İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

KONTROL SİSTEMLERİNDE YAPAY SİNİR AĞI UYGULAMALARI

YÜKSEK LİSANS TEZİ

Müh. Sertaç CANARAN

Tezin Enstitüye Verildiği Tarih : 15 Ocak 1996
Tezin Savunulduğu Tarih : 2 Şubat 1996

Tez Danışmanı : Prof. Dr. Leyla GÖREN
Diğer Jüri Üyeleri : Doç. Dr. Cüneyt GÜZELİŞ
Doç. Dr. İbrahim EKSİN

ŞUBAT 1996

ÖNSÖZ

Tezimi oluştururken değerli yönlendirmelerinden dolayı danışmanım Prof. Dr. Leyla GÖREN'e ve konu hakkında fikir alışverişi yapmaktan zevk duyduğum arkadaşım Araş. Gör. Mehmet YILDIZDOĞAN'a sonsuz teşekkürler.

Sertaç CANARAN
Ocak 1996

İÇİNDEKİLER

ÖZET

Bu tezde, günümüzde kullanım alanları gittikçe artan Yapay Sinir Ağları konusu içerisinde yer alan CMAC (Cerebellar Model Arithmetic Computer) yaklaşımı ele alınarak çeşitli uygulamalara ilişkin simülasyon sonuçları verilmiştir.

Bölüm 1'de Yapay sinir ağları hakkında genel bilgiler kısaca verilmiştir.

Bölüm 2'de CMAC'ın dayandığı nörolojik model verilerek kullanılan dönüşüm ve öğrenme algoritmaları incelenmiş, çeşitli özellikleri verilerek diğer yapay sinir ağı modelleri ile karşılaştırılması yapılmıştır.

Bölüm 3'de CMAC tek değişkenli fonksiyonların öğrenilmesinde, ters kinematik denklemlerin çözümünde ve sistem belirleme işleminde kullanılmış, yapılan simülasyon sonuçları verilmiştir.

SUMMARY

ARTIFICIAL NEURAL NETWORK APPLICATIONS IN CONTROL SYSTEMS

The recent resurgence of interest in neural networks has its roots in the recognition that brain performs computations in a different manner than do conventional digital computers. Computers are extremely fast and precise at executing sequences of instructions that have been formulated for them. A human information processing system is composed of neurons switching at speeds about a million times slower than computer gates. Yet, humans are more efficient than computers at computationally complex tasks such as speech understanding. Moreover, not only humans, but even animals can process visual information better than the fastest computers.

A neural network's ability to perform computations is based on the hope that we can reproduce some of the flexibility and power of the human brain by artificial means. Network computation is based is performed by a dense mesh of computing nodes and connections. They operate collectively and simultaneously on most or all data and inputs.

The basic processing elements of neural networks are called artificial neurons, or simply neurons. Often they are called as nodes. Neurons perform as summing and nonlinear mapping junctions. In some cases they can be considered as threshold units that fire their total input exceeds certain bias levels. Neurons usually operate in parallel and configured inregular architectures. They are often organized in layers, and feedback connections both within the layer and toward adjacent layers are allowed. Each connection strength is expressed by a numerical value called a weight, which can be modified.

The attractive feature of the neural networks is their learning capabilities. They can learn complex nonlinear relationships trough a training procedure which is done by changing the connection weights with a learning

algorithm. There are different architectures and different learning algorithms which are used for different purposes.

The CMAC (Cerebeller Model Arithmetic Computer) algorithm which is a less known technique as artificial neural networks, imitates the human cerebellum.

The cerebellum which is attached to the midbrain and nestles up under the visual cortex is intimately involved with control of rapid, precise, coordinated movements of limbs, hands and eyes. Injury to the cerebellum results in motor deficiencies such as overshot in reaching for objects, lack of coordinating and inability to execute delicate tasks or track precisely with the eyes.

Input to the cerebellum arrives in the form of sensory and proprioceptive feedback from the muscles, joints and skin together with commands from higher level motor centres concerning what movement is to be performed. This input constitutes an address, the contents of which are the appropriate muscle actuator signals required to carry out the desired movement. At each point of the time the input addresses an output which drives the muscle control circuits. The resulting motion produces a new input and the process is repeated. The result is a trajectory of the limb trough space. At each point on the trajectory the state of the limb is sent to the cerebellum as input and the cerebellar memory responds with actuator signals which drive the limb to the next point on the the trajectory.

In the case of N input variables with R distinguishable levels there are R^N possible inputs and it is assumed that the brain does not have sufficient cells to store the corresponding control values for this amount of inputs and uses a memory management algorithm to solve the problem

The theory of the CMAC was first proposed by J.S. Albus. CMAC is a neuromuscular control system by which control functions for many degrees of freedom can be computed simultaneously in a table look-up fashion rather than mathematical solution of simultaneous equations.

The CMAC algorithm can be classified as an advanced look-up table technique with learning capabilities.

Normal look-up table techniques transform an input vector into a pointer, which is used to index a table. With this pointer the input vector is

uniquely coupled to a weight (table location). Using normal look-up table techniques a number of problems occur. The main problems are:

- The input signals must be quantized. Quantization adds quantizing noise to the input signals. The quantization interval can not be chosen too small because of the increasing table size.

- The size of the table increases exponentially as the number of inputs increases.

- Output values corresponding to nearby input vectors are not correlated because every weight is uniquely coupled to a point in the quantized input space. But in smooth-continuous functions nearby input vectors produce similar outputs. Normal look-up table techniques do not use this property to their advantage.

Despite these disadvantages of look-up table techniques, two major advantages exist:

- A function of any shape can be learned because the function is stored numerically, so linearity is no constraint.

- Can be very fast when simple addressing algorithms used.

The extensions of CMAC to the normal look-up table methods are: Distributed storage and Generalization algorithm.

Distributed storage means that the output value CMAC is stored as the sum of number of weights.

The generalization algorithm uses the distributed storage feature in a clever way: nearby input vectors address many weights commonly.
The number of selected (addressed) weights (memory locations) is called as ρ, generalization number. In normal CMAC applications $\rho>1$, for $\rho=1$ the algorithm reduces a simple look-up table.

As a result of generalization, CMAC memory addresses in the same neighborhood are not independent. Data storage at ant point alters the values stored at neighboring points, so the training is not required for all possible input vectors in the input space.

The training of the CMAC table is done by taking the difference between the actual and desired output and distributing equal shares of that difference to ρ memory locations.(For more detail see part 2) Changing ρ locations instead of all weights in entire network is a significant advantage of CMAC over other neural network algorithms. The result of this property is fast learning and fast response.

Because function values are simply stored, CMAC can only generate a reasonable output in an area where learning has been performed.

Compared to the normal look-up methods CMAC offers the advantages of memory saving and generalization. Compared to other neural network algorithms CMAC can be trained in significially shorter time because for each training data only a small subset of the memory table adjusted instead of the entire network. The calculating effort is lower but memory consumption is higher.

İn part 3, CMAC is used for;

- Learning single variable functions showing the properties,
- Solving Inverse kinematics equations,
- System Identification.

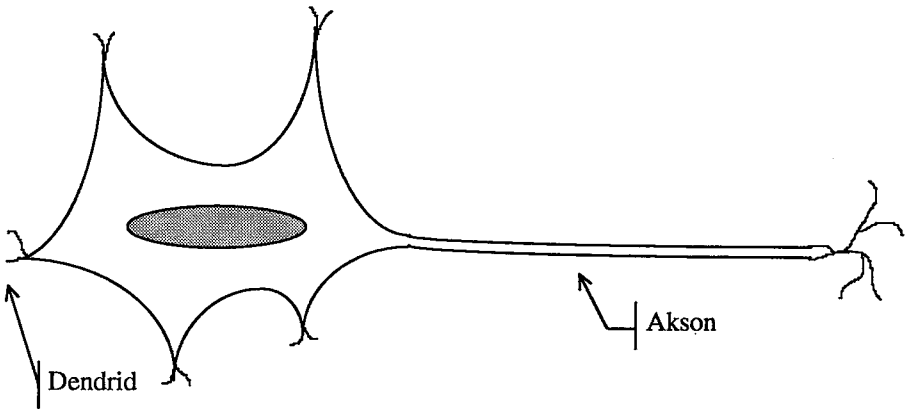
BÖLÜM 1. YAPAY SİNİR AĞLARI

Günümüzde yapay sinir ağlarına olan ilgi, bir takım problemlerin çözümüne farklı yaklaşımlar getirdikleri için gün geçtikçe artmaktadır. Bilgisayarlar kendilerine tanımlanan problemleri ardışıl olarak çok hızlı bir şekilde yapabilmektedirler. Fakat insanların bilgi'yi işleme birimi olan sinir hücreleri bilgisayarlara göre çok daha yavaş çalışmakla beraber görüntü,ses işleme ve tanıma gibi problemlerde çok daha başarılı olmaktadırlar. Bu üstünlük yoğun paralel veri işlemeden kaynaklanmaktadır.

Yapay sinir ağları insan beyni gibi çalışan makina yapma isteğinin bir sonucu olarak ilgi çekmektedir. Yine bu amaçla geliştirilen sayısal bilgisayarlar insanın beyinsel yeteneğinin en zayıf olduğu çarpma, bölme gibi matematiksel ve algoritmik hesaplama işlemlerinde hız ve doğruluk açısından yüzlerce kat başarılı olmalarına rağmen insan beyninin öğrenme ve tanıma gibi işlevlerini hala yeteri kadar gerçekleştirememektedirler.

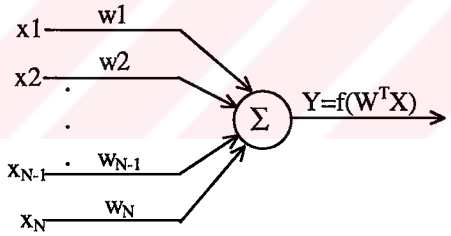
Canlılarda yaşamsal ve düşünsel faaliyetleri yürüten en küçük birim sinir hücresidir (şekil 1-1). Sinir sistemi birbiri ile etkileşen sinir hücrelerinden oluşmuştur. Sinir sistemi aktivitelerinin çoğu duyu reseptörlerinden gelen duysal bilgiler ile başlar. Alınan bu duysal bilgiler, ani yada depolanarak gelecekteki reaksiyonlara neden olur.

Bir sinir hücresinin giriş birimine Dendrit, çıkış birimine ise Akson denir. Hücrelerinin birbirlerine bağlantı yeri olan sinapslarda sinyal iletisinin olduğu bölgelerdir. Her sinir hücresi uyarı için bir eşik gerilim değerine sahiptir. Hücre girişlerinin toplamı bu eşik değerinin aşılmasını sağlar ise hücre çıkışı aktif olur. Sinir hücresinin çok sayıdaki girişlerinden hücre potansiyelini arttıracak yönde olanlar uyarıcı, azaltacak yönde olanlar bastırıcı adını alırlar [1].



Şekil 1-1

Sinir hücresinin giriş çıkış fonksiyonunun deneysel bilgilere dayanılarak modellenmesi ile pek çok yapay sinir ağı modeli ortaya konmuştur. En genel olarak sinir hücresi, girişlerinin ağırlıklı toplamının lineer olmayan bir fonksiyonunu çıkışında veren bir işlem elemanı olarak modellenebilir (Şekil 1-2).



Şekil 1-2

Yapay sinir hücresi, doğal sinir hücrelerinden esinlenen modellere dayanmaktadır. Doğal sinir hücreleri işlevlerine göre çok değişik özellikler taşımaktadır. Görevlerde ortaya çıkan bu farklılık, hücre yapısına ve çalışma ilkesine de yansıyarak farklı yapılar oluşmasına neden olmuştur.

Doğal sinir hücrelerinin ortak yönlerinden biri, bir sinir hücresinin yaptığı işin, sistemin görevinin yalnızca küçük bir bölümü olmasıdır. Bu nedenle görevin türüne ve işlenecek bilginin büyüklüğüne bağlı olarak değişik sayıda sinir hücresi birlikte çalışırlar. Birlikte çalışmak için ise yoğun bir bağlantı ağı oluşmuştur.

Hücreler arası bağlantıdaki ağırlık katsayıları, yapay sinir ağlarının en önemli bölümüdür. Yapay sinir ağlarının belirli bir işlevi yerine getirebilmesi için yapacağı işe uygun bir eğitim sürecinden geçmesi gerekir.

Günümüzde bir yandan var olan yapay sinir ağı modelleri geliştirilirken, bir yandanda yeni modeller ortaya konmaktadır. Genel olarak yapay sinir ağları üç sınıfa ayrılabilir:

İleribeslemeli (feedforward) ağlar : Giriş işaretlerinin bir dönüşümünü alırlar. Dönüşüm şekli ağ parametreleri dışardan ayarlanarak belirlenir [2].

Geribeslemeli (recurrent) ağlar : Dinamiği olan bu ağlarda giriş işaretleri ve ağın başlangıç durumuna göre ağ bir süre sonra bir son duruma yakınsar [2].

Yarışan eğitimsiz özörgütlenmeli (self organizing) ağlar : Komşu hücreler karşılıklı etkileşimde bulunarlar ve ağ giriş işaretlerine göre kendilerini değiştirir [2].

Öğrenme işlemi bağlantı ağırlıklarının belirli bir kurala göre değiştirilmesine dayanır.

Eğiticili öğrenme: Ağa giriş ve istenen çıkış değerleri verilerek ağın bu çiftler arasındaki ilişkiyi öğrenmesi ağlanır [2].

Eğiticisiz öğrenme: Ağa yalnızca giriş bilgileri verilir ve bu bilgilere göre ağın yapısını kendi kendine değiştirmesi sağlanır [2].

En karmaşık sinir ağı olan Cerebral Cortex'de yani insan beyninde 10^{10} sinir hücresi ve her hücre için 10^4 bağlantı mevcuttur. Bu mertebede bir ağı yapay olarak gerçeklemek bugünkü teknoloji ile imkansızdır. Buna karşılık günümüzde bilim adamları kendi problemlerine uygun yapay sinir ağı modellerini kullanmakta ve geliştirmektedirler.

BÖLÜM 2. CMAC (Cerebellar Model Arithmetic Computer)

2.1. CMAC'ın Dayandığı Nörolojik Model

Beyinde görme korteksinin hemen altında bulunan beyincik (cerebellum) [1] insanda el, kol gibi uzuvların ve gözlerin seri, koordineli hareketlerini kontrol eden bir organdır. Beyincikte meydana gelen herhangi bir hasar nesnelere tam ulaşamamaya, koordinasyon eksikliğine ve gözlerin cisimleri takip edememesine yol açar.

Beyinciğe giriş olarak komut girişleri ve geri besleme girişlerinin toplamı gelmektedir. Komut girişleri YAKLAŞ, TUT veya BIRAK gibi hareket komutları ile hareketin hızı, gerekli kuvvet veya hareketin son bulacağı nokta gibi bilgilerden oluşurken geri besleme bilgileri; eklemlerin konumu, kas gerginlileri veya derideki basınç gibi bilgileri içerir.

Beyne gelen bu girişler uygun hareketi oluşturan kas uyarım bilgilerinin bulunduğu yerlerin adresini oluşturur. Zaman içerisinde her an gelen girişlere karşılık düşen kas uyarım miktarları bu şekilde elde edilerek hareket yörüngesi üzerinde diğer noktaya ulaşabilmek için gerekli çıkış işaretleri oluşur.

N adet giriş değişkenin R farklı değer alabildiği bir durumda R^N olası farklı giriş sözkonusudur. Bu mertebede farklı girişe karşılık düşen çıkış bilgilerinin tamamının beyinde tutulamayacağı ve bu yüzden de gerekli eklem hareketlerinin oluşturulması problemini beynin bir çeşit bellek kullanım yöntemi ile çözdüğü kabul edilmektedir [6].

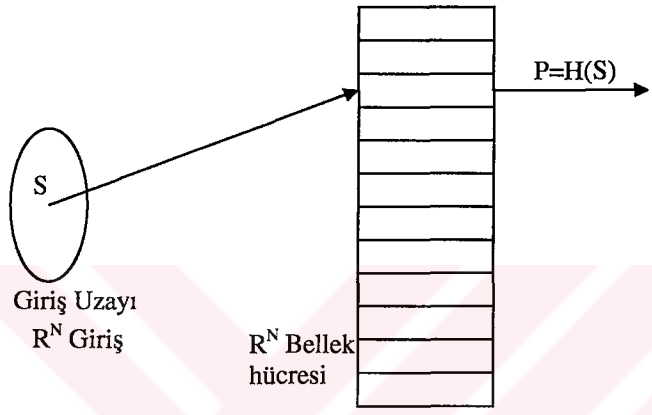
2.2. CMAC'ın Yapısı

CMAC ile ilgili ilk teoriler 1975 yılında J.S.ALBUS tarafından verilmiştir [3]. Burada CMAC çok serbestlik derecesine sahip robot kontrol

sistemlerinde eklemlerin hareketini, yoğun trigonometrik denklemlerin çözümü yerine bir tür Look-Up Table ile sağlamaktadır.

CMAC asıl olarak öğrenme yeteneğine sahip bir dağıtılmış Look-Up Table algoritmasıdır.

Normal Look-Up Table algoritmasında bir giriş vektörü tablodan belli bir hücreyi seçerek o hücrenin içeriğine karşılık düşer (şekil 2.2.1).



Şekil 2-2-1

Look-Up table algoritmalarının üstün tarafları şöyle sıralanabilir;

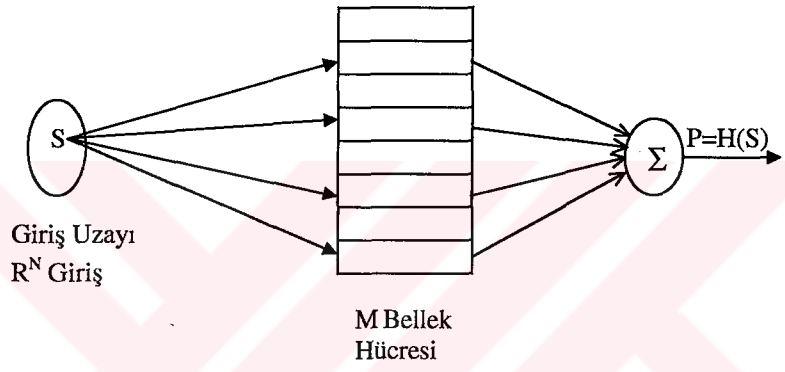
- Her şekilde fonksiyon öğrenilebilir çünkü fonksiyon önemli değildir, girişe karşılık düşen çıkış nümerik olarak tabloda tutulur.
- Basit adresleme teknikleri ile çok hızlı sonuç alınabilir.

Bu üstünlüklerine karşı bazı dezavantajlarıda vardır;

- Giriş işaretleri kuantalanmak zorundadır bu ise bir kuantalama gürültüsü meydana getirir. Azaltmak için kuantalama aralığı küçük seçilebilir fakat bu tablo boyunun artması demektir.
- Giriş sayısı arttıkça tablo boyu da üstel olarak artar. R farklı değer alabilen N değişken için tablo boyu R^N olur (şekil 2.2.1).
- Birbirine benzer girişlere karşı düşen çıkışlar arasında bir ilişki yoktur çünkü her çıkış tek ve farklı bir giriş vektörü tarafından oluşturulur. Oysa hızlı değişmeyen sürekli fonksiyonlarda benzer girişlere benzer çıkışlar karşılık düşmektedir. Bu bir avantaj olarak kullanılmamaktadır.

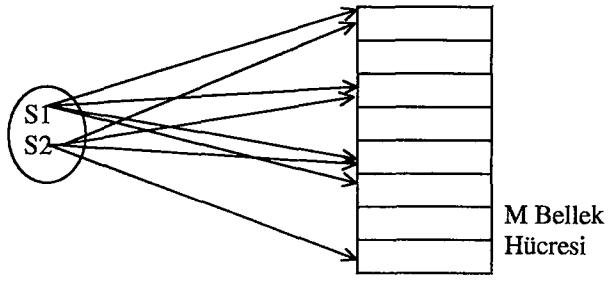
CMAC algoritması ise gelişmiş bir Look-Up table tekniği olarak düşünülebilir. Algoritmanın normal Look-Up table metodlarına göre üstünlükleri, dağıtılmış veri saklama ile genelleştirme özelliğidir.

Hızlı değişmeyen sürekli fonksiyonlarda birbirine yakın giriş vektörleri benzer çıkışlar üretirler. Dolayısıyla bu çıkışlar arasında bir ilişki mevcuttur. Cmac bu özelliği kullanarak belli bir girişe karşılık düşen çıkışı birden fazla bellek hücresinin toplamı olarak tutarak kullanır. Buna dağıtılmış veri saklama adı verilir. Şekil 2.2.2 de görüldüğü gibi bir giriş vektörü dört bellek hücresini seçmekte ve çıkış bu hücrelerin içeriklerinin toplamında oluşmaktadır.



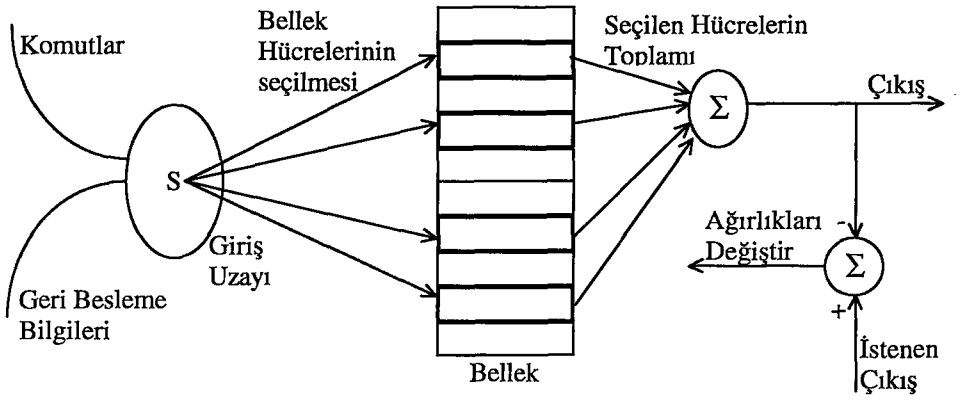
Şekil 2.2.2

Genelleştirme algoritması ile de, komşu giriş vektörlerinin seçtiği bellek hücreleri vektörler arası uzaklıkla ters orantılı olarak ortak tutulur. Giriş vektörleri birbirine nekadar benziyorsa ortak hücre sayısı da o oranda fazla olur. Şekil 2.2.3 de görüldüğü gibi birbirine benzer S1 ve S2 vektörlerinden S2, S1'in seçtiği dört hücrenin üç tanesini aynen seçmektedir. Sadece bir hücreleri farklıdır dolayısıyla oluşan çıkışlar sadece ortak olmayan hücre içeriği kadar farklı olacaktır.



Şekil 2.2.3

Genel olarak CMAC yapısı şu şekilde verilebilir (Şekil 2.2.4).



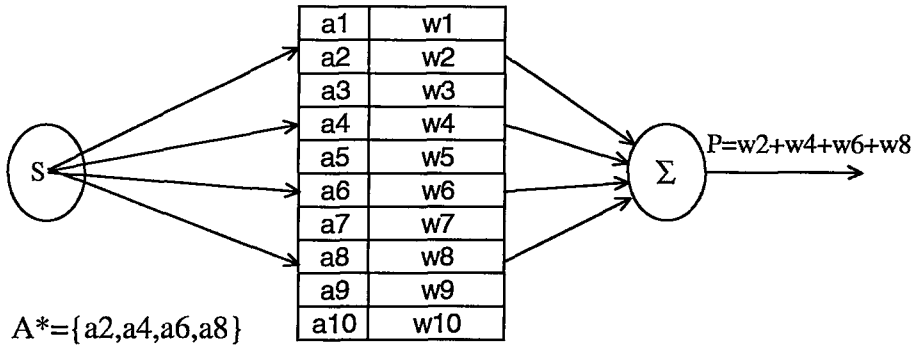
Şekil 2.2.4

CMAC algoritması bir dizi dönüşüm ve öğrenme algoritması olaral iki kısımda incelenebilir.

2.2.1 CMAC'da Dönüşümler

S : Giriş vektörü
A : Adres çözücü vektör
P : Çıkış vektörü olmak üzere

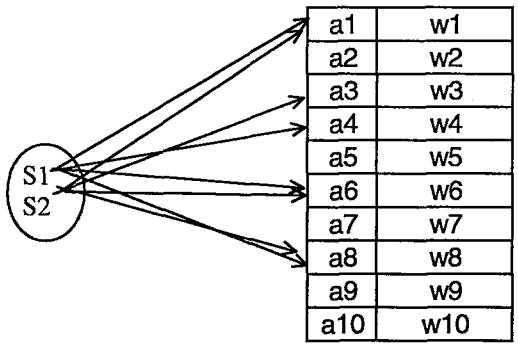
S->A->P şeklinde ardışıl iki dönüşüm mevcuttur.
Bir S giriş vektörü A adres kod çözücü vektörde A* diye tanımlanan elemanları sıfırdan farklı bir küme seçer (şekil 2.2.1.1).
Çıkış bu A* kümesinin seçtiği hücrelerin içeriklerinin toplamından oluşur.
Herhangibir S girişi için istenen çıkış değerinin elde edilebilemesi için de sadece A* kümesinin seçtiği bellek hücreleri değiştirilir.



Şekil 2.2.1.1

A* vektörünün eleman sayısı ρ, genelleme sayısı olarak tanımlanır. ρ=1 olması durumu normal Look-Up table algoritmasına denk düşer.

Şekil 2.2.1.2 da görüldüğü gibi S1 ve S2 vektörleri için istenen çıkış değerleri birbirine yakın ise A vektöründen birbiriyle kesişen A1* ve A2* gibi iki kümeyi seçerler. Bu durumda S1 veya S2 için yapılacak bir eğitim sonunda diğeri için bir eğitim yapılmadığı halde ortak elemanlardan dolayı isenilene yakın bir sonuç alınması sözkonusudur.
Burada bu fark sadece w3 ve w4 arasındaki fark kadar olacaktır.



A1*={a1,a4,a6,a8}
A2*={a1,a3,a6,a8}
A1*∧A2*={a1,a6,a8}

Şekil 2.2.1.2

Bu özelliğe genelleştirme (generalization) adı verilir. Canlılardaki bir öğrenme deneyiminden benzer bir durum için genelleme yapma yeteneğine denk düşer.

Fakat S2 giriş vektörünün oluşturması istenen çıkış S1'inkine göre çok farklı ise bu sefer bu kesişim zorluk çıkaracaktır. A2*'ın seçtiği ağırlıklar S2'ye göre değiştirilir ise bu durumda S1 için çıkışta bir bozulma meydana gelecektir. Bu olaya da öğrenme karışması (learning interference) adı verilir.

Bu durumdan kaçınmak için A1* ile A2*'ın eleman sayıları fazla, buna karşılık ortak elemanları diğer bir değişle genelleştirmenin az olması gerekmektedir.

CMAC Dönüşüm Algoritması:

S->A dönüşümü S->M ve M->A olmak üzere iki ardışıl dönüşüme ayrılır.

Tablo 2.2.1.1

S	Aa	Ab	Ac	Ad	Ae	Af	Ag	Ah	Ai	Aj	Ak	Al
1	1	1	1	1	0	0	0	0	0	0	0	0
2	0	1	1	1	1	0	0	0	0	0	0	0
3	0	0	1	1	1	1	0	0	0	0	0	0
4	0	0	0	1	1	1	1	0	0	0	0	0
5	0	0	0	0	1	1	1	1	0	0	0	0
6	0	0	0	0	0	1	1	1	1	0	0	0
7	0	0	0	0	0	0	1	1	1	1	0	0
8	0	0	0	0	0	0	0	1	1	1	1	0
9	0	0	0	0	0	0	0	0	1	1	1	1

Tablo 2.2.1.1 de görüldüğü gibi her giriş vektörüne '1' ler ve '0' lardan oluşan bir m vektörü karşılık düşürülür. Tek değişkenli bir durumda S->M dönüşümünün yapılması S->A dönüşümüne denktir. M->A dönüşümü giriş değişken sayısı iki veya daha fazla olduğu zaman gerekmektedir.

Bu örnekte ρ=|m*|=|A*|=4 olarak seçilmiştir. Tablo 2.2.1.2 de ise S->M dönüşümünün kısaltılmış olarak ifade edilmiş hali görülmektedir.

Tablo 2.2.1.2

S	m*
1	a,b,c,d
2	e,b,c,d
3	e,f,c,d
4	e,f,g,d
5	e,f,g,h
6	i,f,g,h
7	i,j,g,h
8	i,j,k,h
9	i,j,k,l

s1=1 ve s2=3 vektörleri için

A1*={a,b,c,d}

A2*={c,d,e,f}

ve A1*ΛA2*={c,d} dır. Görüldüğü gibi s1 ve s2 girişleri için seçilen bellek gözlerinden iki tanesi ortaktır.

s1=1 , s2=5 için ise hiç ortak bellek gözü sözkonusu değildir.

Çok değişkenli durumda ise her değişken için önce benzer şekilde S->M dönüşümü yapılır.

S=(s1,s2) olduğu iki değişkenli durumda

0<s1<6 , 0<s2<8 ve ρ=|m*|=|A*|=4 olduğunu varsayalım.

Tablo 2.2.1.3 de görüldüğü gibi s1->m1* ve s1->m2* dönüşümleri gerçekleştirilir. Burada;

s1=2 ve s2=4 girişi için

m1*={E,B,C,D} ve m2*={e,f,g,d} dır.

Daha sonra m1* ve m2* birleştirilerek A* elde edilir.

A*={Ee,Bf,Cg,Dd}

Tablo 2.2.1.3

S1	m1*
1	A,B,C,D
2	E,B,C,D
3	E,F,C,D
4	E,F,G,D
5	E,F,G,H

S2	m2*
1	a,b,c,d
2	e,b,c,d
3	e,f,c,d
4	e,f,g,d
5	e,f,g,h
6	i,f,g,h
7	i,j,g,h

A* 'daki her eleman bir bellek hücresine karşılık düşmektedir. Bütün S değerleri için A* tablo 2.2.1.4 de görülmektedir.

Tablo 2.2.1.4

	S2						
i,j,g,h	7	Ai,Bj,Cg,Dh	Ei,Bj,Cg,Dh	Ei,Fj,Cg,Dh	Ei,Fj,Gg,Dh	Ei,Fj,Gg,Hh	
i,f,g,h	6	Ai,Bf,Cg,Dh	Ei,Bf,Cg,Dh	Ei,Ff,Cg,Dh	Ei,Ff,Gg,Dh	Ei,Ff,Gg,Hh	
e,f,g,h	5	Ae,Bf,CG,Dh	Ee,Bf,Cg,Dh	Ee,Ff,Cg,Dh	Ee,Ff,Gg,Dh	Ee,Ff,Gg,Hh	
e,f,g,d	4	Ae,Bf,Cg,Dd	Ee,Bf,Cg,Dd	Ee,Ff,Cg,Dd	Ee,Ff,Gg,Dd	Ee,Ff,Gg,Hd	
e,f,c,d	3	Ae,Bf,Cc,Dd	Ee,Bf,Cc,Dd	Ee,Ff,Cc,Dd	Ee,Ff,Gc,Dd	Ee,Ff,Gc,Hd	
e,b,c,d	2	Ae,Bb,Cc,Dd	Ee,Bb,Cc,Dd	Ee,Fb,Cc,Dd	Ee,Fb,Gc,Dd	Ee,Fb,Gc,Hd	
a,b,c,d	1	Aa,Bb,Cc,Dd	Ea,Bb,Cc,Dd	Ea,Fb,Cc,Dd	Ea,Fb,Gc,Dd	Ea,Fb,Gc,Hd	
		1	2	3	4	5	S1
		A,B,C,D	E,B,C,D	E,F,C,D	E,F,G,D	E,F,G,H	

Tablo 2.2.1.5 de ise s1=(2,3) ile s2=(i,j) vektörleri arasındaki |A1*ΛA2*| miktarı görülmektedir.

Tablo 2.2.1.5

S2						
7	0	0	0	0	0	
6	1	1	0	0	0	
5	1	2	1	1	1	
4	2	3	2	2	1	
3	3	4	3	2	1	
2	2	3	3	2	1	
1	2	2	2	1	0	
	1	2	3	4	5	S1

2.2.2 CMAC 'da Öğrenme

H fonksiyonunun CMAC'ın öğrenmesini istediğimiz fonksiyon olduğunu varsayalım. Bu durumda P=H(S) giriş uzayındaki her nokta için çıkış vektörünün istenen değeridir.

Giriş uzayından bir giriş seçilir ve buna H fonksiyonu uygulanarak istenen çıkış değeri Pd elde edilir. Bu giriş CMAC'a da uygulanır ve bu girişe o anki cevap P değeri bulunur.

$|Pd-P| \leq \zeta$ ise CMAC tablo herhangibir değişiklik yapılmaz.
Burada ζ kabul edilen maximum hata miktarıdır.

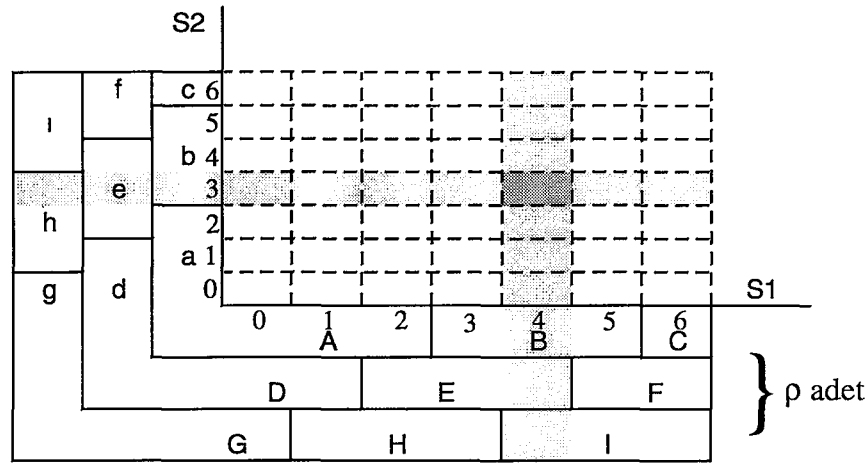
Eğer $|Pd-P| > \zeta$ ise ;

$\Delta = g\frac{Pd-P}{|A^*|}$ değeri toplamı o çıkışı oluşturan hücrelerin her birine eklenir.

g : Öğrenme oranıdır. Eğer $0<g<1$ seçilirse istenen değere yavaş yavaş yaklaşılır. $g=1$ olma durumu ise bir defada (one-shot) öğrenmedir . Tek bir adımda hata düzeltilir.

2 Değişkenli bir durum için dönüşümleri ve öğrenme algoritmasını gösteren bir örnek ;

$0 \leq s1 < 7$, $0 \leq s2 < 7$ ve $\rho=3$ için;



Şekil 2.2.2.1

Giriş uzayıeksenleri şekil 2.2.2.1 de görüldüğü gibi ρ adet katmandan oluşur. İlk katman ρ uzunluğundaki bloklara ayrılır, diğer katmanlar ise bir öncekine göre bir birim ötelenir [4].

ρ=3 olduğundan bir giriş vektörünü adreslemek için 3 bellek hücresinin seçilmesi gerekmektedir.

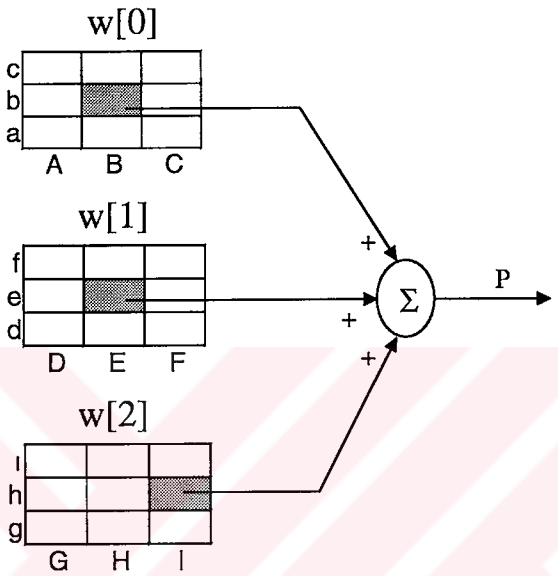
S=(4,3) girişi için;
1.katmanda w_{Bb}
2.katmanda w_{Ee}
3.katmanda w_{Ih}

ağırlıkları seçilerek $w_{Bb}+w_{Ee}+w_{Ih}$ değeri çıkışı oluşturur.

Burada;
m1*={B,E,I}
m2*={b,e,h}
A*={Bb,Ee,Ih} olmaktadır.
Bilgisayarda bu işlemi gerçeklemek amacıyla her katman için 3x3 matrisler kullanmak adresleme işlemlerini kolaylıkla yapmayı sağlamaktadır.

Normal Look-Up Table yönteminde 7x7=49 adet bellek gözü gerekirken bu örnekte toplam olarak w[3][3][3] şeklinde tanımlanmış 27 adet bellek hücresi kullanılmakta ve %44 bellek tasarrufu sağlanmaktadır.

Tanımlanan 3 adet w[3][3] matrisleri üzerinde S=(4,3) girişinin seçtiği hücreleri şu şekilde gösterebiliriz;



Şekil 2.2.2.2

Herhangibir S(s1,s2) girişine karşılık düşen çıkış şu şekilde hesaplanır;

$$\sum_{i=0}^{\rho-1} w[i][X][Y] \qquad (2.2.2.1)$$

$X , Y \in N$

$$X=[(s1+i)/\rho] \qquad (2.2.2.2)$$

$$Y=[(s2+i)/\rho] \qquad (2.2.2.3)$$

c= [d] fonksiyonu tam değer fonksiyonudur.

S=(4,3) girişi için bu şekilde bir hesaplama yaparsak;

i=0 için w[0][1][1]
i=1 için w[1][1][1]
i=2 için w[2][2][1] değerleri elde edilir.

w[0] da 1,1 hücresini ->Bb
w[1] da 1,1 hücresini ->Ee
w[2] de 2,1 hücresini yani Ih'ı seçmiş oluruz (Şekil 2.2.2.2).

Öğretilmek istenen fonksiyon P=s1+s2 olsun.

Başlangıçta tüm CMAC tablo yani w'lar sıfır değerine sahiptir.
S=(4,3) noktası için tablodan $w_{Bb}+w_{Ee}+w_{Ih} = 0$ değeri elde edilecektir. Öğrenme algoritmasına göre;

$$\Delta=g\frac{Pd - P}{|A^*|} \quad \Delta=(7-0)/3 = 2.3333$$

$$w_{Bb} = w_{Bb} + \Delta = 2.3333$$
$$w_{Ee} = w_{Ee} + \Delta = 2.3333$$
$$w_{Ih} = w_{Ih} + \Delta = 2.3333$$

değerine ulaşacaktır dolayısıyla tablodan S=(4,3) noktasına baktığımızda P=7 doğru değeri elde edilebilecektir.

S=(4,3) noktasının yakınında bulunan S=(4,2) noktası için tablonun vereceği değere bakacak olursak ;

$$P= w_{Ba}+w_{Ee}+w_{Ih} = 0 + 2.3333 + 2.3333 = 4.6666$$

Bu nokta için Pd=6 iken hiç eğitim yapılmamasına rağmen P=4.6666 değeri elde edilmektedir.

Genelleştirme (generalization) özelliğinden dolayı her noktanın eğitilmesine gerek olmadığı burada görülmektedir.

S=(4,2) noktasını eğittiğimiz zaman
Δ=(6-4.6666)/3 = 0.4444

w_{Ba} = 0.4444
w_{Be} = 2.3333 + 0.4444 = 2.7777
w_{Ih} = 2.3333 + 0.4444 = 2.7777

S=(4,2) için P=6 ve
S=(4,3) için P=2.3333+2.7777+2.7777=7.88 olacaktır.

Buradan da görülmektedirki ikinci öğrenme birincide bir miktar bozulma meydana getirmektedir (learning interference).

2.2.2.1 ifadesi n giriş için en genel halde şu formu almaktadır;

$$P=\sum_{i=0}^{\rho-1} w[i][I_1][I_2][I_3][I_4].....[I_n] \qquad (2.2.2.4)$$

I_J (j=1,2,....,n) değerleri (2.2.2.2) ye benzer şekilde hesaplanır.

Bu durumda tanımlanacak W matrisi ise şu şekilde olmaktadır;

$$w[\rho][[\frac{(I_{1max}+\rho-1)}{\rho}]+1][[\frac{(I_{2max}+\rho-1)}{\rho}]+1][.].....[.] \qquad (2.2.2.5)$$

I_{jmax} j. girişin alabileceği max. değerdir.

2.2.2.1 ifadesinde X,Y$\in$N 'dır. Normalde giriş uzayı gerçel sayılardan oluştuğu için gerçel sayılardan doğal sayılara "R->N" bir dönüşüm gerekmektedir.

Bu amaçla giriş uzayı kuantalanarak 0 noktasına ötelenir. Girişin herhangibir değerinin karşılığı bu kuantalanmış ve ötelenmiş eksen üzerinde kaçıncı aralıkta bulunduğudur.

2.3 CMAC' da Bellek Gereksinimi;

Normal Look-Up Table metoduna göre CMAC çok daha az bellek hücresi kullanmaktadır.

R çözünürlük, N girişli bir durumda R^N bellek gözüne ihtiyaç vardır, fakat CMAC , ρ genelleştirme yani katman sayısı ve Q da her katmandaki blok sayısı olmak üzere en fazla $\rho x Q^N$ bellek hücresi kullanır. Bölüm 2-2-2' deki örnekte ρ=3 , N=2 ve Q=3 değerleri için $3x3^2 = 27$ bellek hücresi kullanılmıştır. Normal Look-Up Table algoritmasında ise $7^2 = 49$ adet bellek hücresi gerekecekti [4].

CMAC'ın kullanacağı bellek miktarı tam olarak şu şekilde hesaplanabilir;

$$M=\sum_{j=1}^{\rho}\prod_{i=1}^{N}\left(f\left(\frac{Ri-j}{\rho}\right)+1\right) \qquad (2.3.1)$$

ρ : genelleştirme sayısı (generalization number)
N : giriş sayısı
Ri : i.girişteki toplam aralık sayısı.

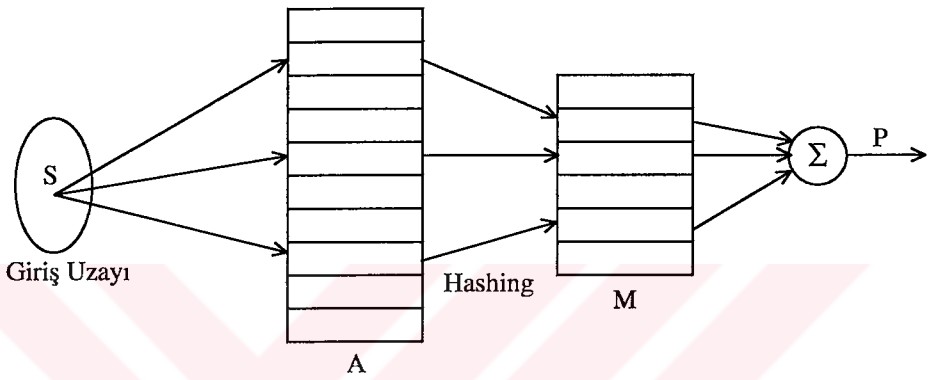
$$f(x)=\begin{cases} x & x-[x]=0 \\ [x]+1 & x-[x]>0 \end{cases}$$

Bazı durumlarda HASHING kodlama tekniği kullanarak daha az bellek alanı kullanılabilir.

Hashing, büyük ama seyrek olarak düzenlenmiş bellek alanının daha küçük bir alana yerleştirilmesini sağlayan ve oldukça çok kullanılan bir bellek adresleme tekniğidir [5].

Bu yöntem büyük adres alanındaki birden fazla adresin küçük alandaki aynı yeri gösterme olasılığının az olduğu durumlarda kullanılabilir.

Giriş vektörünün sınırları belirsiz ve dolayısıyla seyrek bir kuantalanmış giriş uzayının sözkonusu olduğu durumlarda CMAC hashing yöntemini fazla bir problem yaşamadan kullanabilir (Şekil 2.3.1).



Şekil 2.3.1

Normal Look-Up Table yönteminde herhangi iki girişin aynı bellek alanını göstermesi tamamen hatalı bir sonuca neden olur. Ancak CMAC'da çıkış birden fazla alanın toplamından oluştuğundan girişler aynı yeri gösterdiğinde çıkışta sadece ufak bir degişiklik olur. Burada sadece küçük bir gürültüden söz edilebilir. Bu aslında mevcut olan öğrenme karışması (learning interference) problemine benzerdir [3],[4].

2.4. CMAC'ın Özellikleri

- Giriş kuantalanmak zorundadır. Bu kuantalama gürültüsüne sebep olur. Azaltmak için kuantalama aralığı küçük seçilebilir fakat bu gerekli bellek miktarını arttırır. Ayrıca kuantalama genelleştirme uzaklığına etki eder. Genelleştirme uzaklığı iki giriş vektörü arasındaki ilişkinin sıfırlandığı mesafe olarak tanımlanır.

- Genelleştirme özelliğinden dolayı tüm girişler için eğitim yapılmasına gerek yoktur. Seçilen uygun girişler için eğitim yapılması büyük oranda yakınsamayı sağlar. Genelleştirmenin arttırılması gerekli bellek miktarını azaltacaktır fakat fonksiyonun durumuna göre hata artabilecektir.

- Hashing yöntemindeki hücre çakışması problemi biryere kadar gözardı edilerek yöntem kullanılabilir [3],[4].

- Eğitim yapılan bölge dışarısında doğru cevap veremez.

- Öğrenme karışması diğer bir değişle önceki öğrendiklerini bir miktar unutma sözkonusudur.

- CMAC'da yakınsama genelleştirme sayısı ρ ve fonksiyonun değişim şekline bağlıdır. Genelleştirmenin sözkonusu olduğu komşuluk içerisindeki girişler için fonksiyon yavaş değişiyorsa kabul edilebilir bir ζ hata miktarına sahip bir öğrenme işlemi gerçekleştirilebilir. Fakat bu komşuluk içerisinde fonksiyon hızlı değişiyor ise istenen doğrulukta bir sonuç alınması mümkün olmayacaktır. Bu yüzden yakınsama için fonksiyon değişim hızına göre bir genelleştirme yapılması gerekmektedir.

- Benzer işlevleri yerine getiren diğer yapay sinir ağları modelleri ile karşılaştırıldığına daha hızlı bir öğrenme ve cevap verme sözkonusudur. Örnek olarak ileri beslemeli ağlardaki geriye yayılım (Backpropagation) algoritmasında [2] eğitim sırasında tüm ağ bağlantı katsayıları değiştirilmekte CMAC'da ise sadece bir bölgede değişiklik yapılmaktadır. Hızlı cevap verme özelliğinden dolayı gerçek zaman uygulamalarında kullanılabilir. Yapılan hesaplamalar nispeten basittir. Bellek tüketimi ise daha fazladır.

3.CMAC Uygulamaları

3.1 Tek değişkenli fonksiyon öğrenme

CMAC yaklaşımının özelliklerini kolay bir şekilde inceleyebilmek için tek değişkenli fonksiyonlarda çeşitli genelleştirme sayısı ρ ve farklı eğitim noktaları simülasyonlar yapılmıştır.

Bunun için iki farklı fonksiyon seçilmiştir;

f1: $y=\sin(\Pi \bullet x/180)$

f2: $y=\begin{cases} 1 & \sin(\Pi \bullet x/90)>0 \\ 0 & \sin(\Pi \bullet x/90)<=0 \end{cases}$

x=0,....,359 , giriş kuanta aralığı 1 ve öğrenme oranı g=1 alınmıştır.

w[M] şeklindeki bellek tablosunun (cmac tablo) tüm hücreleri başlangıçta sıfır değerine sahiptir.

ρ=|A*|=30 için

$P(x)=\sum_{i=x}^{x+\rho-1} w[i] = \sum_{i=x}^{x+29} w[i]$ şeklindedir.

f1 fonksiyonu için, x=90 noktasında bir eğitim yapalım;

Pd(90)=1 ve başlangıçta P(90)=0 'dır.

$\Delta=\frac{Pd - P}{\rho} = (1-0)/30 = 1/30;$

$w[i]=w[i] + \Delta \quad (i=90,...,119)$

yapılır ve sonuçta P(90)=1 değeri elde edilir.

$P(100) = \sum_{i=100}^{129} w[i]$ değerine bakacak olursak,

i=100,...119 için w[i]=1/30 olduğundan P(100)=2/3 değerine bu nokta için eğitim yapılmadan erişmiş olur. Pd(100)=0.98'dir.

Bu sonuçlar şekil 3.1.1 de görülmektedir.

İkinci adımda x=270 noktası için eğitim yapıldığında elde edilen sonuçlar 3.1.2 'de ve,

$x=k \bullet 30 \quad (k=1,...12)$ noktaları için eğitim yapıldığında şekil 3.1.3 deki sonuçlar elde edilmektedir.

Her durum için ;

$$E=\sqrt{\frac{1}{360}\sum_{x=0}^{359}(Pd(x) - P(x))^2} \qquad (3.1.1)$$

şeklinde tanımlanan hata terimi ve maximun hata verilmiştir.

f2 fonksiyonunda da ρ=30 alınarak aynı noktalar için eğitim yapıldığında elde edilen sonuçlar şekil 3.1.4 , şekil 3.1.5 ve şekil 3.1.6 da görülmektedir.

Grafiklerden görüldüğü gibi genelleştirme özelliğinden dolayı eğitim yapılan noktalar civarında da bir miktar eğitim olmaktadır.

Genel olarak f1 ve f2 fonksiyonlarının öğretilmesi için eğitime x=90 noktasından başlanmıştır. Bir eğitim yapıldıktan sonra en fazla hata olan giriş tespit edilerek bir sonraki adımda bu noktanın eğitimi yapılmıştır.

Bu şekilde f1 için ρ=5, 20, 40 ve 60 değerleri için yaklaşık 3000 adım eğitimler gerçekleştirilmiş hata miktarının değişimi sırasıyla şekil 3.1.7, 3.1.8,

3.1.9 ve 3.1.10' da verilmiştir.

f2 için de ρ=2, 4, 10, ve 30 için hatanın değişimi sırasıyla şekil 3.1.11, 3.1.12, 3.1.13 ve 3.1.14 de verilmiştir.

Bu eğitimlerde E_{max} değerleri ise tablo 3.1.1'de verilmiştir.

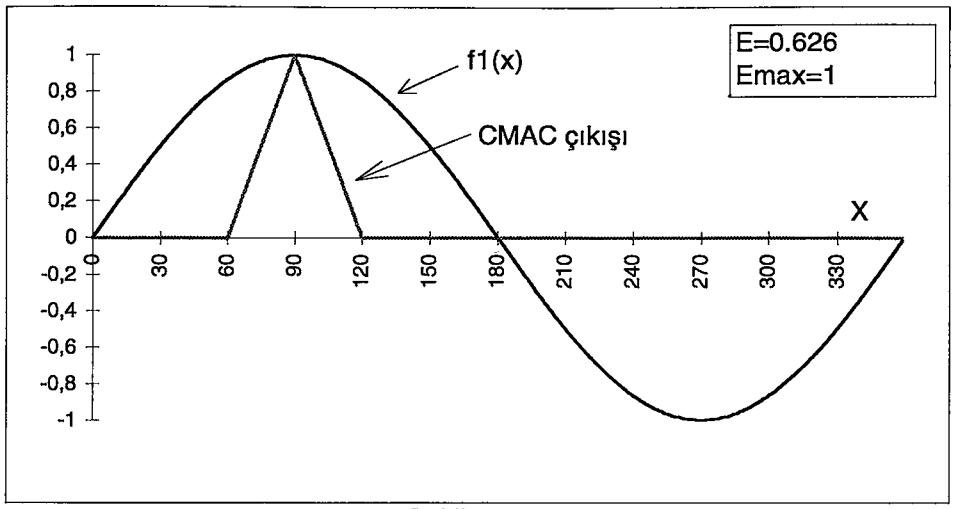
Tablo 3.1.1

f1 fonksiyonu için

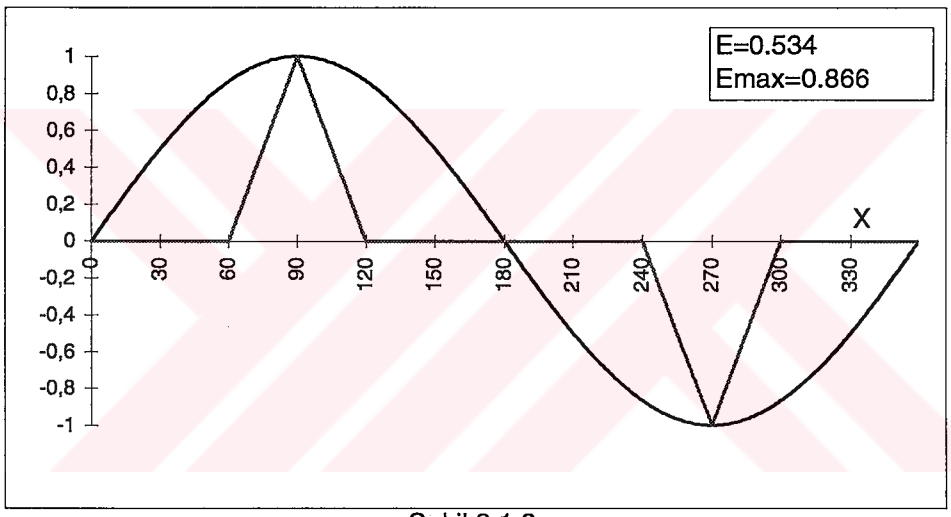
ρ	E_{max}
5	0.0007
20	0.0021
40	0.0006
60	0.0007

f2 fonksiyonu için

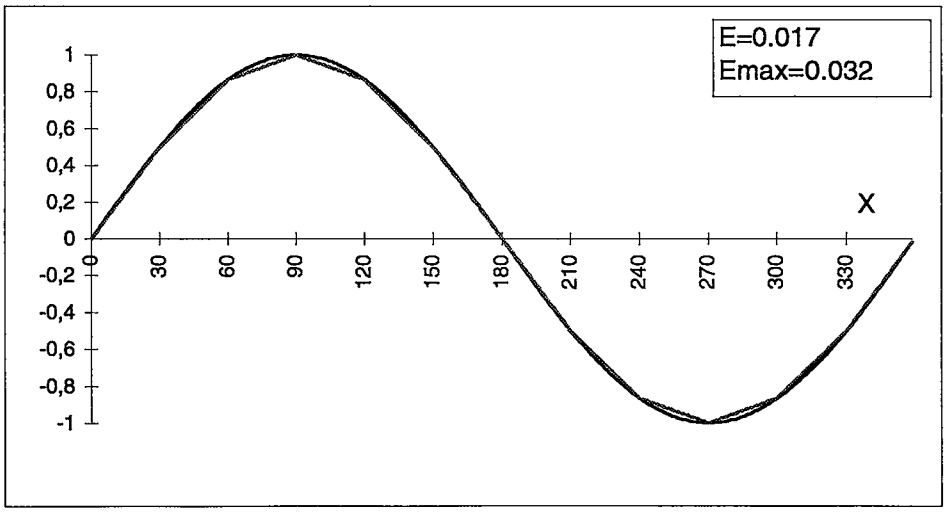
ρ	E_{max}
2	0.0328
4	0.0624
10	0.0754
30	0.0215



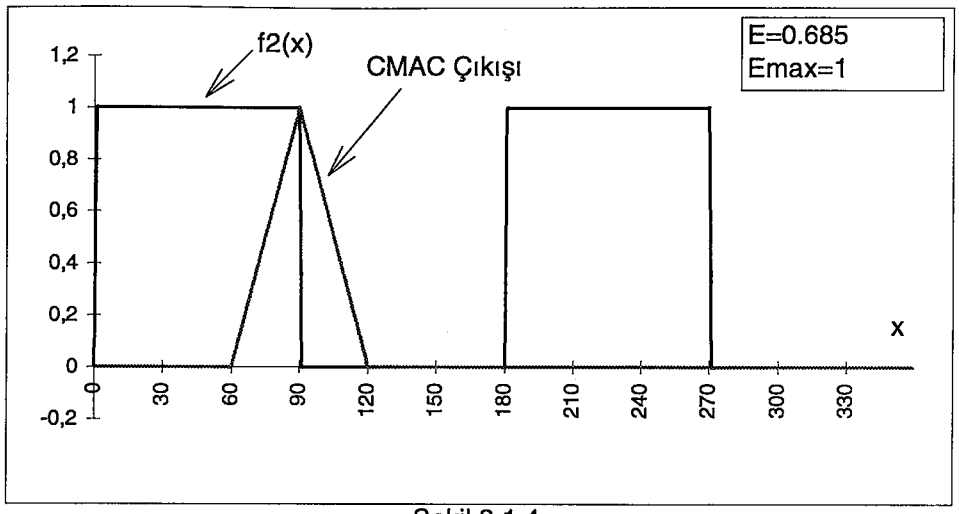
Şekil 3.1.1



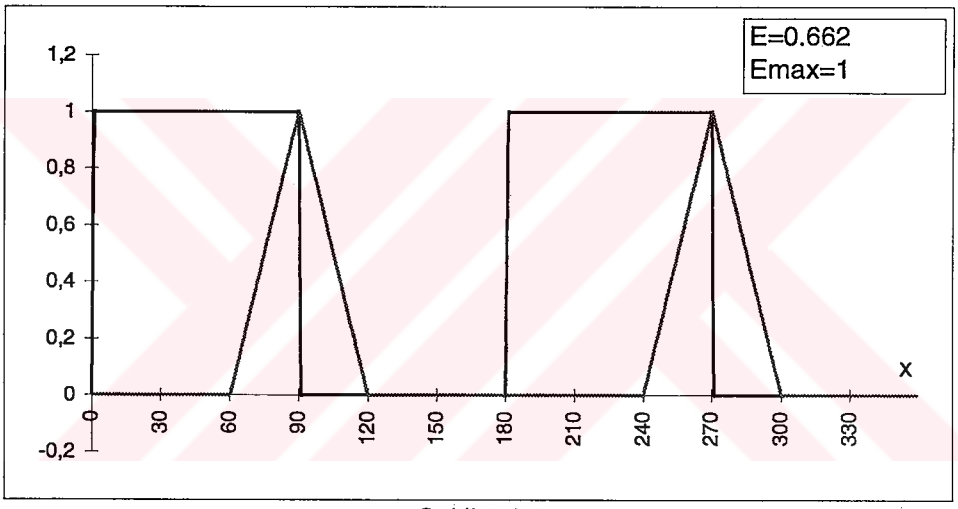
Şekil 3.1.2



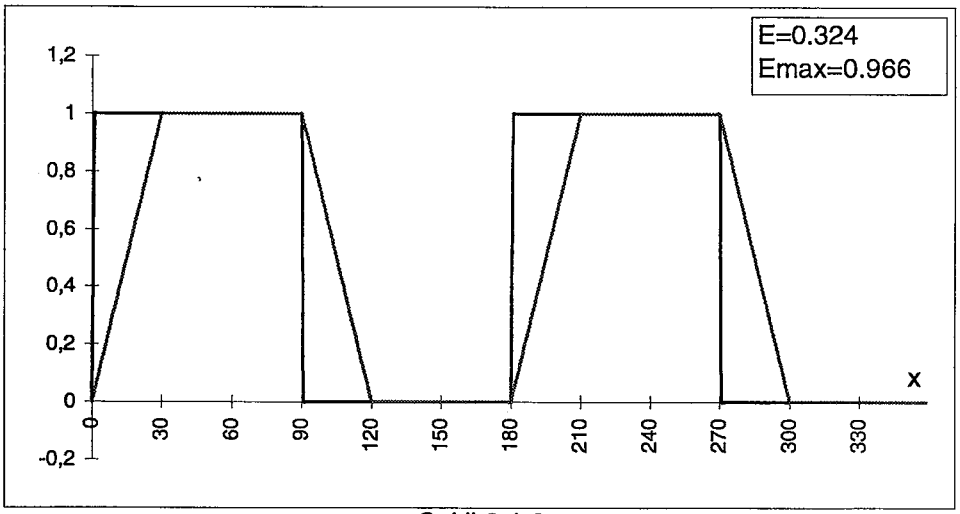
Şekil 3.1.3



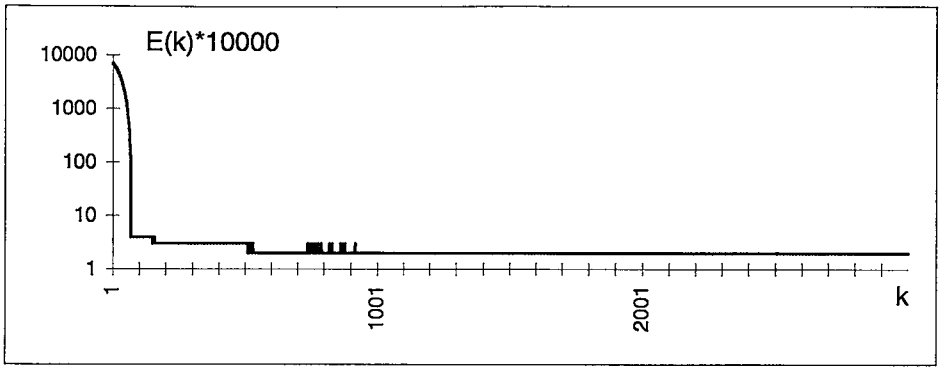
Şekil 3.1.4



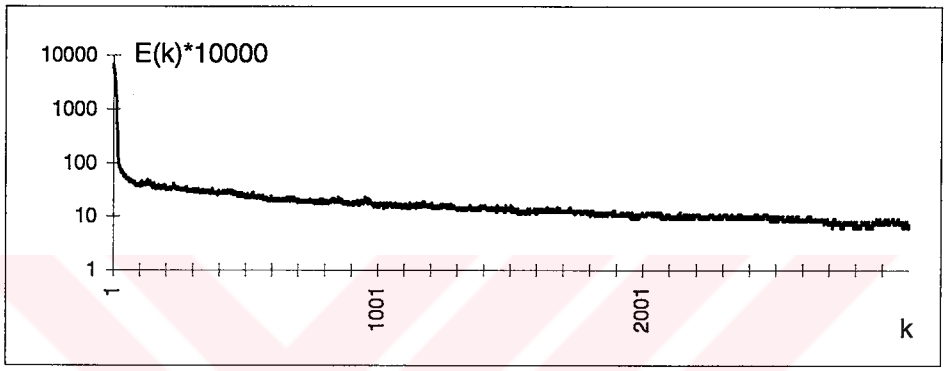
Şekil 3.1.5



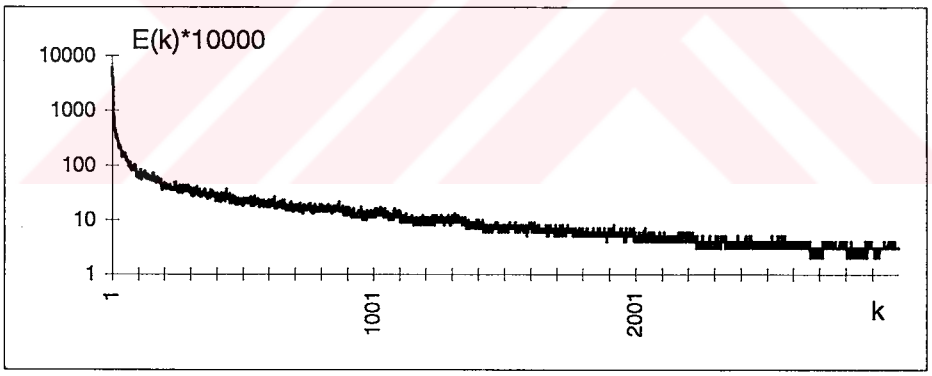
Şekil 3.1.6



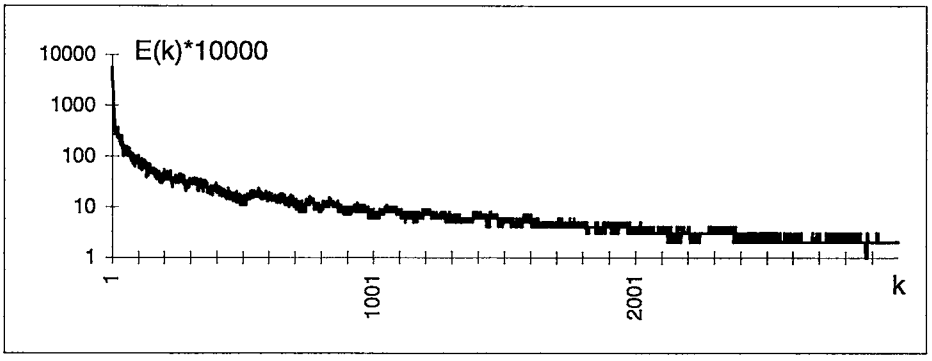
Şekil 3.1.7



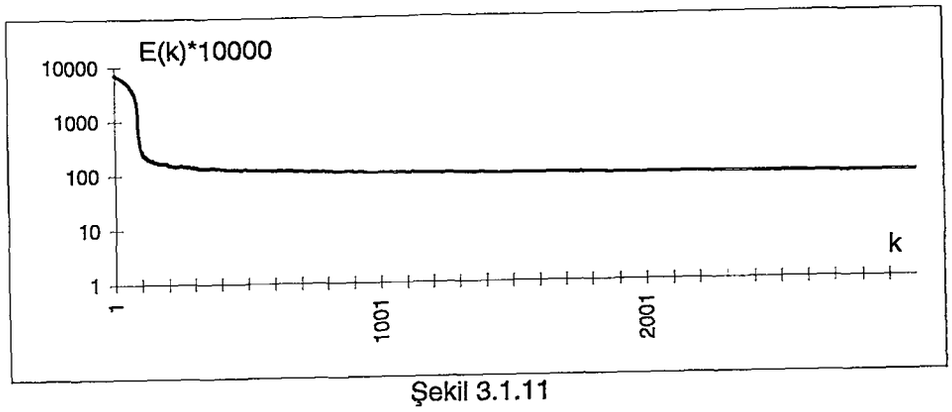
Şekil 3.1.8



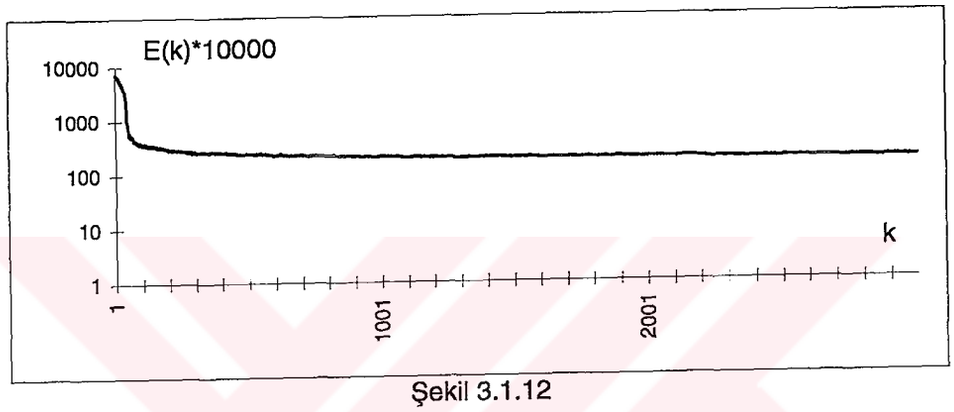
Şekil 3.1.9



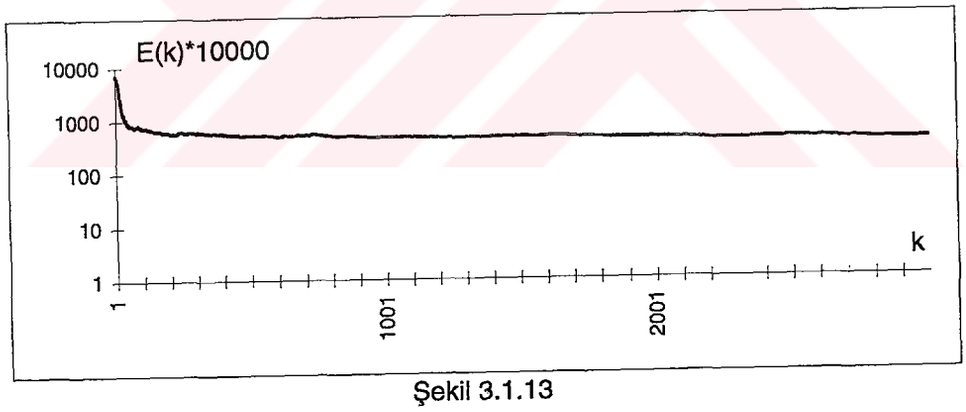
Şekil 3.1.10



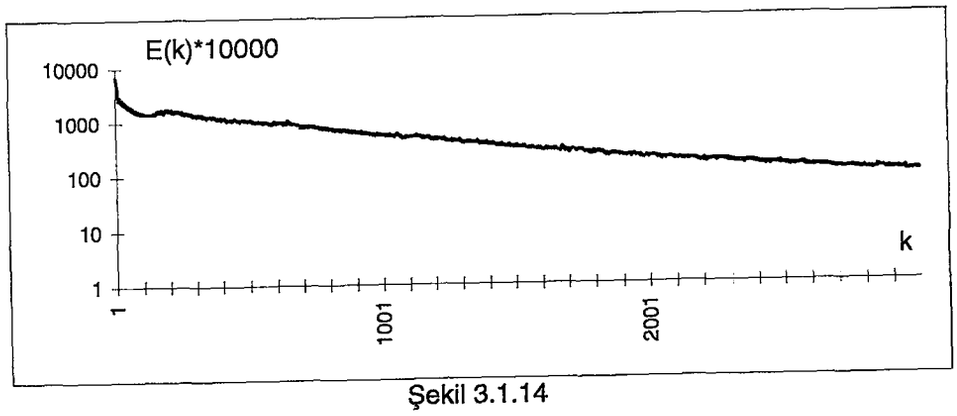
Şekil 3.1.11



Şekil 3.1.12



Şekil 3.1.13

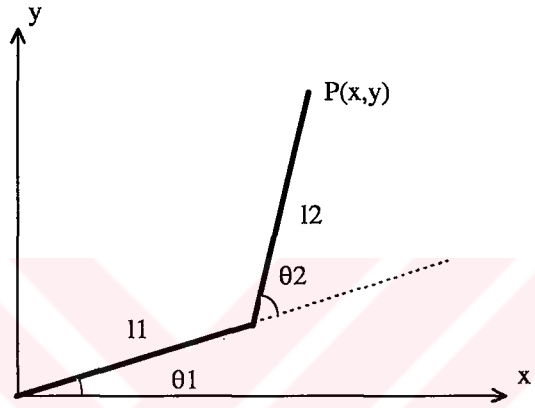


Şekil 3.1.14

3.2 Ters Kinematik denklemlerin çözümü

Bu amaçla şekil 3.2.1 ' de görülen iki serbestlik derecesine sahip bir robot manipulatör ele alınmıştır.

Burada istenen bir P(x,y) noktası için θ1 ve θ2 değerlerinin gerçek zamanda hesaplanması gerekmektedir.



Şekil 3.2.1

Yukarıdaki sistem için ;

$\theta 2=\cos^{-1}[(x^2+y^2-l1^2-l2^2)/2 \bullet l1 \bullet l2]$ ve

$\theta 1=\tan^{-1}(y/x) - \tan^{-1}[l2 \bullet \sin(\theta 2)/(l1+l2 \bullet \cos(\theta 2))]$

ifadeleri geçerlidir.

Yapılan simülasyonlarda l1=l2=5 ve giriş kuanta aralığı 1 alınarak;
θ2=f(x,y) ve θ1=f(x,y) için iki adet CMAC yapı kullanılmıştır.

Farklı genelleştirme sayısı ρ ve öğrenme oranı g için yapılan simülasyon sonuçları şu şekildedir;

ρ=4 , g=1 için 100 çevrim sonunda E1=0.103 ve E2=0.073 'e,

ρ=2 , g=1 için 100 çevrim sonunda E1=0.057 ve E2=0.020 'ye ve de

ρ=2 , g=0.1 için 200 çevrim sonumda E1=0.031 ve E2=0.011 'e yakınsamıştır.

E1 ve E2 hata terimleri 3.1.1' e benzer şekilde hesaplanmıştır.

Bazı (x,y) noktaları için olması gereken ve eğitim sonundaki CMAC çıkışları tablo 3.2.1'de verilmiştir.

Tablo 3.2.1

				ρ=4	g=1	ρ=2	g=1	ρ=2	g=0.1
x	y	θ1	θ2	Cmacθ1	Cmacθ2	Cmacθ1	Cmacθ2	Cmacθ1	Cmacθ2
5	5	0	1.570	0.057	1.665	0	1.579	0	1.575
6	6	0.227	1.115	0.301	0.952	0.234	1.099	0.236	1.098
1	6	0.488	1.833	0.422	1.906	0.468	1.849	0.478	1.836
0	8	0.927	1.287	0.987	1.296	0.933	1.291	0.931	1.289
				100 çevrim		100 çevrim		200 çevrim	

Sonuçlar incelendiğinde genelleştirme arttığı zaman hatanın arttığı, öğrenme oranı küçültüldüğünde eğitimin uzun süre aldığı görülmektedir. Öğrenme oranı büyük olduğunda hatanın artması, öğrenme karışmasının (learning interference) artmasından kaynaklanmaktadır.

0<=x<=10 ve 0<=y<=10 aralığında erişebilinecek nokta sayısı, yani x+y<=100 şartını sağlayan x,y çifti sayısı 90'dır. Normal look-up table kullanılsaydı 90 bellek hücresi gerekecekti oysa 2.3.1 'den;

ρ=2 için M=72

ρ=4 için M=50 bellek hücresi gerekir.

3.3 Sistem Belirleme

CMAC yardımıyla sistem belirleme işleminin gerçekleştirilmesi amacıyla aşağıda fark denklemi verilen sistem üzerinde incelemeler yapılmıştır.

$$y(k+1) = \frac{3}{4}\left[\frac{y(k)}{2+y^2(k)} + u^3(k)\right]$$

Burada y(k) ve u(k) CMAC'a giriş vektörünü, y(k+1) değeri de bu giriş vektörüne karşılık düşen çıkış değerini oluşturmaktadır.

Sisteme y(0)=0 başlangıç koşulundan başlıyarak [-1,+1] aralığında rastgele değişen u girişi uygulanmakta ve elde edilen çıkış değerine göre eğitim yapılmaktadır.

Yukarıda tanımlanan sistemin;

I-) ρ=4, giriş kuanta aralığı=0.02, g=0.9
II-) ρ=2, giriş kuanta aralığı=0.02, g=0.9
III-) ρ=2, giriş kuanta aralığı=0.04, g=0.9

şeklindeki üç farklı durum için aynı sayıda giriş vektörü ile eğitimi yapılmış ve daha sonra üç farklı test girişi asıl sisteme ve CMAC'a uygulanarak elde edilen sonuçlar verilmiştir.

Sistemin her durumu için 30000 adet rastgele giriş uygulanmıştır.

Test girişleri u1,u2 ve u3 sırasıyla şekil 3.3.1, 3.3.2 ve 3.3.3 'de, bu test girişlerine asıl sistemin verdiği cevaplar ise şekil 3.3.4, şekil 3.3.5 ve şekil 3.3.6'da görülmektedir.

I. durumda ;

Normal Look-Up table kullanılsaydı 10201 adet bellek hücresi gerekecekti oysa burada 2704 adet hücre kullanılmaktadır.

u1,u2 ve u3 test girişleri için 3.1.1'e benzer şekilde hesaplanmış hata terimleri;

E1=0.005, E2=0.006, E3=0.011 şeklinde elde edilmiştir. Bu girişlere karşı CMAC çıkışları ise şekil 3.3.7, 3.3.8 ve 3.3.9'da görülmektedir.

II. durumda ;

Normal Look-Up table kullanılsaydı 10201 adet bellek hücresi gerekecekti oysa burada 5202 adet hücre kullanılmaktadır.

u1,u2 ve u3 test girişleri için 3.1.1'e benzer şekilde hesaplanmış hata terimleri;

E1=0.005, E2=0.006, E3=0.019 şeklinde elde edilmiştir. Bu girişlere karşı CMAC çıkışları ise şekil 3.3.10, 3.3.11 ve 3.3.12'de görülmektedir.

III. durumda ;

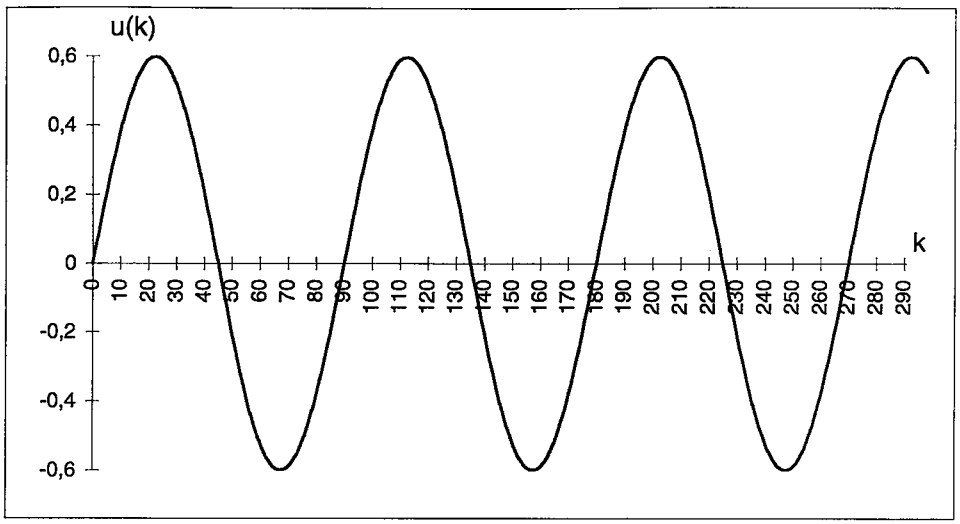
Normal Look-Up table kullanılsaydı 2601 adet bellek hücresi gerekecekti oysa burada 1352 adet hücre kullanılmaktadır.

u1,u2 ve u3 test girişleri için 3.1.1'e benzer şekilde hesaplanmış hata terimleri;

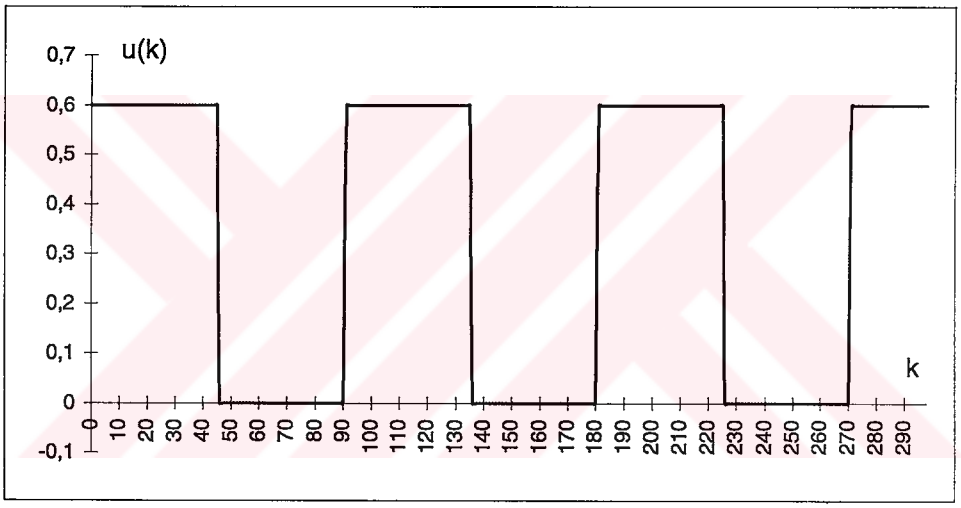
E1=0.009, E2=0.012, E3=0.016 şeklinde elde edilmiştir. Bu girişlere karşı CMAC çıkışları ise şekil 3.3.13, 3.3.14 ve 3.3.15'de görülmektedir.

Grafikler incelendiğinde asıl sisteme yakın çıkışlar elde edilebildiği görülmektedir.

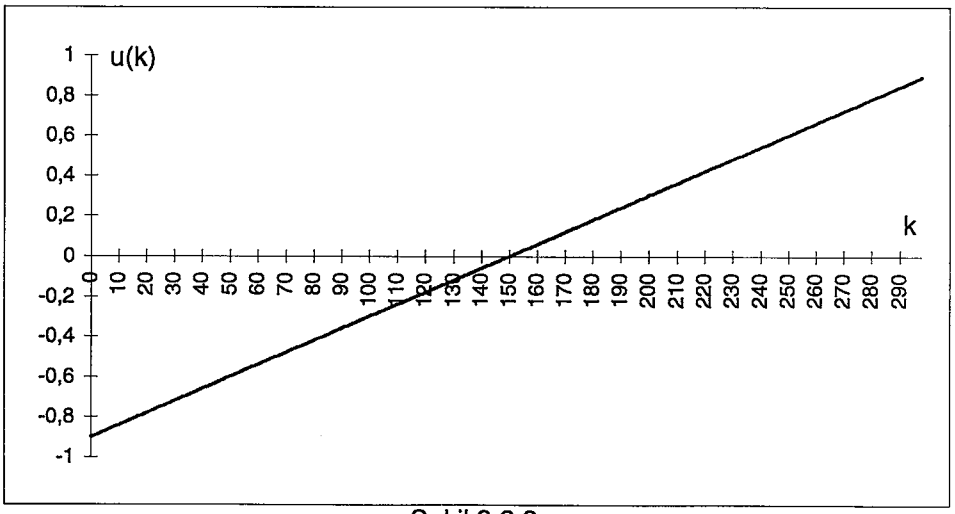
Performansın arttırılması için eğitimde kullanılan elemanların giriş uzayına düzgün dağılmış ve fazla sayıda olması gerekmektedir.



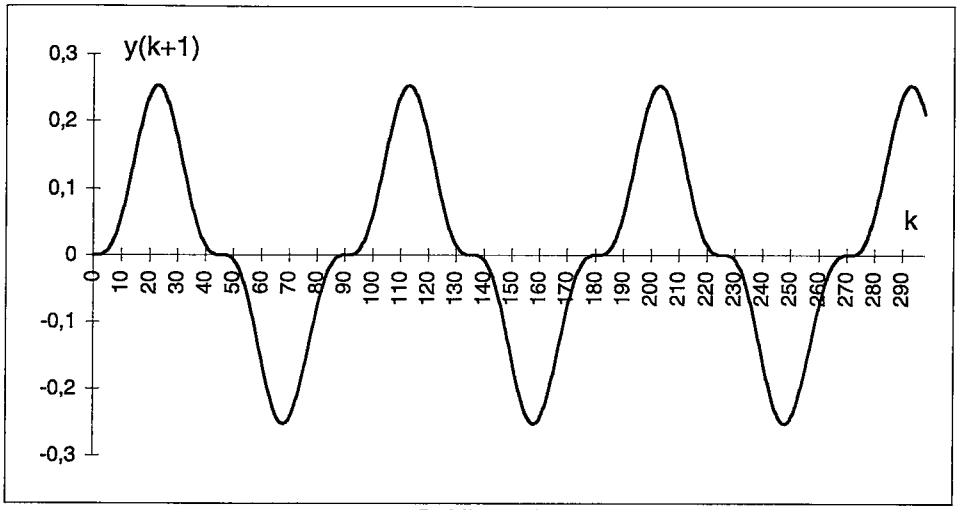
Şekil 3.3.1



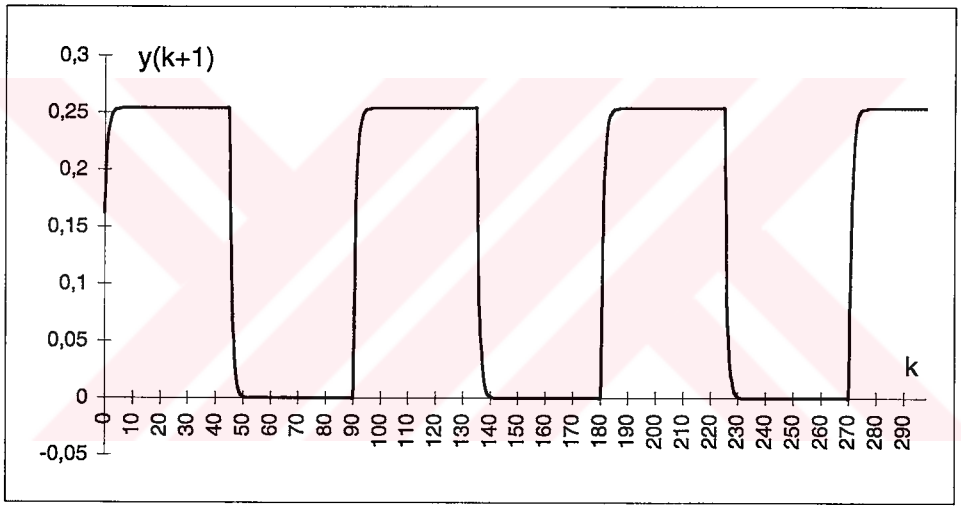
Şekil 3.3.2



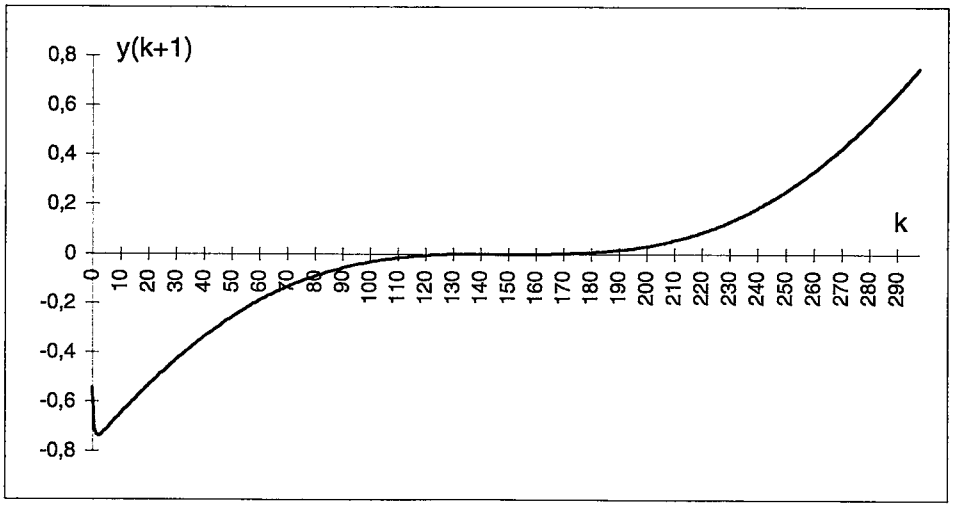
Şekil 3.3.3



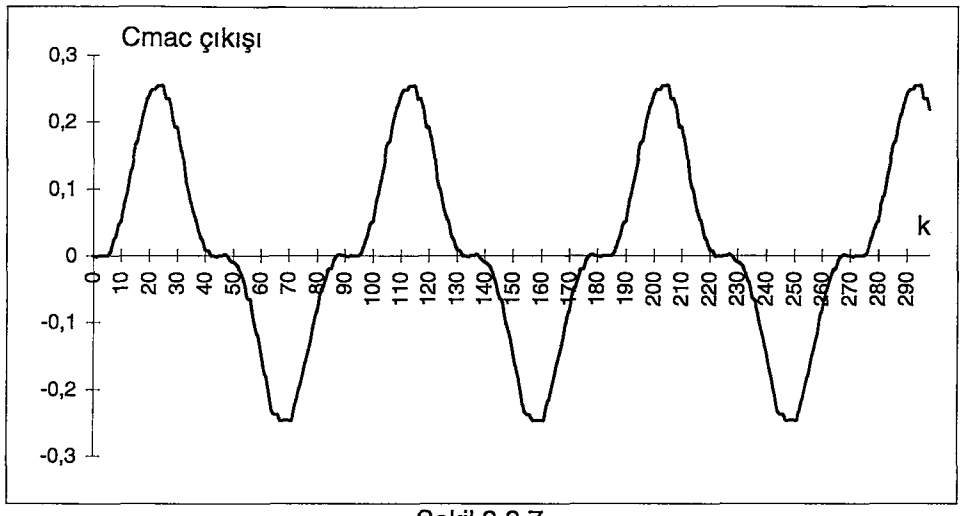
Şekil 3.3.4



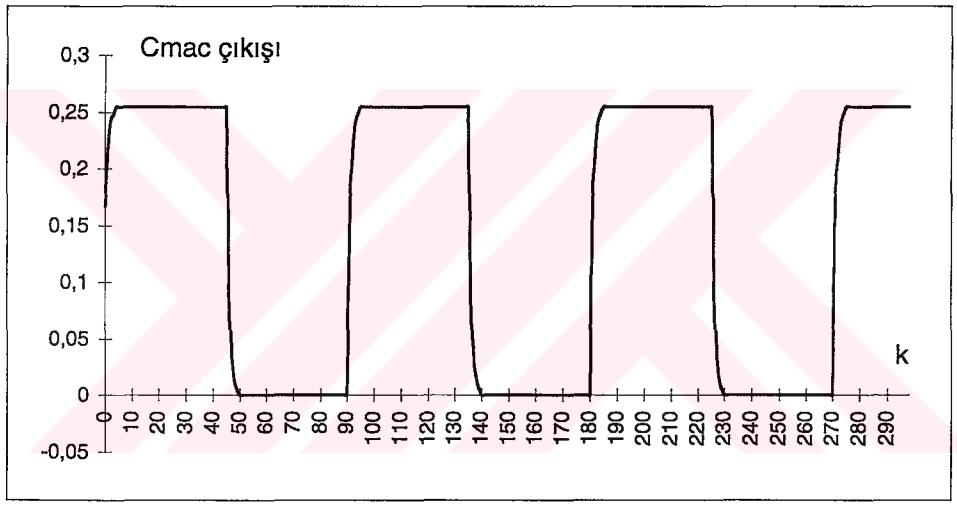
Şekil 3.3.5



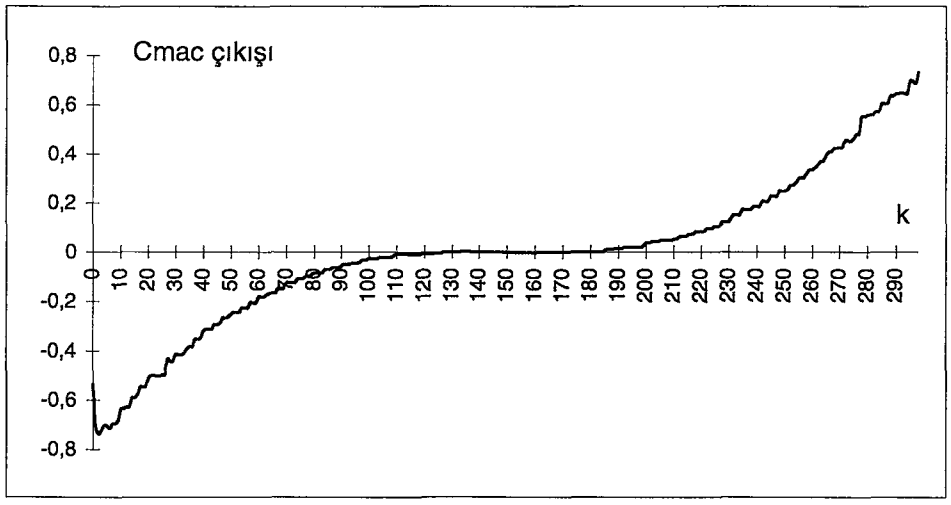
Şekil 3.3.6



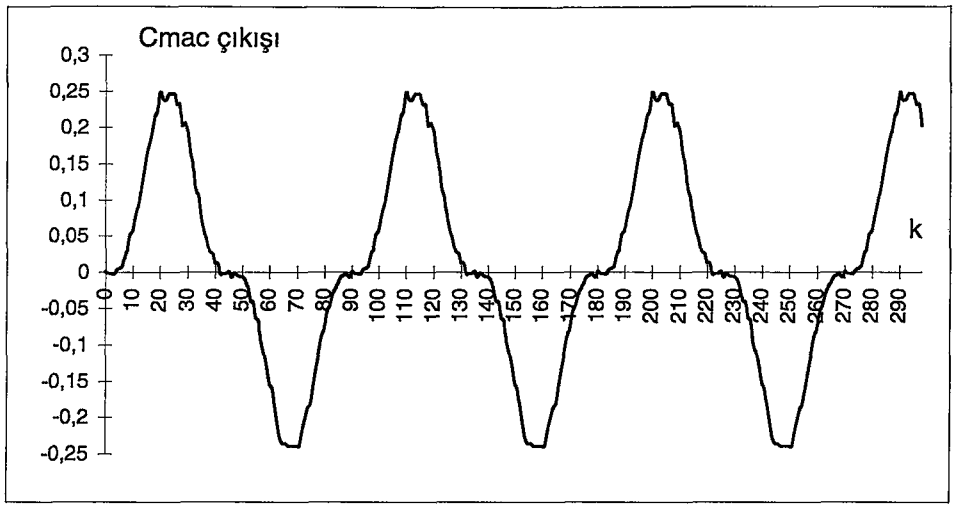
Şekil 3.3.7



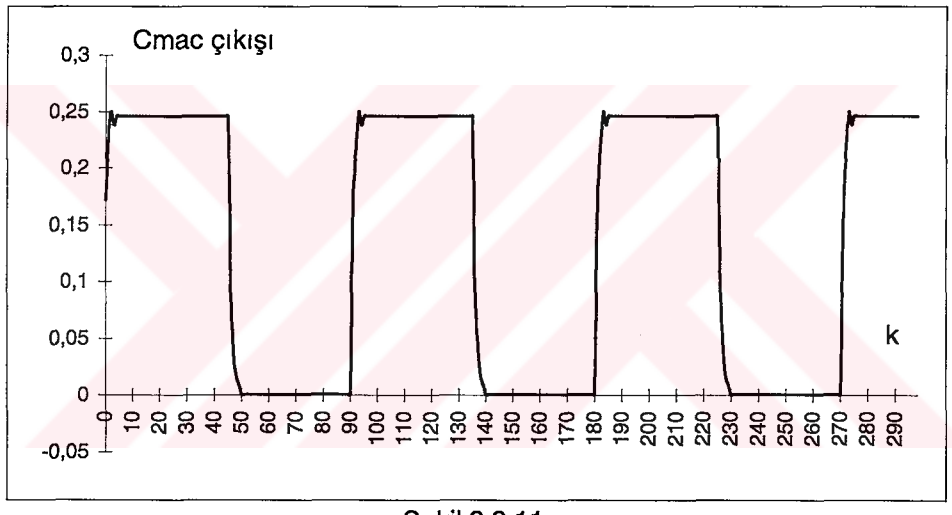
Şekil 3.3.8



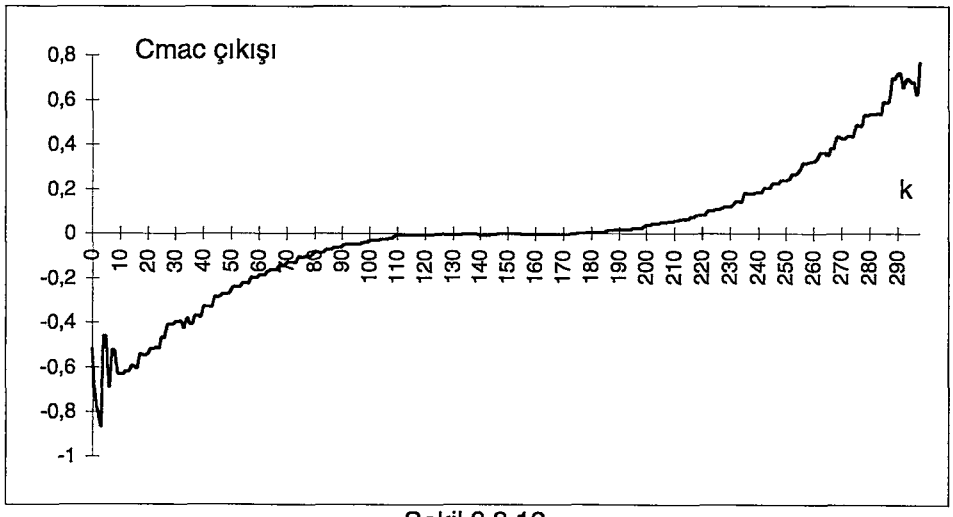
Şekil 3.3.9



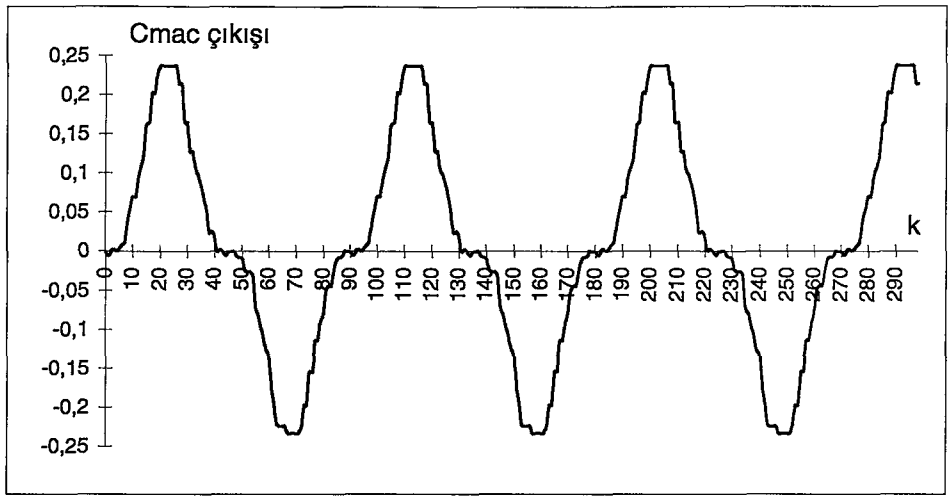
Şekil 3.3.10



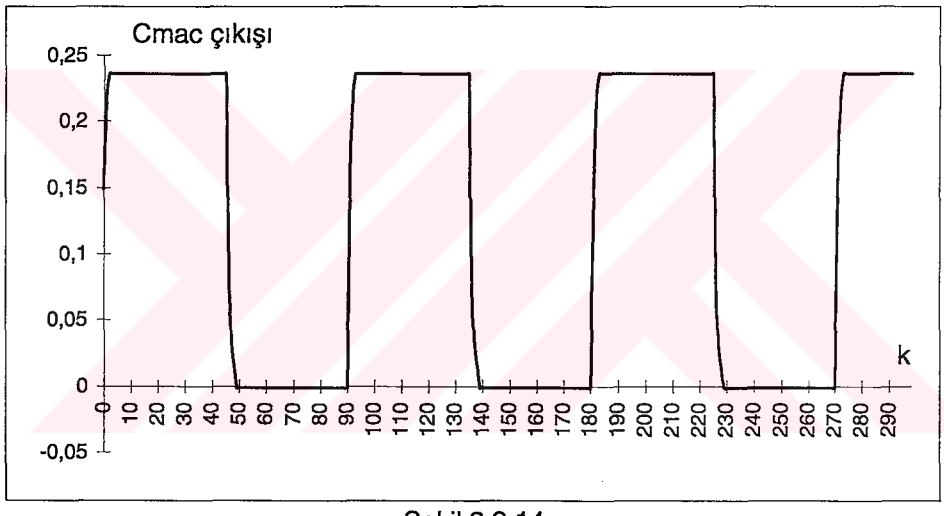
Şekil 3.3.11



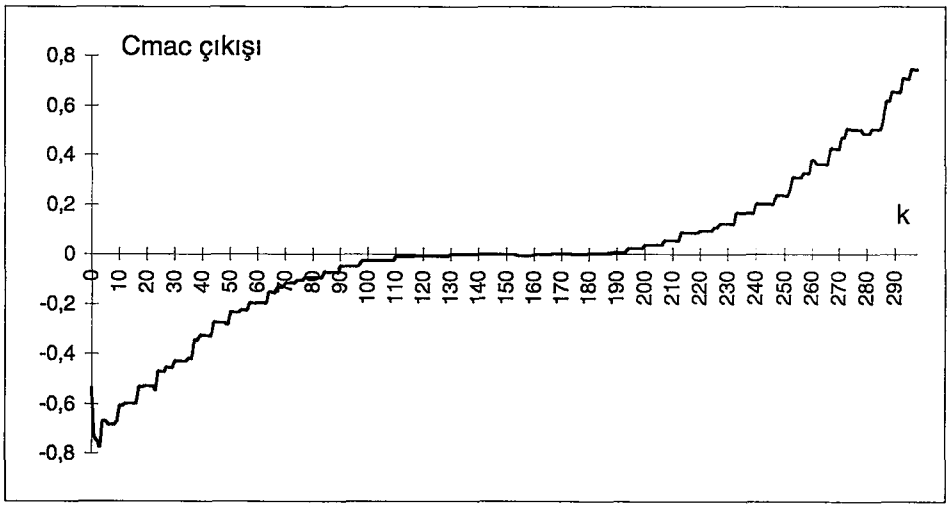
Şekil 3.3.12



Şekil 3.3.13



Şekil 3.3.14



Şekil 3.3.15

SONUÇLAR

Bir tür look-up table yöntemi olan CMAC'ın normal look-up table metodlarına göre öğreneceği fonksiyonun sahip olması gereken özelliklere bağlı olarak getirdiği daha az bellek kullanma özelliği vardır. Bu özelliğe sahip fonksiyonlar için klasik yöntemlere göre büyük miktarda bellek tasarrufu sağlamaktadır.

Hızından dolayı gerçek zaman uygulamalarına diğer yapay sinir ağı modellerinden daha uygundur.

Dezavantajı ise eğitim bölgesi dışında doğru sonuç vermemesidir. Bu yüzden çalışma bölgesi sınırlı olan mesela robot eklemleri konumlandırma işleminde kullanılmaya uygundur. Bu konuda yapılan simülasyonlarda da iyi sonuçlar elde edilmiştir.

Giriş uzayının sınırlarının belli olmadığı durumlarda kullanımında adresleme ve gerekli bellek miktarı konusunda problemler çıkabilir. Burada da Hashing yöntemi bellek çakışma problemi bir yere kadar gözardı edilerek doğrudan, fakat belli bir yerden sonra da bu problem için kullandığı algoritmalar devreye sokularak kullanılabilir.

KAYNAKLAR

[1] VILLEA CLAUDE A., SOLOMON ELORA P., MARTIN C.E. , Biology , Saunders College Publishing 1989.

[2] ZURADA JACEK M., Introduction to Artificial Neural Systems , West Publishing, 1992.

[3] ALBUS J.S.,"A New Approach to Manipulator Control: The Cerebellar Model Articulating Controller (CMAC)", Transactions of ASME, Journal of Dynamical Systems, Measurement and Control, Series G, Vol.97, No.3 pp.220-227 September 1975.

[4] ALBUS J.S., Brain Behaviour and Robotics McGraw-Hill,1981.

[5] KRUSE ROBERT L., Data Structures and Program Design , Prentice-Hall 1984.

[6] ALBUS J.S.,"Data Storage in the Cerebellar Model Articulating Controller (CMAC)",Transactions of ASME, Journal of Dynamical Systems,Measurement and Control, Series G, Vol.97, No.3 pp.228-233,September 1975.

ÖZGEÇMİŞ

Sertaç CANARAN 1970 yılında İstanbul'da doğdu. Maçka Teknik Lisesi Elektronik bölümünü 1988 yılında bitirerek aynı yıl İstanbul Teknik Üniversitesi Elektrik-Elektronik Fakültesi Kontrol ve Bilgisayar Mühendisliği bölümüne girdi. 1993 yılında İ.T.Ü Fen Bilimleri Enstitüsü Kontrol ve Bilgisayar anabilim dalında Yüksek Lisans eğitimine başladı.