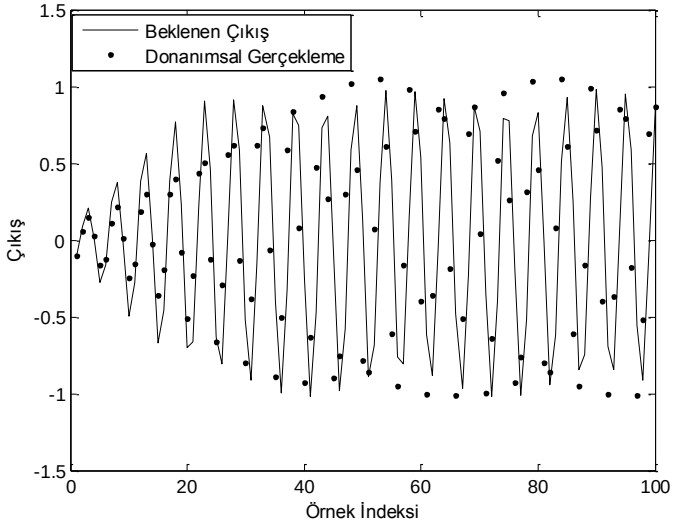
(a)



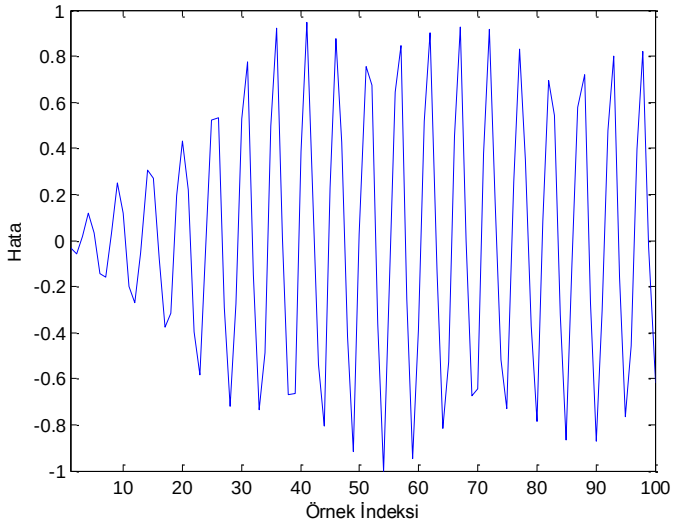
(b)

Şekil 4.17. Örnek 2 için PSO algoritması kullanılarak eğitilen ağa gösterilmeyen giriş değerlerine ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Şekil 4.18a'da Örnek 2 için $y[0] = y[-1] = 0.1$ başlangıç değerleri ile oluşturulan giriş dizisi ile YAK algoritması ile eğitilmiş ağın $y[0] = y[-1] = -0.1$ başlangıç değerleri ile oluşturulan giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal çıktıları göstermektedir. Şekil 4.18b'de, YAK algoritması ile eğitilmiş ağın $y[0] = y[-1] = -0.1$ başlangıç değerleri ile oluşturulan giriş dizisi için dinamik sistem tanıma işlemine ait donanımsal gerçekleme hataları gösterilmektedir.



(a)



(b)

Şekil 4.18. Örnek 2 için YAK algoritması kullanılarak eğitilen ağa gösterilmeyen giriş değerlerine ait deneysel sonuçlar: a) Deneysel başarım b) Sistem tanıma hatası

Çizelge 4.3'de ağa gösterilen giriş dizisine göre eğitilen ağın eğitimde kullanılan algoritmalara göre uygunluk değerleri ile ağa gösterilmeyen giriş dizisi için elde edilen uygunluk değerleri verilmiştir. Uygunluk değerleri 10 farklı eğitim sonucunda elde edilen uygunluk değerlerinin ortalaması olarak verilmiştir. Şekil 4.19'da eğitim verileri üzerinde elde edilen hata değerleri tek grafikte gösterilmiştir. Şekil 4.20'de ise test verileri üzerinde elde edilen hata değerleri tek grafikte gösterilmiştir.

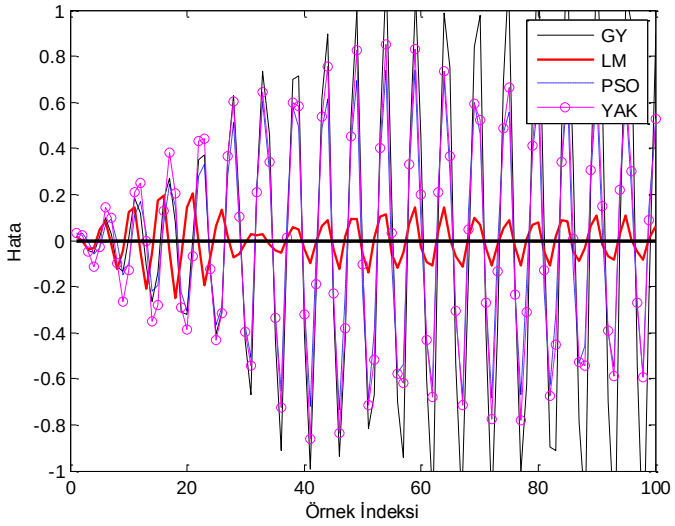
Çizelge 4.4'de GY, LM, PSO ve YAK algoritma ile eğitilmiş YSA'nın FPGA üzerinde gerçeklenmesine ilişkin sentez sonuçları verilmiştir.

Çizelge 4.3. Örnek 2 için eğitim ve test verilerine göre ağ uygunluk değerleri

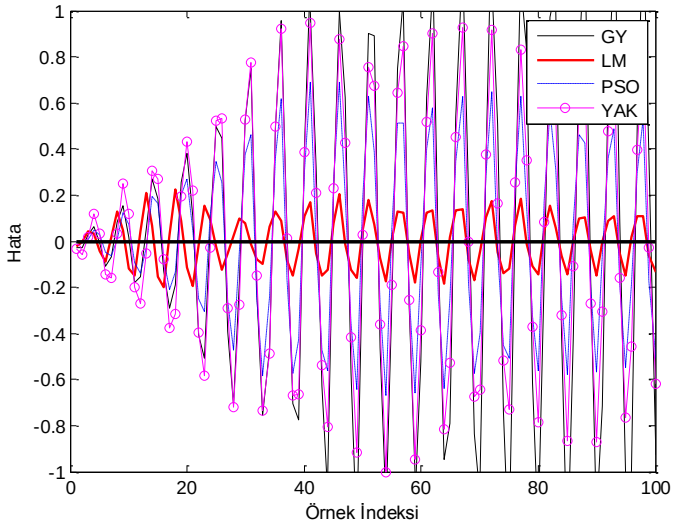
	Eğitim	**Test**	**İterasyon**
GY	21.5792	24.8085	2000
LM	0.3844	0.6155	20
PSO	8.3222	7.0423	2000
YAK	10.5075	14.2890	2000

Çizelge 4.4. Örnek 2 için sentez sonuçları

		Slice Reg.	**Slice LUTs**	**Block RAM**	**DSP48E1s**
	Mevcut	407600	203800	445	840
BP	Kullanılan	3503	20442	2	82
	Yüzde	0	10	0	9
LM	Kullanılan	19314	53709	22	150
	Yüzde	4	26	4	17
PSO	Kullanılan	5587	18372	8	29
	Yüzde	1	9	1	3
YAK	Kullanılan	5639	18442	4	26
	Yüzde	1	9	0	3

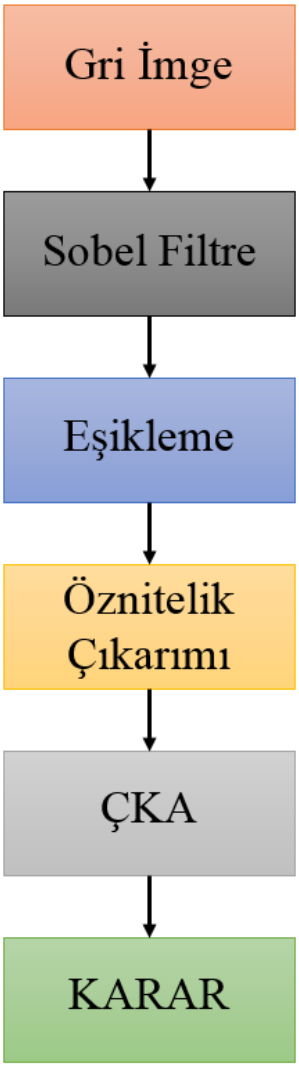


Şekil 4.19. Örnek 2 için eğitim datalarında elde edilen hata grafiği



Şekil 4.20. Örnek 2 için test verilerinde elde edilen hata grafiği

Örnek 3:Bu örnekte plaka yeri bulma tespiti için GY, L&M, PSO ve YAK algoritmaları kullanılarak YSA eğitimini gerçeklenmiştir. YSA plaka yeri bulma işleminde seçilen aday bölgenin plaka olup olamadığına ilişkin karar vermek üzere kullanılmıştır (Şekil 4.21).



Şekil 4.21. Plaka tespit işlem süreci

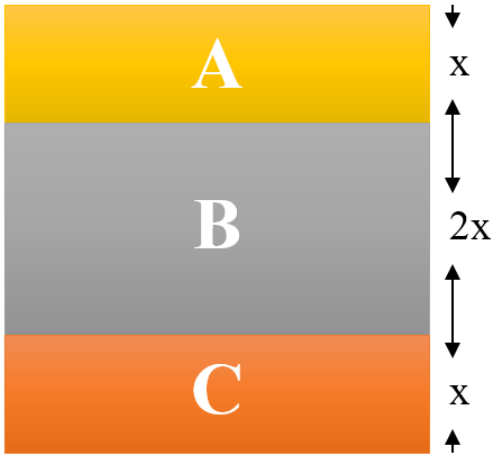
Şekil 4.21'dende görüleceği üzere bu örnekte ÇKA'ya giriş olarak seçilen plaka bölgesinin kenarları, dikey sobel filtresi kullanılarak belirginleştirilmiştir. Daha sonra bu kenar değerleri eşikleme işlemine tabi tutulmuştur. Eşikleme sonucunda elde edilen gri imge ikili sayı formatına çevrilmiştir. Elde edilen yeni imge üzerinde seçilen bölgenin plaka olup olmadığın karar verilebilmesi için seçilen bölgeden öznitelik çıkarımları yapılmıştır (Çavuşlu, Karakaya ve Altun 2008). Bu özniteliklerden birincisi ortalama değer hesabı ile yapılmaktadır. Denklem (4.5)'den de görüleceği üzere ortalama değer hesabı seçilen aday bölgeye ait toplam siyah piksel değerlerinin, aday bölge piksel sayısına oranı ile edilmektedir. İkinci öznitelik ise aday bölge de değişinti (varyans)hesaplanması ile yapılmaktadır (Denklem (4.6)). Üçüncü öznitelik ise plaka aday bölgesinin yatayda 3 parçaya bölünmesi ile elde edilmektedir. A ve C parçaları

birbirine eşit piksel sayısına sahip, B parçası ise diğer parçaların 2 kat piksel sayısına sahiptir (Şekil 4.22). Daha sonra bu parçalara ait toplam siyah piksel değerleri ayrı ayrı hesaplanır. B bölgesine ait toplam siyah piksel değeri, A ve C bölgesine ait toplam siyah piksel değerinden farkının, aday bölgeye ait piksel sayısına oranı ile bulunur (Denklem (4.7))

$$Öznitelik\_1 = Ortalama\_Deger = \frac{Siyah\_Piksel}{Toplam\_Piksel} \tag{4.5}$$

$$Öznitelik\_2 = Degi\sin ti = \frac{1}{N}\sum_{i=1}^{N}\left(İmge(\mathrm{i}) - \mathrm{Ortalama\_Deger}\right)^2 \tag{4.6}$$

$$Öznitelik\_3 = Rc = \frac{Siyah\_\mathrm{Piksel\_B} - \left(Siyah\_\mathrm{Piksel\_}A + Siyah\_\mathrm{Piksel\_}C\right)}{Toplam\_\mathrm{Piksel}} \tag{4.6}$$



Şekil 4.22. Plaka tespit işlem süreci

Bu örnek için 3 adet öznitelik kullanılması nedeniyle ÇKA 3 girişli olarak tasarlanmıştır. Gizli katmanında 3 ve çıkış katmanında ise 1 hücre kullanılmıştır. Gizli katman hücreleri için aktivasyon fonksiyonu olarak logaritmik sigmoidal fonksiyonu, çıkış hücresi için lineer aktivasyon fonksiyonu kullanılmıştır. Ağ eğitiminde 50 adet plaka (Şekil 4.23a) ve 50 adet plaka olmayan(Şekil 4.23b) toplam 100 adet eğitim seti kullanılmıştır.

(a) (b)

Şekil 4.23. Plaka ve plaka olmayan imgeler

Çizelge 4.5'te GY, L&M, PSO ve YAK algoritmaları ile eğitilmiş ÇKA'nın plaka tanıma işlemindeki başarımları gösterilmiştir. Çizelge 4.6'da GY, LM, PSO ve YAK algoritma ile eğitilmiş YSA'nın FPGA üzerinde gerçeklenmesine ilişkin sentez sonuçları verilmiştir.

Çizelge 4.5. GY, L&M, PSO ve YAK algoritmaları ile eğitilmiş ÇKA'nın plaka tanıma işlemindeki başarımları

		Doğru	**Yanlış**	**Toplam**	**Yüzde**	**İterasyon**
BP	Plaka	1824	39	1863	97,15	**2000**
	Plaka Değil	1850	13	1863	96,99	
	Toplam	**3674**	**52**	**3726**	**98,60**	
LM	Plaka	1820	43	1863	97,69	**100**
	Plaka Değil	1852	11	1863	17	
	Toplam	**3672**	**54**	**3726**	**98,55**	
PSO	Plaka	1771	92	1863	95,06	**2000**
	Plaka Değil	1822	41	1863	97,79	
	Toplam	**3593**	**133**	**3726**	**96,43**	
YAK	Plaka	1853	10	1863	99,46	**2000**
	Plaka Değil	1829	34	1863	98,17	
	Toplam	**3682**	**44**	**3726**	**98,82**	

Çizelge 4.6. Örnek 3 için sentez sonuçları

		Slice Reg.	Slice LUTs	Block RAM	DSP48E1s
	Mevcut	407600	203800	445	840
BP	Kullanılan	3824	26874	3	100
	Yüzde	0	13	0	11
LM	Kullanılan	21977	66773	27	186
	Yüzde	5	32	6	22
PSO	Kullanılan	5486	21194	9	34
	Yüzde	1	10	2	4
YAK	Kullanılan	5699	21514	5	30
	Yüzde	1	9	1	3

BÖLÜM V

YORUMLAR

YSA'ların en önemli avantajı paralel veri işleme özelliğidir. Bu bakımdan, YSA'nın paralel veri işleme özelliğinin ön plana çıkacak donanım ortamında gerçeklenmesi çok önemlidir. FPGA'lar, YSA'ların bu özelliğini gerçeklemek için uygun bir platformdur.

Bu çalışmada GY, LM, PSO ve YAK öğrenmeli YSA'nın FPGA üzerinde gerçeklenmesi sunulmuştur. Donanımsal gerçekleme dinamik sistem tanıma ve plaka yeri bulma problemleri kullanılarak test edilmiş ve elde edilen deneysel sonuçlar karşılaştırmalı olarak verilmiştir.

Elde edilen sonuçlar, GY, LM, PSO ve YAK öğrenmeli YSA'nın başarılı bir şekilde FPGA donanımı üzerinde gerçeklenebildiğini göstermektedir.

Donanımsal gerçeklemeyi test etmek için kullanılan örneklerden birincisi olan sistem tanıma probleminin sonuçları analiz edildiğinde LM ve PSO algoritmaları kullanılarak eğitilen YSA'nın daha başarılı bir genelleme yaptığı ancak YAK ve GY ile eğitilen YSA'nın kötü bir performans sergilediği gözlemlenmiştir. Yine bir sistem tanıma problemi olan ikinci örnekte de LM algoritmalası ile eğitilen YSA en iyi genellemeyi yaparken bu kez GY algoritması ile eğitilen YSA'nın en kötü performansı sergilediği gözlemlenmiştir. Sistem tanıma problemlerinde LM ile eğitilen YSA'nın açık ara en iyi sonuçları sağladığı görülmektedir. Üstelik LM bu başarıyı diğerlerine oranla çok daha az bir iterasyonda gerçekleştirmiştir.

Üçüncü örneğimiz olan plaka yeri bulma probleminde ise LM, BP ve ABC ile eğitilen YSA'ların birbirine yakın başarımlar sergilerken, PSO ile eğitilen YSA'nın performansının düşük kaldığı gözlemlenmiştir. LM, BP ve ABC ile eğitilen YSA'lar içinde ise ABC ile eğitilen ağın az bir farkla da olsa en iyi sonucu verdiği gözlemlenmiştir.

Türev bilgisi gerektiren algoritmaların, evrimsel algoritmalara göre daha çok alan kapladığı görülmektedir. LM algoritması matris çarpımı, matrisin tersini alma gibi işlemler gerektirdiği için en fazla alan tüketen algoritma olmuştur.

KAYNAKLAR

Akay, B., ve Karaboga, D., "Parameter Tuning for the Artificial Bee Colony Algorithm", ***1st International Conference on Computational Collective Intelligence - Semantic Web, Social Networks & Multiagent Systems***, 5-7 October, 2009 Wrocław

Brysbaert, M., "Algorithms for randomness in the behavioral sciences: A tutorial". ***Behavior Research Methods, Instruments, & Computers,*** Volume 23, Pages 45-60,1991

Chen, S., ve Billings, S.A., "Neural networks for nonlinear dynamic system modelling and identification", ***Int. J. Control***, vol. 56, no 2, pp. 319-346, 1992.

Çavuşlu, M.A., Karakaya, F. ve Altun, H., "ÇKA Tipi Yapay Sinir Aği Kullanılarak Plaka Yeri Tespitinin FPGA'da Donanımsal Gerçeklenmesi", in Proc. ***Akıllı Sistemlerde Yenilikler ve Uygulamalar Sempozyumu 2008 (ASYU 2008)*** (Isparta, Turkey, 2008).

Çavuşlu, M.A., Karakuzu, C., ve Şahin, S., "Neural Networkwork Hardware Implementation Using FPGA", ***in ISEECE 2006 3rd International Symposium on Electrical, Electronic and Computer Engineering Symposium Proceedings***, Nicosia, TRNC,, Pages. 287-290, 2006.

Çavuşlu, M.A., Karakuzu, C., "Nöral ve Bulanık Sistem Hücre Aktivasyon Yaklaşımları ve FPGA'da Donanımsal Gerçeklenmesi", ***Sakarya Üniversitesi Fen Bilimleri Enstitüsü Dergisi***, 15. Cilt, 1. Sayı, s.8-16, 2011

Çavuşlu, M. A. , Karakuzu, C., Şahin, S., Yakut, M., "Neural Network Training Based on FPGA with Floating Point Number Format and It's Performance", ***Neural Computing & Applications,***, Volume 20, Number 2,Pages: 195-202, March 2011

Çavuşlu, M.A., Karakuzu, C. ve Karakaya, F., "Neural Identification of Dynamic Systems on FPGA with PSO Learning", ***Applied Soft Computing***, Volume 12, Issue 9, Pages:2707–2718, 2012

Dohnal, J.: "Using of Levenberg-Marquardt Method in Identification by Neural Networks." ***In Student EEICT 2004. Student EEICT*** 2004. Brno: Ing. Zdenk Novotn CSc., 2004, str. 361 - 365, ISBN 80-214-2636-5

Economou, G.P.K., Mariatos, E.P., Economopoulos, N.M., Lymberopoulos, D. ve Goutis, C.E., "FPGA implementation of artificial neural networks: an application on medical expert systems", ***in: Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems,*** 1994, pp. 287-293.

Elliot, D. L., "A Better Activation Function for Artificial Neural Networks", ***Technical Research Report T.R. 93-8***, Institute for Systems Research, University of Maryland, 1993.

Farmahini-Farahani, A., Fakhraie, S. M., Safari, S., "Scalable Architecture for on-Chip Neural Network Training using Swarm Intelligence", ***Proc. of the Design, Automation and Test in Europe Conf.*** (DATE'08), Munich, Germany, Pages.1340-1345, 2008.

Ferrari, S. ve Jensenius, M. "A constrained optimization approach to preserving prior knowledge during incremental training," ***IEEE Trans. Neural Netw***., vol. 19, no. 6, pp. 996–1009, Jun. 2008.

Ferreira, P., Ribeiro, P., Antunes, A. ve Dias, F.M. "A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function", ***Neurocomputing***, Volume 71, Issue (1-3), Pages. 71-77, 2006.

Ferrer, D., Gonzalez, R., Fleitas, R., Acle, J.P., ve Canetti, R., "NeuroFPGA - Implementing Artificial Neural Networks on Programmable Logic Devices",

Proceedings of Design, Automation and Test in Europe Conference and Exhibition, Volume 3, Pages.218-223, 2004.

Gallagher S, "Accelerating DSP Algorithms Using FPGAs.", http://klabs.org/mapld04/presentations/session_p/p188_gallagher_s.ppt. p 6, Erişim 07 Ekim 2013

Haridas, N., ve Devi, M. N., "PSO learning on artificial neural networks", ***in Proc. A2CWiC***, 2010, pp.3-3.

Hashemipoor, S.S, Suratgar, A.A., ve Fard, A, "Designing a Robust MEMS AC Voltage Reference Source Using Artificial Neural Network", ***Australian Journal of Basic and Applied Sciences,*** 4(6): 1183-1189, 2010

Haykın, S. "***Neural Networks A Comprehensive Foundation***", 2nd edition, Prentice Hall Publishing, New Jersey 07458, USA, 1999

Karaboga, D., "Artificial Bee Colony (ABC) Algorithm on Training Artificial Neural Networks", ***IEEE 15th Signal Processing and Communications Applications***, 2007. SIU 2007. Page(s): 1 – 4

Karaboga, D. Akay, B. ve Ozturk, C., "Artificial Bee Colony (ABC) Optimization Algorithm For Training Feed-Forward Neural Networks", ***Modeling Decisions for Artificial Intelligence, 4th International Conference,*** MDAI 2007, Japan, 16-18 August 2007, vol.4617/2007, pp.318-319.

Karaboğa, D., ve Akay, B., "A comparative study of Artificial Bee Colony algorithm", ***Applied Mathematics and Computation*** vol : 214, 108–132, 2009

Karaboga, D., "An Idea Based On Honey Bee Swarm For Numerıcal Optımızatıon",***Technıcal Report-Tr06,Erciyes University***, Engineering Faculty, Computer Engineering Department, 2005

Karakuzu C. ve Öztürk S, "A Comparison of fuzzy, neuro and classical control techniques based on an experimental application", ***Journal of Quafquaz University***, No:6, Fall 2000 p. 189-198.

Kennedy, J., ve Eberhart, R.C., "Particle Swarm Optimization", ***Proc, IEEE Int. Conf. Neural Network IV***, Porth Australia, pp.1942-1948, 1995.

Khosravi.Z, M.H, Barghinia S., Ansarimehr, "New Momentum Adjustment Technique for Levenberg-Marquardt Neural Network Used in Short Term Load Forecasting", ***21st International Power System Conference*** (PSC 2006), Tehran, Iran.

Krips, M., Lammert, T., ve Kummert, A., "FPGA implementation of a neural network for a real-time handtracking system", ***in Proceedings of The First IEEE International Workshop on Electronic Design***, Test and Applications (2002) 313 – 317.

Kumbhar, P. Y., ve Krishnan, S., "Use Of Artificial Bee Colony (ABC) Algorithm in Artificial Neural Network Synthesis***", International Journal Of Advanced Engineering Sciences And Technologies***, Vol No. 11, Issue No. 1, 162 – 171, 2011.

Li, X. ve Areibi, S., "A Hardware Software Co-design Approach for Face Recognaiton",Microelectronics,. ***ICM 2004 Proceedings. The 16th International Conference on***, p: 55- 58, 6-8 Dec. 2004.

Li, Y. ve Chen, X., "A New Stochastic PSO Technique for Neural Network Training. ***International Symposium on Neural Networks (ISNN)***, Chengdu (China), pp. 564-569. 2006.

Mandal, S. K., Sural, S. ve Patra, A., "ANN- and PSO-Based Synthesis of On-Chip Spiral Inductors for RF ICs", ***IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems***, Vol. 27, No. 1, pp. 188-192, 2008.

Martinez, J., Toledo, F.J., Fernandez, E., ve J.M. Ferrandez, "A retinomorphic architecture based on discrete-time cellular neural networks using reconfigurable computing", **Neurocomputing**, 2008, Volume 71, Issues 4-6, January 2008, Pages 766-775

Merchant, S., Peterson, G. ve Kong, S. "Intrinsic Embedded Hardware Evolution of Block-based Neural Networks", ***International Conference on Engineering of Reconfigurable Systems & Algorithms***, June 26-29, 2006,

Mousa, M., Areibi, S. ve Nichols, K., "On the Arithmetic Precision for Implementing Back-Propagation Networks on FPGA: A Case Study," ***in FPGA Implementations of Neural Networks***, eds. Amos R. Omondi and Jagath C. Rajapakse (Springer US, 2006), pp 37-61

Narendra, K. S., Parthasaraty, K.,"Identification and Control of Dynamical Systems Using Neural Network", ***IEEE Transactions on Neural Networkworks*** 1(1) (1990) 4-27.

Nedjah, N., Silva, R.M.D., Mourelle, L.M.M., ve Silva, M.V.C.D., "Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs", ***Neurocomputing,*** Volume 72, Issues 10-12, Pages.2171-2179, 2009.

Ossoinig, H., Reisinger, E., Steger, C. ve Weiss, R., "Design and FPGA-Implementation of a Neural Networkwork", ***in Proceedings of the 7th International Conference on signal Processing Applications & Technology*** (Boston, USA, 1996) pp. 939-943.

Öztemel, E., ***"Yapay Sinir Ağları***", Papatya Yayıncılık, İstanbul, 2003.

Sahin, S., Becerikli, Y., ve Yazici, S., "Neural Network Implementation in Hardware Using FPGAs", ***Lecture Notes in Computer Science*** 4234 (2006) 1105-112.

Savich, A.W., Moussa, M., Areibi,S., "The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study", ***IEEE Transactions on Neural Networks***, Volume 18, Issue 1, Pages:240 – 252, 2007.

Vilovic, I., Burum, N. ve Milic, D., "Using particle swarm optimization in training neural network for indoor field strength prediction" , ***International Symposium ELMAR***, 2009. ELMAR '09. Page(s): 275 – 278.

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

Won, E., "A hardware implementation of articial neural nertworks using field programmable gate arrays", N***uclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment***, Volume 581, Issue 3, Pages 816-820, 1 November 2007.

Yu, X., ve Deni D. "Implementing Neural Networks In FPGAs", ***The Institution of Electrical Engineers, IEE published***, Savoy Place, London WC2R 0BL, UK, 1999.

Zhu, J., Milne, G.J., Gunther, ve B. K., "Towards an FPGA Based Reconfigurable Computing Environment for Neural Network Implementations", ***in Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99) IEE Conference Proceedings 470 (Edinburgh, UK ,1999)***, pp.661-666.

ÖZ GEÇMİŞ

Mehmet Ali Çavuşlu 06.12.1983 tarihinde Kocaeli ili Alikahya beldesinde doğdu. 2001yılında Kocaeli Üniversitesi Elektronik ve Haberleşme Mühendisliği Bölümünü kazandı. İlk yılı hazırlıkla geçen üniversite hayatını 2006 yılında tamamladı. Özel sektörde Ar-Ge Mühendisi, Proje Yürütücüsü olarak çalıştıktan sonra şu anda Koç Bilgi ve Savunma Teknolojileri A.Ş. şirketinde Gömülü Yazılım Geliştirme Uzmanı olarak çalışmaktadır. Uzmanlık alanları FPGA, yapay zekâ, sinyal ve görüntü işleme, sualtı akustiği, optimizasyon'dur.