

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DİNAMİK YAPAY SİNİR AĞLARI SİSTEMİ ÖNERİSİ

DOKTORA TEZİ

Erkam GÜREŞEN

Endüstri Mühendisliği Anabilim Dalı

Endüstri Mühendisliği Programı

MAYIS 2014

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DİNAMİK YAPAY SİNİR AĞLARI SİSTEMİ ÖNERİSİ

DOKTORA TEZİ

Erkam GÜREŞEN
(507082103)

Endüstri Mühendisliği Anabilim Dalı

Endüstri Mühendisliği Programı

Tez Danışmanı: Doç.Dr. Gülgün KAYAKUTLU

MAYIS 2014

İTÜ, Fen Bilimleri Enstitüsü'nün 507082103 numaralı Doktora Öğrencisi **Erkam GÜREŞEN**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı “**DİNAMİK YAPAY SİNİR AĞLARI SİSTEMİ ÖNERİSİ**” başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı : **Doç.Dr. Gülgün KAYAKUTLU**
İstanbul Teknik Üniversitesi

Jüri Üyeleri : **Prof.Dr. M. Nahit SERARSLAN**
İstanbul Teknik Üniversitesi

Prof.Dr. Cengiz KAHRAMAN
İstanbul Teknik Üniversitesi

Prof.Dr. Orhan FEYZİOĞLU
Galatasaray Üniversitesi

Doç.Dr. Umut R. TUZKAYA
Yıldız Teknik Üniversitesi

Teslim Tarihi : **04 Nisan 2014**
Savunma Tarihi : **23 Mayıs 2014**

Bilim için bir ömür çalışanlara,

ÖNSÖZ

Doktora eğitimim süresince engin bilgi ve birikimiyle yol gösteren tez danışmanım Sayın Doç.Dr. Gülgün KAYAKUTLU'ya ve değerli hocam Prof.Dr. M.Nahit SERARSLAN'a, finansal desteği için TÜBİTAK BİDEB'e, tezim ile ilgili yaptığı katkılardan dolayı Sayın Prof.Dr. Orhan FEYZİOĞLU'na ve tezimin yazım aşamasında verdiği destek ve gösterdiği sabır için Sayın Tnk.Kur.Alb. Mustafa DÖNMEZ'e en derin saygı ve şükranlarımı sunarım.

Ayrıca hayatım boyunca maddi-manevi desteklerini hiçbir zaman esirgemeyen çok değerli aileme sonsuz teşekkür ederim.

Nisan 2014

Erkam GÜREŞEN
(Endüstri Yüksek Mühendisi)

İÇİNDEKİLER

Sayfa

ÖNSÖZ.....	vii
İÇİNDEKİLER	ix
KISALTMALAR	xiii
ÇİZELGE LİSTESİ.....	xv
ŞEKİL LİSTESİ.....	xix
SEMBOL LİSTESİ	xxiii
ÖZET.....	xxv
SUMMARY	xxvii
1. GİRİŞ	1
1.1 Tezin Amacı	2
1.2 Çalışmanın Gerekliliği ve Literatüre Katkıları.....	3
1.3 Tezin Kapsamı.....	3
1.4 Tezin Yapısı	5
2. YAPAY SİNİR AĞLARI.....	7
2.1 Yapay Sinir Ağları ve Sinir Sistemi	7
2.1.1 Biyolojik sinir ağlarından yapay sinir ağlarına geçiş.....	7
2.2 Yapay Sinir Ağları Gelişimi.....	8
2.3 Yapay Sinir Ağı Tanımı	10
2.4 Kapsamlı Bir Yapay Sinir Ağı Tanımı.....	12
2.5 Yapay Sinir Ağı Modeli	14
2.5.1 Transfer fonksiyonu türleri	14
2.5.2 Kavrama - uyarılma türleri	19
2.5.3 Gözetimli (supervised) kavrama	19
2.5.4 Gözetimsiz (unsupervised) kavrama	20
2.5.5 Takviyeli (reinforced) kavrama	21
2.6 Çok Katmanlı Algılayıcılar (ÇKA)	23
2.6.1 Geri yayılım algoritması	25
2.7 Öz-örgütlemeli Haritalar	29
2.7.1 Kohonen öz-örgütlemeli haritası.....	29
3. YSA YAPILANDIRMA ALGORİTMALARI	31
3.1 Yapıcı Algoritmalar.....	31
3.1.1 Dinamik düğüm yaratma algoritması.....	31
3.1.2 İzdüşüm takip regresyonu	32
3.1.3 Basamak-korelasyon algoritması	32
3.1.4 Kaynak dağıtım ağları.....	33
3.1.5 Grup metoduyla veri işleme	35
3.1.6 Ma ve Khorasani'nin yapıcı stratejisi	36
3.1.7 DAN2	37
3.1.8 Kiremit algoritması	40
3.1.9 Kule algoritması.....	40
3.1.10 Upstart algoritması.....	43

3.1.11 CARVE	44
3.1.12 İleri beslemeli sinir ağlarının çapraz geçerlilikle inşa edilmesi	46
3.1.13 STRIP ağ büyüme algoritması	47
3.1.14 Petrol bulma algoritması	48
3.1.15 Sınırlandırılmış Coulomb enerji ağı.....	48
3.1.16 Rasgele bağlantılandırılmış ileri beslemeli sinir ağlarını birleştirme.....	49
3.1.17 Evrimsel yapıcı ve budayıcı algoritma (ECPA).....	50
3.1.18 İleri beslemeli sinir ağlarında bir yapı optimizasyon algoritması	52
3.1.19 Sinir ağları için hızlı yapıcı-kaplayıcı algoritma.....	52
3.1.20 C-Mantec.....	52
3.2 Budama Algoritmaları	53
3.3 Düzenleme Algoritmaları	54
3.4 Evrimsel Algoritmalar	54
3.5 Modüler Yapay Sinir Ağları ve Komite Makineleri.....	55
4. DİNAMİK VE BÜTÜNLEŞİK YSA GELİŞTİRİLMESİ.....	57
4.1 Gelişen (Improving Structured) YSA Gereksinimi.....	58
4.1.1 Türetilmiş YSA	62
4.1.2 Dinamik YSA.....	67
4.1.3 Bütünleşik YSA.....	71
4.2 Dinamik Yapay Sinir Ağı Algoritmaları	74
4.2.1 Temel algoritma 1: sadece bitiş elemanı ekleme	76
4.2.2 Temel algoritma 2: bitiş elemanı ve işlem elemanı ekleme	77
4.2.3 Temel algoritma 3: bitiş elemanını ihtiyaç anında belirleme.....	79
4.3 Aday Havuzu Oluşturma Yöntemleri	83
4.3.1 Kopyalama ile tek EE ekleme yöntemi (#EE00)	84
4.3.2 Rassal tek EE ekleme yöntemi(#EE01)	85
4.3.3 Ayırma ile tek EE ekleme yöntemi (#EE02).....	85
4.3.4 Paylaştırma ile tek EE ekleme yöntemi (#EE03)	86
4.3.5 Ters işaretli ağırlıkla EE ekleme yöntemi (#EE04)	86
4.3.6 Toplama ile EE ekleme yöntemi (#EE05).....	87
4.3.7 Birleştirme ile EE ekleme yöntemi (#EE06).....	87
4.3.8 Ayırma ile iki aday EE ekleme yöntemi (#EE07).....	87
4.3.9 Paylaştırma ile iki aday EE ekleme yöntemi (#EE08)	88
4.3.10 Kopyalama ile tek PE ekleme yöntemi (#PE00).....	88
4.3.11 Rassal tek PE ekleme yöntemi (#PE01).....	89
4.3.12 Ayırma ile tek PE ekleme yöntemi (#PE02)	90
4.3.13 Paylaştırma ile tek PE ekleme yöntemi (#PE03)	90
4.3.14 Ters işaretli ağırlıkla PE ekleme yöntemi (#PE04).....	91
4.3.15 Toplama ile PE ekleme yöntemi (#PE05).....	92
4.3.16 Birleştirme ile PE ekleme yöntemi (#PE06)	92
4.3.17 Ayırma ile iki aday PE ekleme yöntemi (#PE07)	93
4.3.18 Paylaştırma ile iki aday PE ekleme yöntemi (#PE08).....	94
4.4 Aday Seçimi	94
4.4.1 K-kat çapraz geçerlilik	95
4.4.2 K'nın L'li çapraz geçerliliği	97
4.5 Algoritmaların Durma Koşulları	99
5. DİNAMİK YAPAY SİNİR AĞI YAZILIMI.....	101
5.1 DYSA İçin Yazılım İhtiyacının Tespiti.....	101
5.1.1 Java NN.....	101
5.1.2 Heaton Research.....	102

5.1.3 Neurograph (Neuroph)	102
5.1.4 Mevcut yazılımların kullanımının değerlendirilmesi.....	102
5.1.5 Java editör seçimi	103
5.2 Yazılımın Mimari Yapısı	104
5.3 YSA Yapısı	104
5.4 DYSA Yapısı.....	106
5.5 Yazılımın Test Prosedürü.....	106
5.6 Algoritmaların Durma Koşulları	110
5.7 Aday Havuzu Oluşturma Yöntemleri.....	112
5.7.1 Kopyalama ile tek EE ekleme yöntemi (#EE00)	112
5.7.2 Rassal tek EE ekleme yöntemi (#EE01)	113
5.7.3 Ayırma ile tek EE ekleme yöntemi (#EE02)	114
5.7.4 Paylaştırma ile tek EE ekleme yöntemi (#EE03).....	114
5.7.5 Ters işaretli ağırlıkla EE ekleme yöntemi (#EE04)	115
5.7.6 Toplama ile EE ekleme yöntemi (#EE05)	116
5.7.7 Birleştirme ile EE ekleme yöntemi (#EE06).....	116
5.7.8 Ayırma ile iki aday EE ekleme yöntemi (#EE07).....	117
5.7.9 Paylaştırma ile iki aday EE ekleme yöntemi (#EE08).....	118
5.7.10 Kopyalama ile tek PE ekleme yöntemi (#PE00).....	119
5.7.11 Rassal tek PE ekleme yöntemi (#PE01).....	120
5.7.12 Ayırma ile tek PE ekleme yöntemi (#PE02)	120
5.7.13 Paylaştırma ile tek PE ekleme yöntemi (#PE03)	122
5.7.14 Ters işaretli ağırlıkla PE ekleme yöntemi (#PE04).....	123
5.7.15 Toplama ile PE ekleme yöntemi (#PE05).....	123
5.7.16 Birleştirme ile PE ekleme yöntemi (#PE06).....	124
5.7.17 Ayırma ile iki aday PE ekleme yöntemi (#PE07).....	125
5.7.18 Paylaştırma ile iki aday PE ekleme yöntemi (#PE08)	127
5.7.19 Temel algoritma 1 aday havuzu	128
5.7.20 Temel algoritma 2 aday havuzu	130
5.7.21 Temel algoritma 3 aday havuzu	132
5.8 Aday Seçimi	133
5.9 Parametreler	134
6. ALGORİTMALARIN PERFORMANS ÖLÇÜMÜ	135
6.1 Ortalama Hata Kareleri (OHK)	135
6.2 İkili Fonksiyon Doğru Sonuç Yüzdesi (İFDSY).....	135
6.3 Doğru Sınıflama Yüzdesi (DSY)	136
6.4 Geri Yayılım Algoritması İterasyon Sayısı	137
6.5 En İyi YSA Yapılarında Kullanılan Yöntemlerin İncelenmesi.....	137
6.6 Performansların Hesaplanması.....	138
7. UYGULAMA ÖRNEKLERİ	141
7.1 Ve, Veya, Özel Veya (And, Or, Xor) Birleşik Mantık Problemi	141
7.1.1 Problem tanımı	141
7.1.2 Uygulanan işlemler	144
7.1.3 TA01V01 sonuçları.....	145
7.1.4 TA02V01 sonuçları.....	149
7.1.5 TA03V01 sonuçları.....	150
7.1.6 TA03V02 sonuçları.....	151
7.2 Wolf'un SUNSPOT Fonksiyon Tahmini Problemi	152
7.2.1 Problem tanımı	152
7.2.2 Uygulanan işlemler	153

7.2.3 TA01V01 sonuçları	154
7.2.4 TA02V01 sonuçları	160
7.2.5 TA03V01 sonuçları	163
7.2.6 TA03V02 sonuçları	164
7.3 Aslanağzı (IRIS) Sınıflama Problemi	165
7.3.1 Problem tanımı	165
7.3.2 Uygulanan işlemler	169
7.3.3 TA01V01 sonuçları	169
7.3.4 TA02V01 sonuçları	176
7.3.5 TA03V01 sonuçları	178
7.3.6 TA03V02 sonuçları	180
7.4 Uygulama Sonuçları	183
7.4.1 En uygun parametreler	183
7.4.2 Adaptif öğrenme oranı	184
7.4.3 Aday havuzu oluşturma yöntemleri	185
7.4.4 Algoritmaların kıyaslanması	189
7.5 Bulguların Değerlendirilmesi	193
8. SONUÇLAR	201
8.1 Geliştirilen Yöntemlerin Potansiyel Uygulama Alanları	204
8.2 Öneriler ve Gelecekteki Çalışma Alanları	205
KAYNAKLAR	207
ÖZGEÇMİŞ	219

KISALTMALAR

AH	: Aday Havuzu
Anl.	: Anlamlılık
BE	: Bias Elemanı
BYSA	: Bütünleşik Yapay Sinir Ağı
cop.	: Kopya (Copy)
ÇG	: Çapraz Geçerlilik (Cross-Validation)
ÇKA	: Çok Katmanlı Algılayıcı (Multi-Layer Perceptron)
DAN2	: Dynamic Architecture for Neural Networks
des.	: İstenen (desired)
DSY	: Doğru Sınıflama Yüzdesi
DYSA	: Dinamik Yapay Sinir Ağı
GY	: Geri Yayılım
GYA	: Geri Yayılım Algoritması
E	: Eleman (SE, PE veya EE)
ECPA	: Evrimsel Yapıcı ve Budayıcı Algoritma (Evaluationary Constructive and Pruning Algorithm)
EE	: Bitiş Elemanı (End Element)
HK	: Hata Karesi
HSD	: Dürüstce Anamlı Fark (Honestly Significant Difference)
İFDSY	: İkili Fonksiyon Doğru Sonuç Yüzdesi
N2C2S	: Neural Network Construction with Cross-Validation Samples
OHK	: Ortalama Hata Karesi
orj.	: Orijinal
ÖÖH	: Öz Örgütlemeli Harita (Self-Organizing Map)
PE	: İşlem Elemanı (Processing Element)
RAN	: Kaynak Dağıtım Ağları (Resource Allocation Networks)
RTF	: Radyal Tabanlı Fonksiyon
RTFA	: Radyal Tabanlı Fonksiyon Ağları
sd	: Serbestlik Derecesi
SE	: Başlangıç Elemanı (Start Element)
seç.	: seçilen
TA01V01	: Temel Algoritma 1 Versiyon 1
TA02V01	: Temel Algoritma 2 Versiyon 1
TA03V01	: Temel Algoritma 3 Versiyon 1
TA03V02	: Temel Algoritma 3 Versiyon 2
TYSA	: Türetilmiş Yapay Sinir Ağı
YSA	: Yapay Sinir Ağı
YÇ	: Yönlü Çizge (Directed Graph)
#EE00	: Kopyalama ile Tek EE Ekleme Yöntemi
#EE01	: Rassal Tek EE Ekleme Yöntemi
#EE02	: Ayırma ile Tek EE Ekleme Yöntemi
#EE03	: Paylaştırma ile Tek EE Ekleme Yöntemi
#EE04	: Ters İşaretleli Ağırlıkla EE Ekleme Yöntemi

#EE05	: Toplama ile EE Ekleme Yöntemi
#EE06	: Birleştirme ile EE Ekleme Yöntemi
#EE07	: Ayırma ile İki Aday EE Ekleme Yöntemi
#EE08	: Paylaştırma ile İki Aday EE Ekleme Yöntemi
#PE00	: Kopyalama ile Tek PE Ekleme Yöntemi
#PE01	: Rassal Tek PE Ekleme Yöntemi
#PE02	: Ayırma ile Tek PE Ekleme Yöntemi
#PE03	: Paylaştırma ile Tek PE Ekleme Yöntemi
#PE04	: Ters İşaretili Ağırlıkla PE Ekleme Yöntemi
#PE05	: Toplama ile PE Ekleme Yöntemi
#PE06	: Birleştirme ile PE Ekleme Yöntemi
#PE07	: Ayırma ile İki Aday PE Ekleme Yöntemi
#PE08	: Paylaştırma ile İki Aday PE Ekleme Yöntemi

ÇİZELGE LİSTESİ

Sayfa

Çizelge 1.1 : Önerilen geliştirme algoritma işlevleri	4
Çizelge 2.1 : Biyolojik sinir ağları ile yapay sinir ağları arasındaki benzerlikler	7
Çizelge 2.2 : Biyolojik sinir hücreleri ile işlem elemanlarının karşılaştırılması	8
Çizelge 2.3 : Yapay sinir ağı çalışmalarının kronolojik gelişimi	8
Çizelge 2.4 : Kavrama algoritmalarının sınıflandırılması	21
Çizelge 2.5 : Kohonen ağına kavrama işlem adımları	30
Çizelge 4.1 : Literatürdeki YSA algoritmalarının özeti.....	58
Çizelge 5.1 : Geri yayılım algoritması durma parametreleri öntanımlı değerleri....	111
Çizelge 5.2 : NeuroBee parametreleri öntanımlı değerleri	134
Çizelge 6.1 : İstatistiksel testlerin H0 hipotezleri ve H0 kabul şartları	138
Çizelge 6.2 : Algoritmaların çalıştırılma planı	140
Çizelge 7.1 : Ve mantık problemi.....	142
Çizelge 7.2 : Veya mantık problemi	142
Çizelge 7.3 : Toplama bit işlemi.....	143
Çizelge 7.4 : Özel veya mantık problemi.	143
Çizelge 7.5 : Bit toplama işlemi	144
Çizelge 7.6 : ANOVA testi sonuçları (ve, veya, özel veya)	145
Çizelge 7.7 : Varyansların homojenliği (Levene) testi sonuçları (ve/veya/özel veya)	145
Çizelge 7.8 : İkili fonksiyon doğru yüzdesinin TA01V01 2 PE için Tamhane Post Hoc testi ile analizi.	146
Çizelge 7.9 : İkili fonksiyon doğru yüzdesinin TA01V01 5 PE için Tamhane Post Hoc testi ile analizi.	146
Çizelge 7.10 : İkili fonksiyon doğru yüzdesinin TA01V01 10 PE için Tamhane Post Hoc testi ile analizi.	146
Çizelge 7.11 : OHK'nin TA01V01 2 PE için Tamhane Post Hoc testi ile analizi. .	147
Çizelge 7.12 : OHK'nin TA01V01 5 PE için Tamhane Post Hoc testi ile analizi. .	147
Çizelge 7.13 : OHK'nin TA01V01 10 PE için Tamhane Post Hoc testi ile analizi.	147
Çizelge 7.14 : Toplam iterasyon sayısının TA01V01 2 PE için Tamhane Post Hoc testi ile analizi.....	148
Çizelge 7.15 : Toplam iterasyon sayısının TA01V01 5 PE için Tamhane Post Hoc testi ile analizi.....	148
Çizelge 7.16 : Toplam iterasyon sayısının TA01V01 10 PE için Tamhane Post Hoc testi ile analizi.....	148
Çizelge 7.17 : İkili fonksiyon doğru yüzdesinin TA02V01 için Tamhane Post Hoc testi ile analizi.....	149
Çizelge 7.18 : OHK'nin TA02V01 için Tamhane Post Hoc testi ile analizi	149
Çizelge 7.19 : Toplam iterasyon sayısının TA02V01 için Tamhane Post Hoc testi ile analizi.	149
Çizelge 7.20 : İkili fonksiyon doğru yüzdesinin TA03V01 için Tamhane Post Hoc testi ile analizi.....	150

Çizelge 7.21 : OHK'nin TA03V01 için Tamhane Post Hoc testi ile analizi.	150
Çizelge 7.22 : Toplam iterasyon sayısının TA03V01 için Tamhane Post Hoc testi ile analizi.....	151
Çizelge 7.23 : İkili fonksiyon doğru yüzdesinin TA03V02 için Tamhane Post Hoc testi ile analizi.	151
Çizelge 7.24 : OHK'nin TA03V02 için Tamhane Post Hoc testi ile analizi.	152
Çizelge 7.25 : Toplam iterasyon sayısının TA03V02 için Tamhane Post Hoc testi ile analizi.....	152
Çizelge 7.26 : SunSpot veri kümesi tanımlayıcı istatistikleri.	153
Çizelge 7.27 : Wolf'un Sunspot problemi için önerilen algoritmalarda kullanılan farklı çapraz geçerlilik değerleri.	153
Çizelge 7.28 : TA01V01'in 2 PE'li farklı ÇG değerleri için ANOVA testi sonuçları.	154
Çizelge 7.29 : TA01V01'in 2 PE'li farklı ÇG değerleri için Levene testi sonuçları..	154
Çizelge 7.30 : OHK bakımından TA01V01'in farklı ÇG değerlerinin 2 PE için Tamhane Post Hoc testi ile analizi.	154
Çizelge 7.31 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 2 PE için Tamhane Post Hoc testi ile analizi.	155
Çizelge 7.32 : TA01V01'in 5 PE'li farklı ÇG değerleri için ANOVA testi sonuçları.	156
Çizelge 7.33 : TA01V01'in 5 PE'li farklı ÇG değerleri için Levene testi sonuçları.	156
Çizelge 7.34 : OHK bakımından TA01V01'in farklı ÇG değerlerinin 5 PE için Tamhane Post Hoc testi ile analizi.	156
Çizelge 7.35 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 5 PE için Tamhane Post Hoc testi ile analizi.	157
Çizelge 7.36 : TA01V01'in 10 PE'li farklı ÇG değerleri için ANOVA testi sonuçları.	158
Çizelge 7.37 : TA01V01'in 10 PE'li farklı ÇG değerleri için Levene testi sonuçları.	158
Çizelge 7.38 : OHK bakımından TA01V01'in farklı ÇG değerlerinin 10 PE için Tamhane Post Hoc testi ile analizi.	159
Çizelge 7.39 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 10 PE için Tamhane Post Hoc testi ile analizi.	159
Çizelge 7.40 : TA02V01'in farklı ÇG değerleri için ANOVA testi sonuçları.	160
Çizelge 7.41 : TA02V01'in farklı ÇG değerleri için Levene testi sonuçları sonuçları.	161
Çizelge 7.42 : OHK bakımından TA02V01'in farklı ÇG değerlerinin Tukey'in HSD Post Hoc testi ile analizi.	161
Çizelge 7.43 : Toplam iterasyon sayısı bakımından TA02V01'in farklı ÇG değerlerinin Tamhane Post Hoc testi ile analizi.	162
Çizelge 7.44 : TA03V01'in farklı ÇG değerleri için ANOVA testi sonuçları.	163
Çizelge 7.45 : TA03V01'in farklı ÇG değerleri için Levene testi sonuçları.	163
Çizelge 7.46 : OHK bakımından TA03V01'in farklı ÇG değerleri için Tukey'in HSD Post Hoc testi ile analizi.	163
Çizelge 7.47 : Toplam iterasyon sayısı bakımından TA03V01'in farklı ÇG değerlerinin için Tamhane Post Hoc testi ile analizi.	164
Çizelge 7.48 : TA03V02'nin farklı ÇG değerleri için ANOVA testi sonuçları.	164
Çizelge 7.49 : TA03V02'nin farklı ÇG değerleri için Levene testi sonuçları.	165

Çizelge 7.50 : Aslanagzı sınıflama problemi için önerilen algoritmalarda kullanılan farklı çapraz geçerlilik değerleri.....	169
Çizelge 7.51 : TA01V01'in 2 PE'li farklı ÇG değerleri için ANOVA testi sonuçları	169
Çizelge 7.52 : TA01V01'in 2 PE'li farklı ÇG değerleri için Levene testi sonuçları	170
Çizelge 7.53 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 2 PE için Tamhane Post Hoc testi ile analizi	170
Çizelge 7.54 : TA01V01'in 5 PE'li farklı ÇG değerleri için ANOVA testi sonuçları	171
Çizelge 7.55 : TA01V01'in 5 PE'li farklı ÇG değerleri için Levene testi sonuçları	171
Çizelge 7.56 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 5 PE için Tamhane Post Hoc testi ile analizi	172
Çizelge 7.57 : TA01V01'in 10 PE'li farklı ÇG değerleri için ANOVA testi sonuçları	173
Çizelge 7.58 : TA01V01'in 10 PE'li farklı ÇG değerleri için Levene testi sonuçları	173
Çizelge 7.59 : TA01V01'in 10 PE'li farklı ÇG değerleri için Levene testi sonuçları	174
Çizelge 7.60 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 10 PE için Tamhane Post Hoc testi ile analizi.	175
Çizelge 7.61 : TA02V01'in farklı ÇG değerleri için ANOVA testi sonuçları.	176
Çizelge 7.62 : TA02V01'in farklı ÇG değerleri için Levene testi sonuçları.	176
Çizelge 7.63 : Toplam iterasyon sayısı bakımından TA02V01'in farklı ÇG değerlerinin Tukey'in HSD Post Hoc testi ile analizi	177
Çizelge 7.64 : TA03V01'in farklı ÇG değerleri için ANOVA testi sonuçları.	178
Çizelge 7.65 : TA03V01'in farklı ÇG değerleri için Levene testi sonuçları.	178
Çizelge 7.66 : Doğru sınıflama yüzdesi bakımından TA03V01'in farklı ÇG değerlerinin Tamhane Post Hoc testi ile analizi.	179
Çizelge 7.67 : OHK bakımından TA03V01'in farklı ÇG değerleri için Tamhane Post Hoc testi ile analizi.	179
Çizelge 7.68 : Toplam iterasyon sayısı bakımından TA03V01'in farklı ÇG değerlerinin için Tamhane Post Hoc testi ile analizi.	180
Çizelge 7.69 : TA03V02'nin farklı ÇG değerleri için ANOVA testi sonuçları.	180
Çizelge 7.70 : TA03V02'nin farklı ÇG değerleri için Levene testi sonuçları.	181
Çizelge 7.71 : OHK bakımından TA03V02'nin farklı ÇG değerleri için Tukey'in HSD Post Hoc testi ile analizi.	181
Çizelge 7.72 : Toplam iterasyon sayısı bakımından TA03V02'nin farklı ÇG değerleri için Tamhane Post Hoc testi ile analizi	181
Çizelge 7.73 : Algoritmaların öne çıkan çapraz geçerlilik değerleri.	183
Çizelge 7.74 : Algoritmaların önerilen çapraz geçerlilik değerleri	184
Çizelge 7.75 : Ve/Veya/Özel Veya problemi test verisinde elde edilen sonuçlar. ..	190
Çizelge 7.76 : Wolf'un SunSpot Zaman Serisi Problemi test verisinde elde edilen sonuçlar.....	191
Çizelge 7.77 : Aslanagzı (Iris) Sınıflama Problemi test verisinde elde edilen sonuçlar.....	192
Çizelge 7.78 : Aday havuzu oluşturmada önerilen öncelikli yöntemler.....	199

ŞEKİL LİSTESİ

Sayfa

Şekil 2.1 : YSA modelinde doğrusal transfer fonksiyonu.YSA modelinde doğrusal transfer fonksiyonu.....	15
Şekil 2.2 : YSA modelinde basamak transfer fonksiyonu.....	16
Şekil 2.3 : YSA modelinde kutuplama değerli transfer fonksiyonu.....	16
Şekil 2.4 : YSA modelinde parçalı doğrusal transfer fonksiyonu.....	17
Şekil 2.5 : YSA modelinde sigmoid transfer fonksiyonu.....	17
Şekil 2.6 : YSA modelinde tanjant hiperbolik transfer fonksiyonu.....	18
Şekil 2.7 : YSA modelinde sinüs transfer fonksiyonu.....	18
Şekil 2.8 : YSA'da çok katmanlı algılayıcı yapısı.....	24
Şekil 2.9 : İleri beslemeli geri yayılım yapısı.....	25
Şekil 2.10 : Hatanın geriye yayılım akış diyagramı	26
Şekil 2.11 : Kohonen öz örgütlemeli harita yapısı	29
Şekil 3.1 : Basamak-Korelasyon yapısı.....	33
Şekil 3.2 : Kaynak dağıtım ağlarının oluşturulması	34
Şekil 3.3 : Büyüyen hücre yapısı örneği.....	35
Şekil 3.4 : Grup metoduyla veri işleme yapısı örneği	36
Şekil 3.5 : (a) Başlangıçtaki lineer ağ (b) İlk gizli katmanın oluşumu.....	37
Şekil 3.6 : (a) İlk gizli katmana PE'lerin eklenmesi (b) İkinci gizli katmanın oluşturulması ve PE'lerin eklenmesi.....	37
Şekil 3.7 : DAN2 yapısı.....	38
Şekil 3.8 : Kiremit algoritmasını işleyişi	40
Şekil 3.9 : Kule algoritmasının gösterimi.....	41
Şekil 3.10 : Genişletilmiş kule algoritmasının gösterimi	42
Şekil 3.11 : Upstart algoritmasında yeni PE'ler eklenmesi	44
Şekil 3.12 : CARVE algoritması ile sınıfların ayrılması	45
Şekil 3.13 : Gizli katmandaki PE'ler ile CARVE algoritmasının gösterimi	45
Şekil 3.14 : N2C2S algoritması sözde programı	46
Şekil 3.15 : En büyük strip örneği	47
Şekil 3.16 : Sınırlandırılmış Coulomb Enerji Ağı nın oluşturulması	49
Şekil 3.17 : Rasgele bağlantılandırılmış ileri beslemeli sinir ağlarını birleştirme algoritması.....	50
Şekil 3.18 : ECPA Algoritması.....	51
Şekil 4.1 : Türetilmiş yapay sinir ağlarının çeşitleri.....	62
Şekil 4.2 : Yapay sinir ağı algoritmalarının sınıflanması.	64
Şekil 4.3 : Tedarik zinciri ağ gösterim örneği.	67
Şekil 4.4 : Eğitilmiş bir YSA'ya yeni bir EE ekleme örneği.....	69
Şekil 4.5 : Eğitilmiş bir YSA'ya yeni bir EE yanında yeni bir PE ekleme örneği. ...	69
Şekil 4.6 : EE'nin ihtiyaca ve şartlara göre belirlendiği YSA örneği.....	70
Şekil 4.7 : Zincir yapay sinir ağı örneği.	71
Şekil 4.8 : Örnek YSA	72
Şekil 4.9 : Gömülü Ağ örneği.....	72

Şekil 4.10 : Seri olarak birleştirilmiş YSA örneği.	73
Şekil 4.11 : Benzer YSA yapısının (a) geçici bağlantılarla elde edilmesi (b).	74
Şekil 4.12 : Temel Algoritma 1'in sözde programı.	76
Şekil 4.13 : TA01V01 algoritmasının çalışma örneği.	77
Şekil 4.14 : Temel Algoritma 2'nin sözde programı.	78
Şekil 4.15 : TA02V01 algoritmasının çalışma örneği.	79
Şekil 4.16 : Temel Algoritma 3 Versiyon 1'in sözde programı.....	81
Şekil 4.17 : TA03V01 algoritmasının çalışma örneği	82
Şekil 4.18 : Temel Algoritma 3 Versiyon 2'nin sözde programı.....	82
Şekil 4.19 : TA03V02 algoritmasının çalışma örneği	83
Şekil 4.20 : K=10 için K-kat çapraz geçerlilik kümeleri	96
Şekil 4.21 : K=10 ve L=3 için K'nın L'li çapraz geçerlilik kümeleri	98
Şekil 5.1 : Neurobee yazılımı temel sınıfları	105
Şekil 5.2 : NetBeans debug ekranı.....	107
Şekil 5.3 : Eclipse JUnit test ekranı (başarılı).....	109
Şekil 5.4 : Eclipse JUnit test ekranı (başarısız).	110
Şekil 5.5 : Kopyalama ile tek EE ekleme yönteminin (#EE00) sözde programı.....	113
Şekil 5.6 : Rassal tek EE ekleme yönteminin (#EE01) sözde programı.....	113
Şekil 5.7 : Ayırma ile tek EE ekleme yönteminin (#EE02) sözde programı.	114
Şekil 5.8 : Paylaştırma ile tek EE ekleme yönteminin (#EE03) sözde programı. ...	115
Şekil 5.9 : Ters işaretli ağırlıkla EE ekleme yönteminin (#EE04) sözde programı.	115
Şekil 5.10 : Toplama ile EE ekleme yönteminin (#EE05) sözde programı	116
Şekil 5.11 : Birleştirme ile EE ekleme yönteminin (#EE06) sözde programı.	117
Şekil 5.12 : Ayırma ile iki aday EE ekleme yönteminin (#EE07) sözde programı.	118
Şekil 5.13 : Paylaştırma ile iki aday EE ekleme yönteminin (#EE08) sözde programı	119
Şekil 5.14 : Kopyalama ile tek PE ekleme yönteminin (#PE00) sözde programı ...	119
Şekil 5.15 : Rassal tek PE ekleme yönteminin (#PE01) sözde programı	120
Şekil 5.16 : Ayırma ile tek PE ekleme yönteminin (#PE02) sözde programı.	121
Şekil 5.17 : Paylaştırma ile tek PE ekleme yöntemi (#PE03) sözde programı.....	122
Şekil 5.18 : Ters işaretli ağırlıkla PE ekleme yönteminin (#PE04) sözde programı.	123
Şekil 5.19 : Toplama ile PE ekleme yöntemi (#PE05) sözde programı.	124
Şekil 5.20 : Birleştirme ile PE ekleme yönteminin (#PE06) sözde programı.....	125
Şekil 5.21 : Ayırma ile iki aday PE ekleme yönteminin (#PE07) sözde programı..	126
Şekil 5.22 : Paylaştırma ile iki aday PE ekleme yönteminin (#PE08) sözde programı	127
Şekil 5.23 : Temel algoritma 1 için kullanılan aday havuzunun sözde programı....	128
Şekil 5.24 : Birinci tip aday oluşturma yönteminin sözde programı.	129
Şekil 5.25 : Temel algoritma 2 için kullanılan aday havuzunun sözde programı....	131
Şekil 5.26 : İkinci tip aday oluşturma yöntemi sözde programı	132
Şekil 5.27 : Aday seçim prosedürü.	133
Şekil 6.1 : Algoritmaların kıyaslanmasında kullanılan prosedür.....	139
Şekil 7.1 : Bit toplama işlemi gösterimi	144
Şekil 7.2 : SunSpot veri kümesi.....	153
Şekil 7.3 : Setosa türü aslanagzı çiçeği (Wikipedia)	166
Şekil 7.4 : Versicolor türü aslanagzı çiçeği (Wikipedia).	166
Şekil 7.5 : Virginica türü aslanagzı çiçeği (Wikipedia).....	167
Şekil 7.6 : Aslanagzı türleri, kırmızı; setosa, yeşil; versicolor, mavi; virginica (Wikipedia).....	168

Şekil 7.7 : Iris Probleminde %98.67 (148/150) oranında doğru sınıflandıran TA01V01 ile elde edilen 2 PE'li YSA	182
Şekil 7.8 : Adaptif kavrama oranı değişim örneği.	184
Şekil 7.9 : EE'ye sahip olmayan YSA'ların aday havuzundan seçilen en iyi yapılarda kullanılan yöntemler.....	186
Şekil 7.10 : Bir EE'ye sahip YSA'ların aday havuzundan seçilen en iyi yapılarda kullanılan yöntemler.....	187
Şekil 7.11 : Birden fazla EE'ye sahip YSA'ların aday havuzundan seçilen en iyi yapılarda kullanılan yöntemler.....	188
Şekil 7.12 : Ve/Veya/Özel Veya problemi ikili fonksiyon doğru yüzdesi sonuçları.	194
Şekil 7.13 : Ve/Veya/Özel Veya problemi OHK sonuçları.	194
Şekil 7.14 : Wolf'un SunSpot zaman serisi problemi OHK sonuçları.	195
Şekil 7.15 : Aslanagzı (Iris) problemi OHK sonuçları.	196
Şekil 7.16 : Aslanagzı (Iris) problemi % doğru sınıflama sonuçları.	196

SEMBOL LİSTESİ

A	: eğitim alternatifleri kümesi
A_i	: i 'nci eğitim alternatifi
α	: momentum oranı
b	: eşik (bias) elemanı sinyali
β_k	: k 'ıncı gözlemde hesaplanan sınıf değeri
$\beta_k^{des.}$	: k 'ıncı gözlemde istenilen sınıf değeri
c	: sıfıra yakın sabit bir sayı
d	: düğüm (Eleman) indisi
e	: hata (error)
ε	: yeterince küçük bir sayı
η	: öğrenme oranı
$\varphi(.)$	: Transfer fonksiyonu
G	: Yönlü çizge
$g(.)$	: olasılık kütle fonksiyonu
$h(.)$	: komşuluk fonksiyonu
I	: Girdi matrisi (Input Matrix)
t_{ki}	: k 'ıncı gözlemin i 'nci özellik değeri (i 'nci özellik aynı zamanda i 'inci SE'ye karşılık geliyor.)
$I(.)$	: Birim fonksiyon (Identity Function)
i	: düğüm (Eleman) indisi
$\mathfrak{I}_k$	: k 'ıncı gözlem için sınıflamanın doğruluğu ikili (binary) değişkeni
j	: düğüm (Eleman) indisi
K	: Çapraz geçerlilik toplam alt küme sayısı sabiti
k	: Gözlem (case, observation, record) indisi
L	: Çapraz geçerlilik seçilen alt küme sayısı sabiti
m	: Gözlem (case, observation, record) sayısı
n	: Özellik (attribute) sayısı veya düğüm sayısı
O	: Çıktı matrisi (Output Matrix)
$O^{des.}$	: İstenen çıktı matrisi
o_{ki}	: k 'ıncı gözlem için hesaplanan (YSA'dan elde edilen) i 'nci çıktı özelliği
o'_{ki}	: k 'ıncı gözlem için hesaplanan i 'nci ikili (binary) fonksiyon değeri
$o_{ki}^{des.}$	: k 'ıncı gözlem için istenen (YSA'dan elde edilen) i 'nci çıktı özelliği
p	: olasılık değeri / mantık önermesi
q	: olasılık değeri / mantık önermesi
r	: rassal sayı

ζ_{ki}	: k 'ıncı gözlem için i 'nci çıktı özelliğinin ikili fonksiyon doğruluk ikili (binary) değişkeni
t	: iterasyon numarası ya da zaman değişkeni
T	: eğitimde kullanacak gözlemler kümesi
T_i	: eğitimde kullanacak i 'nci gözlemler kümesi
τ	: Gözlem Matrisi ($\tau = [\mathbf{I} : \mathbf{O}^{des.}]$)
τ_k	: k 'ıncı gözlem
τ_{ki}	: k 'ıncı gözlemin i 'nci özelliği
θ	: transfer fonksiyonu parametresi
V	: çapraz geçerlemede kullanacak gözlemler kümesi
V_i	: çapraz geçerlemede kullanacak i 'nci gözlemler kümesi
U_i	: i 'inci düğüme gelen sinyallerin ağırlıklı toplamı
$\mathbf{W}$	: katmanlar arası (sinaptik) ağırlık vektörü
w_{0i}	: BE ile i 'nci düğüm arasındaki ayrıtın ağırlığı
w_{ij}	: i 'nci düğümden j 'nci düğüme olan ayrıtın ağırlığı
w_{ij}^*	: i 'nci düğümden j 'nci düğüme olan ayrıtın optimum ağırlığı
Δw_{ij}	: i 'nci düğümden j 'nci düğüme olan ayrıtın ağırlığı
x_i	: i 'nci düğümün durum değişkeni
x_i^k	: k 'ıncı gözlem için i 'nci düğümün durumu
ξ_j	: j 'inci düğümün anlık enerji değişkeni
ζ	: Sınıf (tür) kümesi
ζ_i	: i 'nci sınıf
ζ_{ki}	: k 'ıncı gözlem için i 'nci sınıf hesaplanan değeri

DİNAMİK YAPAY SİNİR AĞLARI SİSTEMİ ÖNERİSİ

ÖZET

Bu çalışmada öncelikle yazında “*öğrenme*” olarak geçen üç ayrı bilgi edinimi, yapılan tanımlarla birbirinden ayrılmıştır. Bu bilgi edinimlerinden ilki, ilk defa model oluşturmak için çevredeki gözlemlerin kullanılarak modelde belli bir bilgi birikimi sağlamayı *belleme* veya *kavrama* (*knowledge accumulation*) şeklinde tanımlanmış, ikinci tür bilgi birikimi ise çevrede meydana gelen değişikliklerden dolayı yeni kullanılabilir hale gelen gözlemlerin mevcut bir modele dâhil edilmesini sağladığı için *uyarlama* (*adaption*) olarak tanımlanmıştır. Üçüncü tür bilgi birikimi ise önceden belli konularda bilgi birikimi elde etmiş veya yetenek kazanmış modellerin, yeni bir konuda bilgi birikimi elde edecek veya mevcut yeteneklerine yeni bir yetenek daha kazanacak şekilde genişletilmesi olarak ifade edilmiş ve *geliştirme* (*enhancement*) olarak adlandırılmıştır.

Yapay sinir ağları açısından mevcut yazın incelendiğinde kavrama ve uyarlama yapan algoritmaların mevcut olduğu ama geliştirme yapan algoritmaların varlığına veya “geliştirme” kapsamında değerlendirilebilecek bir yaklaşıma rastlanmamaktadır.

Yapay sinir ağları açısından “geliştirme” yapan bir algoritmanın hangi işlemler sonucu bunu yapabileceği incelenmiştir. Yapılan incelemede bir algoritmanın yapay sinir ağı açısında geliştirme yapılabilmesi için mutlaka yapısına yeni bir işleve sahip, yeni bir amaca hizmet eden, yeni bir işleve sahip bir bitiş elemanı ekleyebilmesi gerektiği tespit edilmiştir.

Bu dönüşüm veya geliştirmelerin zaman içerisinde yapay sinir ağının yaptığı işi veya sağladığı faydaları yapısal olarak değiştirdiği için bu algoritmalar dinamik yapay sinir ağı algoritmaları olarak adlandırılmış, bu işlemlerin gerçekleştiği yapay sinir ağı da *dinamik yapay sinir ağı* olarak adlandırılmıştır.

Çalışmanın sınırlarını daha iyi çizebilmek için dinamik yapay sinir ağı kavramının yanında türetilmiş yapay sinir ağı ve bütünleşik yapay sinir ağı kavramları da oluşturulmuştur. Bu çalışmanın konu kapsamı olarak ise yazındaki eksiklik nedeniyle dinamik yapay sinir ağları olarak belirlenmiştir.

Dinamik yapay sinir ağı oluşturmak için üç farklı temel algoritma olmak üzere toplam dört yeni algoritma önerilmiştir.

Bu çalışmada önerilen algoritmaların performansları ortalama hata karesi, doğru sınıflama yüzdesi ve doğru ikili (binary) fonksiyon tahmini açısından değerlendirilmiştir. Bu değerlendirme sırasında önerilen algoritmalar için farklı çapraz geçerlilik sonuçları ve algoritmaların hangi aday oluşturma yöntemlerini daha çok kullandıkları incelenmiştir.

Önerilen algoritmalar, mevcut yazılımlardan SPSS Clementine’nın yapay sinir ağı modeli (dinamik) ve NeuroSolutions v.5.06 yazılımının genetik algoritma ile 25

kromozomun 25 nesil boyunca başlangıç ağırlıklarını, öğrenme oranlarını, momentum oranlarını ve gizli hücre sayısını tarayarak oluşturduğu tek gizli katmana sahip çok katmanlı algılayıcılayıcı modelleri ile karşılaştırılmıştır.

Yapılan karşılaştırmalar için üç örnek problem kullanılmıştır. Bu problemler sırasıyla ikili fonksiyon tahminine dayanan ve/veya/özel veya (and/or/xor) mantık problemi, ortalama hata karesini en küçüklenmeye çalışıldığı Wolf'un SunSpot problemi ve son olarak Fisher (1936) tarafından ayırma (discriminant) analizi için kullanılan, türleri doğru sınıflamaya dayanan, Aslanagzı (Iris) Sınıflama problemidir.

Bu çalışma kapsamında önerilen dört algoritma da karşılaştırma problemlerinde kayda değer performans göstermiştir. Önerilen dört algoritmadan her problemde ve her performans kriterinde öne çıkan algoritma TA03V02 olurken, en az performans sergileyen algoritma ise TA03V01 olmuştur.

DYNAMIC ARTIFICIAL NEURAL NETWORK SYSTEM PROPOSAL

SUMMARY

In this thesis three different knowledge gain, all referred as “*learning*” in literature, are differentiated by given definitions. First knowledge gain type called as *knowledge accumulation* which is the knowledge gain during model building phase using observations from environment. Second type of knowledge gain is referred as *adaption*, which is the knowledge gain through adding new accrued observations to an existing model due to environment change. Third type of knowledge gain is referred as *enhancement*, which is the knowledge gain by adding new output properties to an existing working model in such a way that existing model gains a new functionality.

When the literature is searched from artificial neural networks point of view, there are algorithms for knowledge accumulation and adaption but there is not any for enhancement. For this reason, the main aim of an algorithm, which does enhancement, is discussed for artificial neural networks. This results in adding new end element(s) in such a way that by adding a new end element the existing artificial neural network gains a new ability or functionality.

This transformation or enhancement changes purpose and structure of an existing and working artificial neural network model over time. Thus such algorithms, which enables enhancement, are referred as dynamic artificial neural network algorithms (DANNAs) and such artificial neural networks (ANNs) are referred as dynamic artificial neural networks (DANNs) in this study.

To clarify the boundaries of this study, derived artificial neural networks and integrated artificial neural networks are defined as well as dynamic artificial neural networks. Since there are some methods in the literature for derived artificial neural networks and integrated artificial neural networks, this study focuses only on DANNs.

To build a dynamic artificial neural network three fundamental algorithms in a total of four dynamic artificial neural network algorithms are proposed. The proposed algorithms in this study are; Fundamental Algorithm 01 Version 01 (FA01V01), Fundamental Algorithm 02 Version 01 (FA02V01), Fundamental Algorithm 03 Version 01 (FA03V01) and Fundamental Algorithm 03 Version 02 (FA03V02).

Firstly proposed algorithm FA01V01 only adds new end elements to an existing ANN structure and repeats this prose by adding one end element in each step until the desired number of end elements (output units in some manner) is reached.

Second proposed algorithm FA02V01, enables adding processing elements to hidden layer while adding end elements to output layer. FA02V01 repeats this proses by adding one end element in each step until the desired number of end elements (output units in some manner) is reached. FA02V01 differentiates from FA01V01 by evaluating only adding an end element with adding an end element with processing

element(s) to hidden layer at the same time. This prevents algorithm to become a decision model that selects to add an end element or processing element in each step. Instead it selects to add best performing structure by evaluating all possible structure alternatives, each consisting an end element.

Third proposed fundamental algorithm has two versions (FA03V01 and FA03V02). Fundamental Algorithm 3 (FA03) aims to add an end element to an ANN structured whenever needed. Thus it starts with self-organizing maps (in this study Kohonen's self-organizing map) to identify the patterns in data. Then FA03 uses FA01V01 in FA03V01 to add end elements and FA02V01 in FA03V02 to add end elements and processing elements.

When preparing candidate ANN structures, another problem emerges. The first and obvious solution is to prepare a candidate element (end element or processing element) with random weights in its connections. In this study in addition to random weighted elements, eight new methods for end elements and eight new methods for processing elements, in total nine methods for each, are proposed. Some of these proposed methods enable using some of the gained knowledge some of them enables forgetting some of the gained knowledge and some proposed methods enables both using and forgetting some of the gained knowledge.

Candidate structure parts, obtained by using methods mentioned above, are collected in a candidate pool and then best performing candidate structure part is selected by the corresponding proposed algorithm according to the predefined performance criteria for the problem.

Selecting the best performing candidate structure part is an important decision problem itself. Thus L of K cross validation is proposed, which is generalized version of K -fold cross validation. In a similar manner, proposed L of K cross validation divides dataset into K disjoint subsets as a partition set of dataset, in such a way that each subsets has nearly equal number of elements. In L of K cross validation L sequential set is selected for cross validation. K -fold cross validation is a special case of L of K cross validation in which $L=1$.

Best performing candidate is selected upon performance increase in each cross validation sets (L). First the number of performance increased sets is counted and the candidates with max number of performance increased cross validation sets are selected. If there is only one candidate, then the best candidate is this sole candidate, but if there is more than one candidate then the candidate which enables the most performance increase in cross validation sets due to performance criteria (mean square error, percentage of correctly classified examples etc.), is selected.

The proposed algorithms are evaluated due to performance criteria which are mean square error (MSE), percentage of correctly approximated binary function examples and percentage of correctly classified examples, in this study. Different L of K cross validation parameters and candidate preparing methods are used and inspected for proposed algorithms to identify which methods are used frequently.

Results of the proposed algorithms are compared with ANN (dynamic) model of SPSS Clementine software and 3 layered (one hidden layer) multi-layer perceptron (MLP) model of NeuroSolutions v.5.06 in which starting weights, learning rates, momentum rates and number of processing elements in hidden layer are genetically optimized during 25 generations using 25 chromosomes.

For the comparisons three different problems, existing in literature, are used. These problems are; binary function approximation based and/or/xor logic problem, MSE based Wolf's SunSpot problem and Iris Classification Problem, which is used by Fisher (1936) to exemplify discriminant analysis.

First problem, and/or/xor logic problem, has two binary inputs and three binary outputs. Due to recitative (20 repeats for each algorithm / model) calculations SPSS Clementine and NeuroSolutions produced models with 100 % correct results. On the other hand, proposed algorithms FA01V01, FA02V01 and FA03V02, produced models with 100 % correct results but not in all twenty repeats. FA03V01 algorithm produced model with maximum 92 % correct results. During this twenty repeats all four algorithms selected best candidates using "correct binary function result percent" as the performance criteria.

Second problem used in comparisons is the Wolf's Sunspot dataset, which is the annual record of Sun spots. Comparisons showed that both SPSS Clementine and NeuroSolutions models have higher MSE levels than all four proposed algorithms. In this problem, all four algorithms selected best candidates using "mean square error" as the performance criteria during all twenty repeats.

Last problem used in comparisons is the Iris classification problem. Comparisons showed that SPSS Clementine ANN model, NeuroSolutions ANN model, and models of three proposed models (FA01V01, FA02V01 and FA03V02) are reached 100 % correct classification level in test data during twenty repeats. On the other hand proposed FA03V01 algorithm produce ANN models with at most 76.7 % correct classification in test data during twenty repeats. During this twenty repeats all four algorithms selected best candidates using "correct classification percent" as the performance criteria.

In this study best L of K cross validation parameters for each algorithm with respect to each performance criteria is given. A remarkable finding of this study is that 2 of 5 cross validation reduces total number of back-propagation algorithm iterations in all four proposed algorithms.

1 of 10 cross-validation (10-fold cross-validation) is the best parameter for FA01V01 and FA02V01 algorithms with respect to mean square error and correct classification percent. While 10-fold cross-validation is best parameter for FA03V02 with respect to correct classification percent, 1 of 5 cross-validation (5-fold cross-validation) is the best parameter for FA03V02 with respect to mean square error. 2 of 5 cross-validation is the best parameter for FA03V01 algorithm with respect to mean square error, correct classification percent and total number of back-propagation algorithm iterations.

A remarkable structure in this study is constructed by FA01V01 for Iris Classification Problem. This 2 PE (processing element) Multi-Layer Perceptron (MLP) structure has 100 % (30/30) correct classifications in test data and 98.33 % (118/120) in training data. This Multi-Layer Perceptron (MLP) structure has 98,67 % (148/150) correct classification in total, which is the at most correct classification level in literature and higher than many machine learning models, with only. 2 PEs (Processing Elements). This MLP structure achieved this performance with 4 Start Elements and a bias element in first layer, 2 Processing Elements and a bias element in hidden layer and 3 End Elements in third layer.

All four proposed algorithms showed a remarkable performance in comparison problems. While the most noticeable algorithm is FA03V02, due to overall performance in all three comparison problems, FA03V01 is the least performing algorithm.

1. GİRİŞ

İnsanoğlunun bir gün kendi yarattığı zeki makineler tarafından esir edileceği akılları kurcalayan, belki de yapay zekâ kavramı hakkında ortaya atılan en spekülâtif iddialardan biridir. Bu spekülâtif iddialarda makineler kendi kendilerine yeni konulara kafa yormaya başlar ve yeni yargılara ulaşır. Bu iddialardaki ayrıntısına pek girilmeyen temel nokta yapay zekânın yeni konulara eğilebilmesi, yeni konularda bilgi ve görgü sahibi olabilmesidir.

Tüm bu hayali kurgulara rağmen mevcut yapay zekâ yöntemleri başlangıçta oluşturulan modeli daha etkin yapmayı, yani yeni işlev ve özellikler ekleyerek yeni konularda bilgi sahibi olmayı sağlamamaktadır. Bu kapsamda incelendiğinde yapay zeka yöntemleri, insanoğlunun belki de yapay zekâdan en başından beri beklediği, kendi kendine yeni sonuçlar üretebilme ve yeni konularda bilgi sahibi olabilme kabiliyetinden çok uzak bir konumdadır.

Yukarıda açıklanan bu durumu daha iyi ifade edebilmek ve yazında öğrenme (learning) olarak geçen, aslında birbirinden farklı kavramları net bir şekilde ayırabilmek için yeni tanımlar oluşturmak gerekmektedir.

Yazında sinaptik ağırlıkların bilgiyi yakalaması olarak tanımlanan öğrenme (Haykin, 1999; Eberhart ve Shi, 2007; Gurney, 2003; Rojas, 1996) verilerden örüntü çıkarma (pattern recognition) olarak da ifade edilebilir. Ancak yapılacak yeni tanımlamalar ile aşağıda belirtilen üç ayrı bilgi edinimi birbirinden ayrılmalıdır:

- İlk defa modeli oluşturma sırasındaki bilgi edinme,
- Daha önce oluşturulmuş ve kullanılabilir bir modele çevredeki değişimden dolayı oluşan yeni gözlemlerin dâhil edilmesi şeklindeki bilgi edinme,
- Daha önce oluşturulmuş ve kullanılabilir bir modelin yeni bir konuyu da kapsayacak şekilde genişletilmesi suretiyle bilgi edinme.

Yukarıda açıklanan üç bilgi ediniminden ilkinin, ilk defa model oluşturmak için çevredeki gözlemlerin kullanılarak modelde belli bir bilgi birikimi sağlamayı *belleme*

veya *kavrama (knowledge accumulation)* şeklinde tanımlayabiliriz. İkinci tür bilgi birikimini ise çevrede meydana gelen değişikliklerden dolayı yeni kullanılabilir hale gelen gözlemlerin mevcut bir modele dâhil edilmesini *uyarlama (adaption)* olarak tanımlayabiliriz.

Üçüncü tür bilgi birikimi ise önceden belli konularda bilgi birikimi elde etmiş veya yetenek kazanmış modellerin, yeni bir konuda bilgi birikimi elde edecek veya mevcut yeteneklerine yeni bir yetenek daha kazanacak şekilde genişletilmesini ise *geliştirme (enhancement)* olarak tanımlayabiliriz.

1.1 Tezin Amacı

Mevcut yazın incelendiğinde kavrama, uyarlama ve geliştirme olarak adlandırılabilir üç farklı bilgi edinimi için tek bir kavramın kullanıldığı, bunda “öğrenme” olduğu görülmektedir. Bu da mevcut durumdaki eksikliğin tanımlanmasını zorlaştırmaktadır. Örneğin yapay sinir ağı (YSA) oluşturma algoritmalarından yapay sinir ağlarını ilk defa eğitenler, kavramaya karşılık gelen bilgi edinimini sağlamaktadırlar. Değişen çevre şartlarına göre bir yapay sinir ağının yeni örnekler ile eğitilerek hata seviyesinin azaltılması ise uyarlamaya karşılık gelen bilgi edinimidir. Bütün bunlara karşılık bir yapay sinir ağına yeni bir yetenek kazandıracak şekilde yeni bir çıktı sinir hücresi ekleyen, kısaca geliştirme yapan algoritmalar ise bulunmamaktadır.

Yazındaki bütün YSA inşa algoritmaları YSA’nın çıktı katmanını ve burada bulunan çıktı sinir hücre sayısını sabit tutup, kavrama işlemini uygulamaktadır. Bu algoritmalarından bazıları çıktı sinir hücrelerini yapı olarak da sabit tutmaktadır. Fakat bazı algoritmalar ise çıktı sinir hücresini gizli sinir hücresine çevirerek yeni bir çıktı sinir hücresi eklemektedir. İşlevi eski çıktı sinir hücresinin hata miktarını düzeltmek olan bu yaklaşım da YSA yapısına yeni bir işlev katmadığından geliştirme olarak adlandırılmaz. Yapılan bu değiştirme işlemi kavramaya veya yerine göre uyarlamaya karşılık gelmektedir.

Bu çalışmanın temel amacı geliştirme sağlayan YSA modelleme algoritmaları geliştirmektedir. Buna olanak sağlayan YSA *dinamik yapay sinir ağı* (DYSA) olarak adlandırılacaktır. DYSA kavramı ilerleyen bölümlerde ayrıntılı olarak ele alınacaktır.

1.2 Çalışmanın Gerekliliği ve Literatüre Katkıları

Yapay sinir ağları açısından bakıldığında bu çalışmada tanımlanan şekilde geliştirmenin olabilmesi için yapay sinir ağlarının yeni çıktı sinir hücreleri ekleyebilmeleri gerekmektedir. Günümüzde yapay sinir ağları yapıları oluşturulurken, kara kutu yaklaşımı benimsenmektedir. Bu nedenle araştırmacılar ya sadece girdi sinir hücrelerinin sayısının ve gizli katmandaki yapının nasıl olması gerektiğiyle ilgilenir ya da girdi ve çıktı sinir hücrelerini sabitleyip, gizli katmandaki gizli sinir hücrelerinin sayısını tespit etmeye çalışırlar.

Mevcut yazın incelendiğinde yapay sinir ağlarına yeni yetenekler kazandıracak şekilde yeni çıktı sinir hücresi ekleyen bir algoritmanın eksikliği görülmektedir. Ayrıca bu konu ile ilgili kavramların da oluşturulmadığı görülmektedir.

Yapay zekâ çalışmalarının insana yakın düşünebilen yapıya yaklaşabilmesi için aşması gereken ilk problem, yapay zekâ modellerinin hangi yeni işlevleri veya özellikleri kazanması gerektiğidir. İkinci temel problem, bu yeni işlev ve özellik kazanma işlemlerinin yapılması için gerekli koşulların tespitidir. Üçüncüsü ve sonuncusu ise yeni işlev ve özellik kazanma işlemlerinin ne şekilde yapılacağıdır.

Yapay zekâ çalışmalarının önünde duran bu üç temel engeli “neyi”, “ne zaman” ve “nasıl” olarak kısaca özetleyebiliriz. Bu çalışma “nasıl” sorusunu sorarak, üçüncü temel engelin aşılması için bir çözüm arayışı içerisinde.

1.3 Tezin Kapsamı

Yapay zekâ çalışmalarının insan gibi düşünebilen bir yapıya “nasıl” kavuşabileceği bu çalışmanın başlangıç noktasıdır. Bu kapsamda yapay sinir ağları ele alınmış ve yeni çıktı sinir hücrelerinin mevcut bir yapıya nasıl eklenebileceği üzerinde durulmuştur.

Bu çalışmada “nasıl?” sorusuna cevap olarak çıktı hücrelerini ekleyen dört yeni algoritma geliştirilmiştir. Tanımsız bir çıktı katmanı ile başlayan ve istenilen miktarda yeni çıktı sinir hücresi ekleyerek YSA inşa eden yeni algoritmalar geliştirilmiştir. Bu algoritmalar bir yapay sinir ağının çıktı katmanını sıfırdan oluşturabileceği gibi daha önce eğitilmiş bir yapay sinir ağına yeni işlevler eklemek

için çıktı sinir hücresi de ekleyebilmektedir. Bu çalışmada geliştirilen algoritmalar aşağıdaki çizelgede belirtilmiştir.

Çizelge 1.1 : Önerilen geliştirme algoritma işlevleri.

Algoritma	İşlev
Temel Algoritma 1 Versiyon 1 (TA01V01)	Çıktı Sinir Hücresi Ekleme
Temel Algoritma 2 Versiyon 1 (TA02V01)	Gizli Sinir Hücresi Ekleme Çıktı Sinir Hücresi Ekleme
Temel Algoritma 3 Versiyon 1 (TA03V01)	Öz-örgütlemeli Harita ile İz Bulma Çıktı Sinir Hücresi Ekleme
Temel Algoritma 3 Versiyon 2 (TA03V02)	Öz-örgütlemeli Harita ile İz Bulma Gizli Sinir Hücresi Ekleme Çıktı Sinir Hücresi Ekleme

Bu çalışma daha önce açıklanan “nasıl” sorusuna yanıt olarak yeni yöntemler önermeyi amaçlamaktadır. Bu önerilen yöntemlerin geçerliliğini ve sonuçlarını incelemek amacıyla ise aşağıdaki sorulara yanıt arayacaktır:

1. Aday havuzunda yeni adaylar oluşturmak için nasıl bir çapraz geçerlilik kullanılmalıdır?
2. En iyi adayı hangi yöntem oluşturmaktadır?
3. Geliştirilecek değişik işlevli algoritmaların hangisi daha iyi sonuçlar üretir?
4. Önerilen algoritmalar, geri yayılım algoritması (GYA) ve bu algoritmayı iyileştirme yöntemleri kullanan YSA paket programlarının sonuçları ile karşılaştırılabilir mi?

İlk sorunun cevabıyla önerilen algoritmalarda aday seçiminde kullanılan çapraz geçerlikte kullanılan parametrelerin alması gereken değerler tespit edilmeye çalışılacaktır.

İkinci soruda ise algoritmaların çalışırken oluşturdukları adaylardan hangi yöntemle oluşturulan adayların öne çıktığı incelenmek istenmektedir. Böylece bu algoritmalar daha sonra kullanılırken aday oluşturma yöntemleri olarak bu çalışmada öne çıkan yöntemlere daha fazla öncelik verilebilir.

1.4 Tezin Yapısı

Bu çalışma sekiz bölümden oluşmaktadır. Bir sonraki bölümde özellikle bu çalışma kapsamında kullanılan yazındaki yapay sinir ağı modelleri ve tarihi gelişimleri açıklanacaktır. Kısa bir yapay sinir ağı gelişim süreci verilip bu çalışmanın önünü açan, yazındaki kavramları netleştiren kapsamlı bir yapay sinir ağı tanımı verilecektir.

Üçüncü bölümde temel olarak yapay sinir ağı yapısını oluşturmada kullanılan yöntemler ve algoritmalar incelenecektir. Bu bölümde özellikle yapay sinir ağlarına geliştirme yaptırmaya potansiyeli yüksek olan yapıcı algoritmalar ayrıntılı olarak incelenecektir.

Dördüncü bölümde türetilmiş yapay sinir ağı (TYSA), bütünleşik yapay sinir ağı (BYSA) ve dinamik yapay sinir ağı (DYSA) kavramları oluşturulacak ve dinamik yapay sinir ağı için temel algoritmalar önerilecektir. Bu temel algoritmalarda kullanılan aday yapı oluşturma yöntemleri ve yapı seçim prosedürü açıklanacaktır.

Beşinci bölümde ise JAVA programlama dili kullanılarak önerilen algoritmalar için geliştirilen NEUROBEE programında dördüncü bölümde verilen önerilen algoritma ve yöntemlerin nasıl pratiğe dönüştürüldüğü sözde programlar (pseudo codes) yardımıyla açıklanacaktır.

Altıncı bölümde algoritmaların farklı problemlerdeki performanslarının nasıl değerlendirileceği açıklanmıştır. Yedinci bölümde ise algoritmaların farklı problemlere nasıl uygulandığı, bu problemlerin türleri ve bu problemlerdeki uygulamalar sonucunda elde edilen bulgulara ve istatistiksel test sonuçlarına yer verilecektir.

Sekizinci ve son bölümde ise elde edilen sonuçlar değerlendirilecek ve ilerideki çalışmaların hangi yönlerde ilerleyebileceği tartışılacaktır.

2. YAPAY SİNİR AĞLARI

2.1 Yapay Sinir Ağları ve Sinir Sistemi

Yapay sinir ağı (YSA) çalışmaları biyolojik sinir sisteminden esinlenerek matematiksel model oluşturmaya dayanır. 1940’lardan önce çalışmalar bulunsa da (Baş, 2006), bu alandaki ilk somut çalışma olarak McCulloch ve Pitts (1943) kabul edilmektedir.

Yapay sinir ağları açıklanmadan önce biyolojik sinir ağlarından ilham alınan noktaların incelenmesi YSA modellerini algılamayı kolaylaştıracaktır.

2.1.1 Biyolojik sinir ağlarından yapay sinir ağlarına geçiş

Literatürdeki yapay sinir ağlarını açıklayan çalışmaların birçoğunun temelinde biyolojik sinir ağları ile yapay sinir ağlarının karşılaştırması yer almaktadır (Braspenning ve diğ. 1995; Muller ve diğ., 1995; Rojas, 1996; Haykin, 1999). İlham alınan noktalar ve benzerliklerini Çizelge 2.1’deki gibi özetleyebiliriz.

Çizelge 2.1 : Biyolojik sinir ağları ile yapay sinir ağları arasındaki benzerlikler (Braspenning ve diğ. 1995; Muller ve diğ., 1995).

Biyolojik Sinir Ağları	Yapay sinir Ağları
Uyarı	Girdi
Reseptörler	Girdi katmanı
Sinir ağı	İşlem katmanı
Nöron	İşlem elemanı
Efektörler	Çıktı katmanı
Cevap	Çıktı

Biyolojik sinir hücreleri ile işlem elemanlarının faaliyetleri Çizelge 2.2’deki gibi karşılaştırılabilir. Özet olarak sinapslar gelen uyarının ağırlığı gibi çalışması YSA’daki sinaptik ağırlıklara; dendritler gelen ağırlıklı uyarıyı bileştirmesi toplama fonksiyonuna; hücre gövdesinin toplam uyarıyı işleyip yeni bir uyarı üretmesi transfer fonksiyonuna, oluşan çıktıyı diğer sinir hücrelerine ileten akson YSA’da çıktıya ve çıktı bağlantılarına ve son olarak da uyarıyı azaltan arttıran eşik değeri ise sapma’ya (bias) ilham kaynağı olmuştur.

Çizelge 2.2 : Biyolojik sinir hücreleri ile işlem elemanlarının karşılaştırılması (Baş, 2006).

Nöronlar	İşlem Elemanları
Sinapslar	Ağırlıklar
Dentritler	Toplama Fonksiyonu
Hücre Gövdesi	Transfer Fonksiyonu
Akson	Çıktı
Eşik değeri	Sapma (Bias)

2.2 Yapay Sinir Ağları Gelişimi

Yapay sinir ağları tarihi bilgisayarların gelişimi ile paralellik gösterir (Emir, 2013). Her ikisi de benzer zamanlarda, yani 1940’larda başlar. İlk çalışmalar nörobiyoloji konusundaki keşiflerden esinlenen araştırmacıların çalışmaları ile başlamıştır. Çalışmalar, odak açısından 1970 öncesi ve sonrası olmak üzere iki dönemde incelenebilir (Öztemel, 2012). 1970 öncesi çalışmalar daha çok sinir hücrelerini modellemeye yönelmişken 1970 sonrası çalışmalar bilgisayar teknolojisindeki ilerlemelerle birlikte yapay sinir ağı eğitimi ve yapay sinir ağının en iyi yapısını bulmaya yönelmiştir. Yapay sinir ağı çalışmalarının kronolojik gelişimi Şekil 2.3’te verilmiştir.

Çizelge 2.3 : Yapay sinir ağı çalışmalarının kronolojik gelişimi.

Yıllar	Yapılan Çalışmalar
1943	Biyolojik sinir hücresinden esinlenilerek matematiksel model geliştirilmesi (Mc Culloch ve Pitts, 1943)
1949	Hebb kuralının ortaya çıkışı (Hebb, 1949)
1950	Beyindeki bilginin dağıtık (distributed) bir şekilde depolandığının keşfedilmesi (Lashley, 1950)
1954	Adaptif tepki üretme kavramının ortaya çıkışı (Farely ve Clark, 1954)
1958	Algılayıcı (perceptron) modelinin tanıtılması (Rosenblatt, 1958)
1959	ADALINE (ADaptive LInear NEuron) modelinin geliştirilmesi (Mattson, 1959a; 1959b; Widrow ve Hoff, 1960)

Çizelge 2.3 (devam): Yapay sinir ağı çalışmalarının kronolojik gelişimi.

Yıllar	Yapılan Çalışmalar
1961	Çağrışimli bellek (associative memory) ile ilgili ilk çalışmaları yapılması (Steinbruch, 1961)
1965	İlk makine öğrenmesi kitabının yayınlanması (Nilson, 1965)
1969	Algılayıcıları lojik fonksiyonlar için de kullanılması (Minsky ve Papert, 1969)
1969	Perceptron modelinin birçok temel problemi (örneğin XOR problemi) çözme yeteneğinin olmadığını gösterilmesi (Minsky ve Papert, 1969)
1972	Çağrışimli bellek (associative) modellerinin geliştirilmesi (Kohonen, 1972; Anderson, 1972)
1973	Öz örgütlemeli haritaların geliştirilmesi (Malsburg, 1973)
1974	Hatanın geri yayılımı (error back propagation) kavramının geliştirilmesi (Werbos, 1974)
1975	Çok katmanlı algılayıcıların geliştirilmesi (Fukushima, 1975)
1976	ART (Adaptive Resonans Theory) (Grossberg, 1976)
1982	Hopfield ağının açıklanması (Hopfield, 1982)
1982	Kohonen öz örgütlemeli ağının geliştirilmesi(Kohonen, 1982a)
1983	Miyake Fukushima ve Ito Fukushima tarafından el yazısı karakterlerini tanıyabilen “Neocognitron” modelinin tanıtılması
1984	Geoffrey E. Hinton ve Terry Sejnowski tarafından “Boltzmann Makinesi” geliştirilmesi
1985	Gezgin satıcı probleminin Hopfield ağları kullanılarak kabul edilebilir çözümler elde edilmesi (Hopfield, 1985)
1986	Geri yayılım algoritması (Rumelhart ve diğ., 1986)
1988	Olasılıksal Ağ (Probabilistic Neural Networks) modelinin geliştirilmesi (Specht, 1988)
1988	Tekrarlı (recurrent) ağların kullanılması (Webos, 1988)
1990	Çok değerli mantık problemlerine yapay sinir ağlarının uygulanması (Olmsted, 1990)

Çizelge 2.3 (devam): Yapay sinir ağı çalışmalarının kronolojik gelişimi.

Yıllar	Yapılan Çalışmalar
1991	Çok katmanlı algılayıcıların fonksiyon tahmin kabiliyetlerinin ispatlanması (Hornik, 1991)
1991	Genel Regresyon Ağları (General Regression Neural Networks) modelinin geliştirilmesi (Specht, 1991)
1991 sonrası	Yeni çalışmalar daha çok mevcut YSA modellerini eğitme ve sezgisel ve meta-sezgisel yöntemlerle parametrelerinin optimizasyonuna odaklanmıştır.

2.3 Yapay Sinir Ağı Tanımı

YSA'nın saygın bir tanımı Haykin (1999) tarafından çevreden öğrenme yoluyla bilgi edinen basit işlem birimlerinin, kitlesel olarak paralel bağlanması sonucu oluştuğunu ve bağlantı noktalarında öğrendiği bilgiyi sakladığını belirtilerek yapılmıştır.

Eberhart ve Shi (2007), Gurney (2003), Rojas (1996), ve Muller ve diğ. (1995) tanımlarında işlem elemanı ve öğrenme algoritmasını vurgulamışlardır. Rojas (1996) ayrıca hala biyolojik sinir hücresinin hesaplama mekanizmasını tam olarak anlayamadığından dolayısıyla yapay sinir terimi yerine işlem elemanı (PE) terimini kullanıldığını belirtmiştir. Öğrenme ise Haykin (1999), Eberhart ve Shi (2007), Gurney (2003), ve Rojas (1996) tarafından sinaptik ağırlıkların bilgiyi yakalaması olarak tanımlanmıştır. Haykin (1999) ve Gurney (2003) ise YSA'ların kendi yapılarını değiştirebileceklerini vurgulamıştır.

İşlem elemanları modellenirken hayvan sinir sisteminde bulunan sinir hücrelerinden ilham almıştır. Gerçek sinir hücreleri uyarıyı sinapsları vasıtasıyla alıp onları belli oranlarda birleştirir ve girdiye benzemeyen bir çıktı oluştururlar. Bu yüzden Haykin (1999) işlem elemanlarının üç temel parçasını sinaptik ağırlıklar, gelen girdileri birleştirmek için bir toplama fonksiyonu ve bir çıktı üretmek için transfer fonksiyonu olarak tanımlamıştır. Eberhart ve Shi (2007) bir işlem elemanının çıktısını girdileri birleştirmek için toplama yapılan ve sonrasında transfer fonksiyonu kullanılarak tek bir çıktı üretilen bir fonksiyonlar fonksiyonu olarak tanımlamıştır. Benzer şekilde Friedman ve Stuetzle (1981) ve Rojas (1996) biyolojik sinir hücresinin dört yapısının

(dendrit, sinapslar, hücre gövdesi ve akson) modellerde kullanılmak üzere mutlaka alınması gerektiğini belirtmiştir.

Rojas (1996) YSA'yı pragmatik bir şekilde şöyle tanımlamıştır:

“YSA’lar birçok farklı işlem elemanından inşa edilmiş dağılmış, adaptif, genellikle doğrusal olmayan öğrenme makineleridir. Her işlem elemanı kendisinden veya bir öncekinden bağlantılar alır. Bu bağlantılardaki sinyal akışı ağırlık w_{ij} adı verilen ayarlanabilir parametrelerle yeniden ölçeklendirilebilir. İşlem elemanları tüm bu sinyal parçalarını toplayarak ve bu toplamı doğrusal olmayan bir fonksiyonda kullanarak yeni bir çıktı üretirler. İşlem elemanlarının çıktıları ya sistemin çıktısı olurlar ya da diğer işlem elemanlarına veya kendisine girdi olarak yollanır.”

Bu noktadan sonraki tanımları daha anlaşılır kılmak için çizge kuramındaki bazı tanımları belirtmek faydalı olacaktır.

Geometrik olarak çizge uzayda bulunan noktalar (düğümler) kümesinden ve bu noktaların birbirlerine olan bağlantıları bir ayrıtlar kümesiyle tanımlanmasıdır (Gibbons, 1999). *Ağırlıklandırılmış çizge* ise her bir ayrıtına bir ağırlık tanımlanmış bir çizge olarak tanımlanabilir (Gibbons, 1999). *Yönlü çizge* ise her bir e ayrıtının bir okla gösterildiği ve ok yönünde bir yön belirtilmiş olan çizgedir (Gibbons, 1999). Son olarak çizgelerde bağlı olmayı tanımlamamız gerekir. Bir çizgede eğer i düğümünden j düğümüne bir yol varsa i düğümü j düğümüne *bağlıdır* denir (Gibbons, 1999).

Haykin (1999) YSA'yı yönlü çizge olarak tarif ederken 3 adet grafiğin kullanılması gerektiğini söyler, bunlar:

- Ağ çalışmasını gösteren blok diyagramı,
- Ağdaki sinyallerin akışını gösteren sinyal akış çizgesi,
- Ağ yapısının tasvir edildiği mimari bir çizge.

Haykin (1999) bu tanımları öğrenmeyi, girdi ve çıktı kümelerini (girdi ve çıktıda özellik sayısını ve gözlem sayısını), ve paralel yapıyı belirtmediği için tam bir YSA tanımı olarak adlandırılmaz.

Literatürde bulunabilecek matematiksel bir tanım ise Muller ve diğ. (1995) tarafından şöyle ifade edilmiştir:

“Bir yapay sinir ağı modeli aşağıdaki özelliklere sahip bir yönlü çizge olarak ifade edilebilir:

- Her bir i düğümüne bağlı bir x_i durum değişkeni
- Her i düğümünü j düğümüne bağlayan ij ayrıtı için reel bir değere sahip w_{ij} ağırlığı
- Her bir i düğümüne bağlı bir reel değere sahip b_i sapma (bias) değeri
- Her bir i düğümü için tanımlanmış, düğümün durumunu eşik değeri, gelen girdilerin ağırlıklıları ve i düğümüne bağlı her bir j düğümünün durumuna bağlı olarak belirleyen $f_i = \left[x_j, w_{ji}, v_i (i \neq j) \right]$ transfer fonksiyonu”.

Muller ve diğ. (1995) yukarıdaki tanıma ek olarak kendisine gelen bağlantı olmayan düğümleri girdi ve kendisinden çıkan bağlantısı olmayan düğümleri ise çıktı düğümü olarak tanımlamışlardır. Özellikle bu son iki tanım kendi çıktısı kendisinin girdisi olabilen (recurrent) tekrarlı yapay sinir ağlarında sorun oluşturmaktadır. Bu yüzden girdi ve çıktı düğümlerinin tanımları daha net olmalıdır.

Muller ve diğ. (1995) yaptığı tanımla ilgili diğer bir problem ise girdi ve çıktı düğümlerini YSA tanımı içerisinde net bir şekilde kullanmamalarıdır. Gerçekte bir YSA diğer çizgelerin sahip olmadığı ve belli özelliklere sahip girdi ve çıktı düğümlerine sahip olmaktadır. Ayrıca Muller ve diğ. (1995) ve Haykin (1999) olduğu gibi paralel yapıya tanımlarında değinmemişlerdir. Tüm bu tanımlar Guresen ve Kayakutlu (2010; 2011b) belirttiği gibi YSA’ların diğer çizgelerle kolaylıkla karıştırılmasına neden olabilmektedirler.

2.4 Kapsamlı Bir Yapay Sinir Ağı Tanımı

İlham alınan noktaları da göz önüne alarak paralel dağılımı içeren ve YSA’yı diğer çizgelerden kolayca ayırt etmemizi sağlayabilen Guresen ve Kayakutlu (2011a; 2011b)’nin aşağıda verilen YSA tanımını ele alalım;

Eğer bir yönlü çizge

- En az bir başlangıç düğümü (Başlangıç Elemanı, SE) varsa
- En az bir bitiş düğümü (Bitiş Elemanı, EE) varsa
- En az bir işlem elemanı (PE) varsa
- Başlangıç ve bitiş elemanları hariç tüm düğümler işlem elemanı (PEs) ise
- Her bir i düğümüne bağlı bir x_i durum değişkeni

- Her i düğümünü j düğümüne bağlayan ij ayrıtı için reel bir değere sahip w_{ij} ağırlığı
- Her bir i düğümüne bağlı bir reel değere sahip b_i sapma (bias) değeri
- Birden fazla işlem elemanı olması durumunda en az iki tanesinin paralel bağlanmış olması
- Verilen girdiler için arzu edilen çıktıların alınmasını modellemeye yardım eden bir öğrenme algoritmasının varlığı
- j düğümünü her i düğümüne bağlayan her ji ayrıtındaki akışın j düğümün çıktısına bağlı olan x_j 'ye eşit ve aynı olması
- Her bir başlangıç düğümü en az bir bitiş düğümüne bağlıysa ve her bir bitiş düğümü en az bir başlangıç düğümüne bağlıysa
- Her j düğümü ile i düğümü arasında paralel ayrıt yoksa

bu özelliklerinin hepsine sahip olan yönlü çizgelere Yapay Sinir Ağı (YSA) denir.

Bu tanımı tam olarak anlamak için Guresen ve Kayakutlu (2011a; 2011b)'nin aşağıda verilen *Başlangıç Elemanı (SE)*, *Bitiş Elemanı (EE)*, *İşlem Elemanı (PE)* ve *Öğrenme Algoritması* tanımlarını incelemek gerekmektedir.

Yönlü bir çizgedeki i düğümü I_{ki} girdisini n özellikli, m bağımsız kayıttan oluşan $\mathbf{I} = \{I_{di}; d = 1, 2, \dots, m; i = 1, 2, \dots, n\}$ girdi matrisinden alıp yönlü çizgede bir akış başlatıyorsa *Başlangıç Elemanı (SE)*dir (Guresen ve Kayakutlu, 2011a; 2011b).

Yönlü bir çizgedeki i düğümü O_{ki} çıktısını n özellikli, m farklı istenen çıktı kaydından oluşan $\mathbf{O} = \{O_{di}; d = 1, 2, \dots, m; i = 1, 2, \dots, n\}$ çıktı matrisine bir çıktı kaydı için bir özellik üretip yönlü çizgede bir akış bitiriyorsa *Bitiş Elemanı (EE)*dir (Guresen ve Kayakutlu, 2011a; 2011b). G aşağıdaki özelliklere sahip yönlü bir çizge olsun;

- Her bir i düğümüne bağlı bir x_i durum değişkeni olan
- Her i düğümünü j düğümüne bağlayan ij ayrıtı için reel bir değere sahip w_{ij} ağırlığı olan
- Her bir j düğümüne bağlı bir reel değere sahip b_j sapma (bias) değeri olan
- Her i düğümü ile j düğümü arasında sadece bir ayrıt bulunan (paralel ayrıt bulundurmayan)

Böyle bir G çizgesindeki j düğümü için $f_j[x_i, w_{ij}, b_j (i \neq j)]$ fonksiyonu

$$f_j(v_j, b_j) = x_j = \varphi(v_j, b_j) \quad (2.1)$$

olmak üzere, $\varphi(\cdot)$ transfer fonksiyonu ve $v_j; j$ düğümüne bir ayrıtla bağlanmış olan i düğümlerine bağlı olarak tanımlanmışsa bu j düğümüne *İşlem Elemanı (PE)* denir. Burada v_j şöyle tanımlanır:

$$v_j = \sum_{i=1}^n w_{ij} x_i \quad (2.2)$$

Yukarıdaki tanımlar incelendiğinde aslında çizge kuramındaki her bir düğümün (ayrıca başka değerler atanmadıysa) doğal olarak $I(\cdot)$ birim (identity) fonksiyon olmak üzere $w_{ij}=1$, $b_j=0$ ve $\varphi(\cdot)=I(\cdot)$ olan bir işlem elemanıdır (PE). Ayrıca istenen çıktılar tam olarak bilinen değerler olabileceği gibi istenen sınıflar veya veri kümesindeki belli izler olabileceğinden bu tanımlar hem gözetimli hem de gözetimsiz öğrenmeye uygundur (Guresen ve Kayakutlu, 2011a; 2011b).

Öğrenme Algoritması ise bir YSA'nın ağırlıklarının istenen çıktıları verilen girdilerden elde etmek üzere değiştirilmesidir (Guresen ve Kayakutlu, 2011a; 2011b).

Guresen ve Kayakutlu (2011b; Guresen ve Kayakutlu (2011a) tarafından yapılan YSA tanımı birçok çizge türü ile YSA'ları birbirinden rahatlıkla ayırabilecek özelliklere sahiptir. Guresen ve Kayakutlu (2011b) literatürde bulunan bazı YSA tanımlarını, bir şehrin elektrik dağıtım şebekesinin çizgesi üzerinde incelemiş ve önerdikleri yukarıdaki tanım hariç diğer tanımların elektrik dağıtım şebekesinin çizgesini YSA tanımı içersine aldığı belirtmişlerdir.

2.5 Yapay Sinir Ağı Modeli

Bir önceki bölümde verilen yapay sinir ağı tanımı ile yapay sinir ağı modellerinin genel yapısı açıklanmış oldu. Bu bölümde ise yapay sinir ağlarında kullanılan farklı bileşen ve parametre seçenekleri açıklanacaktır.

2.5.1 Transfer fonksiyonu türleri

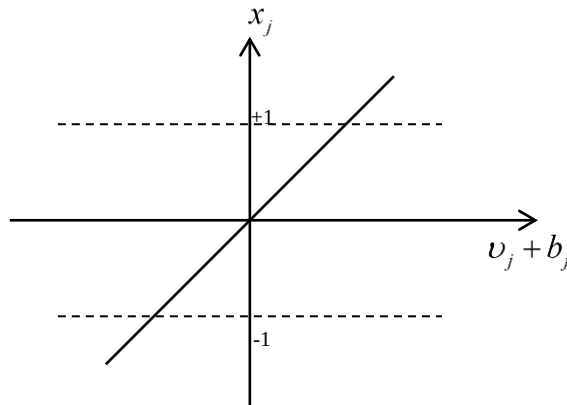
Önceki bölümde yapay sinir ağının tanımı verilmiş ve bir ağın yapay sinir ağı olarak tanımlanabilmesi için gerekli şartlar açıklanmıştı. Bu tanıma göre yapay sinir ağında

bulunması gereken üç temel bileşen başlangıç elemanı, işlem elemanı ve bitiş elemanı olarak belirtilmişti. Başlangıç elemanları girdi matrisinden bir değer alıp yapay sinir ağında bir akım başlatmaktaydılar. İşlem elemanları ve bitiş elemanları ise kendilerine gelen akışların ağırlıklı toplamını alarak bir transfer fonksiyonundan $\varphi(.)$ geçirmektedirler. Yazında rastlanan transfer fonksiyonu türlerini şöyle sıralayabiliriz (Yakut, 2012):

- Doğrusal transfer fonksiyonu
- Basamak transfer fonksiyonu
- Kutuplamalı basamak transfer fonksiyonu
- Parçalı doğrusal transfer fonksiyon
- Sigmoid transfer fonksiyonu
- Tanjant hiperbolik transfer fonksiyonu
- Sinüs transfer fonksiyonu

Doğrusal transfer fonksiyonun formülü ve grafiği (Şekil 2.1) aşağıdaki gibi verilebilir:

$$x_j = \varphi(v_j, b_j) = v_j + b_j = \sum_{i=1}^n w_{ij} x_i + b_j \quad (2.3)$$

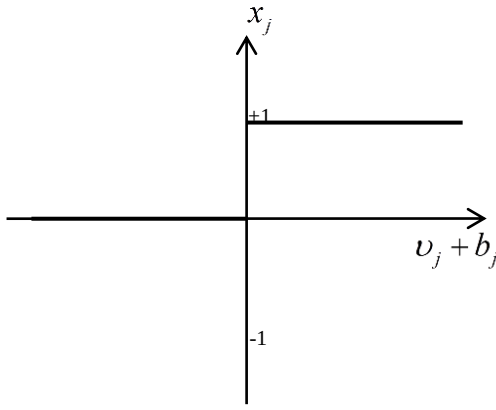


Şekil 2.1 : YSA modelinde doğrusal transfer fonksiyonu.

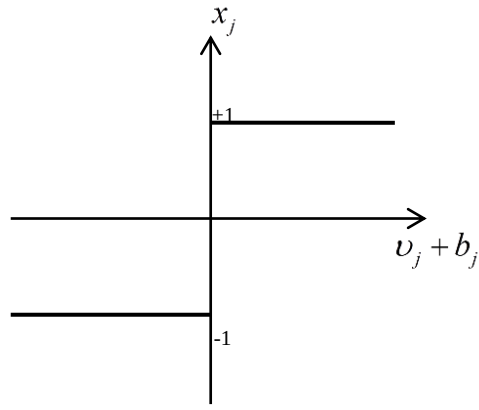
Sırasıyla tek kutuplu ve çift kutuplu basamak transfer fonksiyonunun formülü ve grafiği (Şekil 2.2) aşağıdaki gibi verilebilir:

$$x_j = \varphi(v_j, b_j) = \begin{cases} 1 & v_j + b_j \geq 0 \\ 0 & v_j + b_j < 0 \end{cases} \quad (2.4)$$

$$x_j = \varphi(v_j, b_j) = \begin{cases} +1 & v_j + b_j \geq 0 \\ -1 & v_j + b_j < 0 \end{cases} \quad (2.5)$$



(a) Tek kutuplu



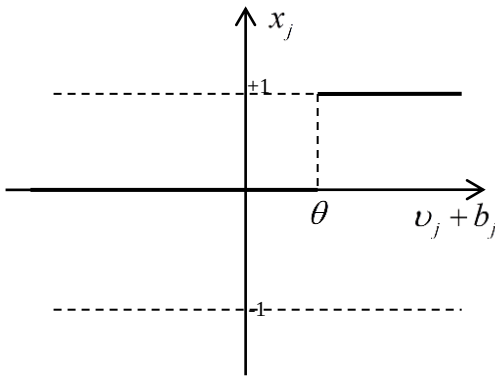
(b) Çift kutuplu

Şekil 2.2 : YSA modelinde basamak transfer fonksiyonu.

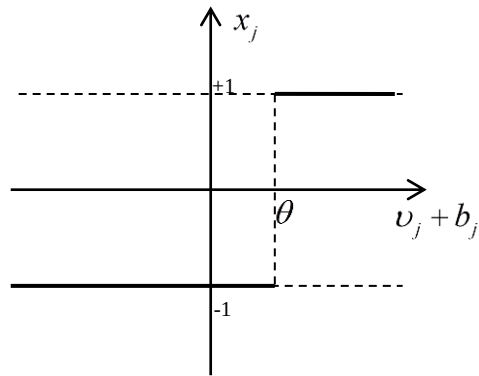
Sırasıyla tek kutuplu ve çift kutuplu basamak transfer fonksiyonunun fonksiyonlarının kutuplama değeri ile ifadesi ve grafiği (Şekil 2.3) aşağıdaki gibi verilebilir:

$$x_j = \varphi(v_j, b_j) = \begin{cases} 1 & v_j + b_j \geq \theta \\ 0 & v_j + b_j < \theta \end{cases} \quad (2.6)$$

$$x_j = \varphi(v_j, b_j) = \begin{cases} +1 & v_j + b_j \geq \theta \\ -1 & v_j + b_j < \theta \end{cases} \quad (2.7)$$



(a) Tek kutuplu

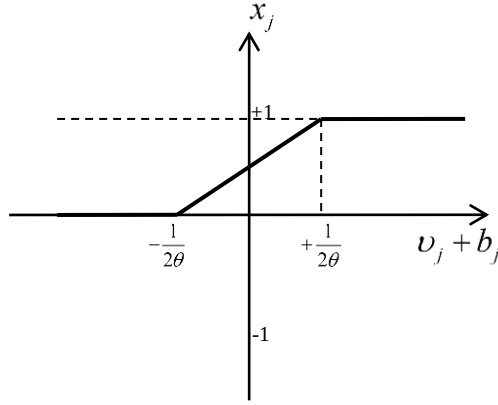


(b) Çift kutuplu

Şekil 2.3 : YSA modelinde kutuplama değerli transfer fonksiyonu.

Parçalı doğrusal transfer fonksiyonun formülü ve grafiği (Şekil 2.4) aşağıdaki gibi verilebilir:

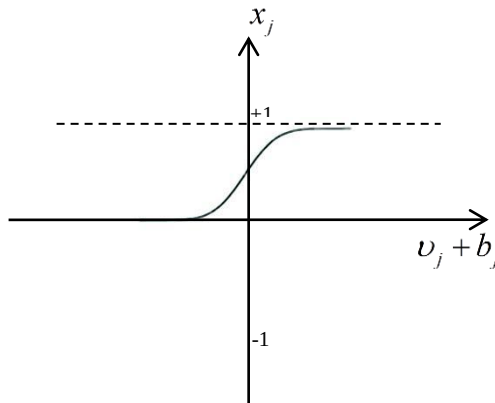
$$x_j = \varphi(v_j, b_j) = \begin{cases} 1 & v_j + b_j \geq \theta \\ \theta(v_j + b_j) + \frac{1}{2} & |v_j + b_j| < \frac{1}{2\theta} \\ 0 & v_j + b_j \leq -\frac{1}{2\theta} \end{cases} \quad (2.8)$$



Şekil 2.4 : YSA modelinde parçalı doğrusal transfer fonksiyonu.

Sigmoid transfer fonksiyonun formülü ve grafiği (Şekil 2.5) aşağıdaki gibi verilebilir:

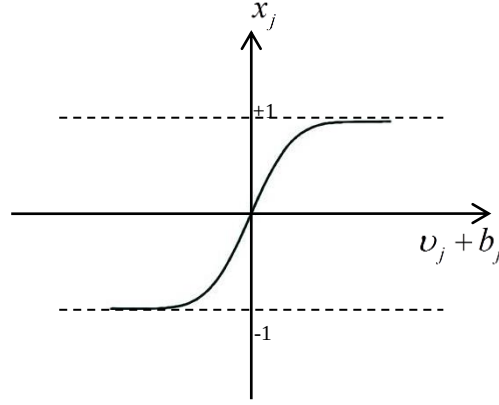
$$x_j = \varphi(v_j, b_j) = \frac{1}{1 + \exp(-\theta(v_j + b_j))} \quad (2.9)$$



Şekil 2.5 : YSA modelinde sigmoid transfer fonksiyonu.

Tanjant hiperbolik transfer fonksiyonun formülü ve grafiği (Şekil 2.6) aşağıdaki gibi verilebilir:

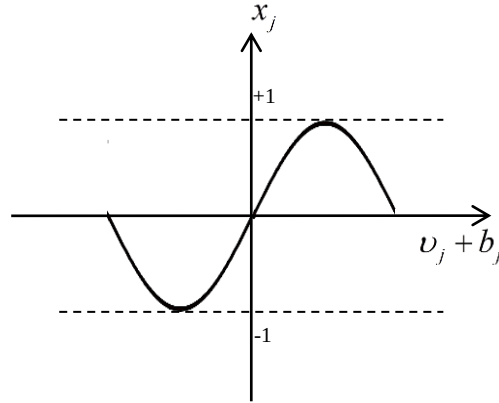
$$x_j = \varphi(v_j, b_j) = \tanh(v_j + b_j) \quad (2.10)$$



Şekil 2.6 : YSA modelinde tanjant hiperbolik transfer fonksiyonu.

Sinüs transfer fonksiyonun formülü ve grafiği (Şekil 2.7) aşağıdaki gibi verilebilir:

$$x_j = \varphi(v_j, b_j) = \sin(v_j + b_j) \quad (2.11)$$



Şekil 2.7 : YSA modelinde sinüs transfer fonksiyonu.

Transfer fonksiyonu tarafından üretilen değer yapay sinir ağındaki elemanların tepkileri ya da sonuçlarıdır. Genelde işlem elemanları bu sonuçları kendilerinden sonraki katmanda bulunan elemanlara gönderirken, bitiş elemanları ise ürettikleri sonucu yapay sinir ağının bulunduğu çevreye gönderir.

2.5.2 Kavrama - uyarlama türleri

Yapay sinir ağlarında bilgi birikimi yapay sinir ağındaki ağırlıklar ile gerçekleştirilir. Yapay sinir ağının bilgi birikimini arttırmak ve istenilen sonuçlara ulaşmak için kavrama veya uyarlama yapan algoritmalar ağırlıklar üzerinde değişiklik yaparlar.

Yapay sinir ağlarında kavrama veya uyarlama yapan algoritmalar istenilen sonuçların bilinip bilinmemesine göre aşağıdaki üç türe ayrılırlar:

- Gözetimli (supervised) kavrama
- Gözetimsiz (unsupervised) kavrama
- Takviyeli (reinforced) kavrama

Eğitim kümesi üzerindeki bütün örnekler için yapay sinir ağının ağırlıklarını bir kez güncellenmesi işlemine *devir (epoch)* denir (Munakata, 2008). Bu işlem sırasında ağırlıkların her örnek için güncellenmesine veya kümülatif olarak bir defa güncellenmesine göre kavrama/uyarlama yapan algoritmalar iki grupta incelenebilir:

- Çevrimiçi (online) kavrama
- Toplu (batch) kavrama

2.5.3 Gözetimli (supervised) kavrama

Gözetimli kavrama yapay sinir ağlarında yaygın olarak kullanılır. Gözetimli kavramanın uygulanabilmesi için yapay sinir ağıyla çözülmeye çalışılan problemdeki her bir gözlemde verilen girdilere karşılık hangi sonuçların alınması gerektiği bilinmelidir. Gözetimli kavramada amaç belli bir gözlem girdileri için istenilen çıktılar ile yapay sinir ağının ürettiği çıktı değerlerinin karşılaştırılarak hatanın azaltılmasıdır.

Bu kavrama türünde yapay sinir ağının hata miktarı en küçüklenmeye çalışılır. En küçükleme işleminde her adımda biraz hatanın biraz daha azaltılması amaçlanır. İstenilen hata seviyesine gelindiğinde veya önceden tanımlanmış belli sayıdaki ardışık adım sonucunda hata seviyesinde önemli bir değişim görülmemişse kavrama algoritması durdurulur.

2.5.4 Gözetimsiz (unsupervised) kavrama

Gözetimsiz (danışmansız) kavrama daha çok kümeleme problemleri için kullanılır. Yapay sinir ağının kavrama işlemi bittikten sonra kümeleme yöntemlerinde olduğu gibi elde edilen çıktıların ne anlama geldiğinin yorumlanması ve etiketlenilmesi kullanıcı tarafından yapılmaktadır.

Bu kavrama türünde, çok değişkenli istatistiksel analizler yöntemi olan kümeleme (cluster) analizinden farklı olarak, herhangi bir ön kabul yapmadan girdi örnekleri içerisindeki ilişkili örüntülerin dışarıdan herhangi bir müdahale olmadan, kendi kendine bulunması amaçlanmaktadır (Bose ve Liang, 1996).

Gözetimsiz kavramada, örneklerdeki parametreler arasındaki ilişkilerin yapay sinir ağı tarafından kavrama algoritmasıyla dışarıdan bir müdahale olmadan kendi kendine tespit etmesi beklenir. Bu kavramada yapay sinir ağları, adım adım mevcut yapısındaki bağlantıların ağırlıklarını mevcut örneklerle göre günceller. Böylece benzer örüntüler için benzer tepkiler / çıktılar üretmesi beklenir.

Bu nedenle gözetimsiz kavrama, ulaşılması hedeflenen çıktılarının verilmemesini / bilinmemesini esas alan bir kavrama yöntemidir. Bu kavrama türünde yapay sinir ağına yaptığı hatayı söyleyerek yapay sinir ağının öğrenmesine yardımcı olan herhangi bir danışman / öğretmen yoktur. Yapay sinir ağları bu kavrama türünde sadece girdi değerlerini kullanır.

Gözetimsiz kavramada yapay sinir ağları, performanslarını kendi iç parametrelerini kullanarak değerlendirirler. Bu öğrenmede yapay sinir ağına hiçbir şekilde bir ürettiği çıktının doğruluğu veya yanlışlığı hakkında bilgi verilmez bunu kendi yapısı ve kavrama yöntemi kuralları çerçevesinde kendisinin bulması beklenir (Vural, 2007).

Bu tip kavrama / uyarılma algoritmalarından en çok bilenen ve kullanılanı öz örgütlemeli haritalardır (SOM; Self Organizing Map). Bunun dışında “Signal Hebbian Öğrenme Kuralı” ve “Diferansiyel öğrenme kuralı” ve ART (Adaptive Resonance Theory) kavrama yöntemleri gözetimsiz kavramaya örnek olarak verilebilir.

2.5.5 Takviyeli (reinforced) kavrama

Bu kavrama türü gözetimli kavramaya ile gözetimsiz kavrama arasında bir yöntemdir. Kavrama için kullanılan örneklerde istenilen çıktı değerleri yer almaz ya da bu değerler kullanılmaz. Ancak gözetimsiz kavramadan farklı olarak algoritmanın performansı dışarıdan belirlenen bir ölçüte göre yapay sinir ağına bilgi olarak verilir. Kısaca belli girdi değerleri için elde edilen çıktı değerlerinin iyiliğini belirten bir geri bildirim kullanılır. Gözetimli kavramadaki gibi direkt olarak yapay sinir ağına hata miktarı geri bildirim olarak verilmez.

Optimizasyon problemlerini çözmek için Hinton ve Sejnowski'nin (1986) geliştirdiği Boltzmann kuralı veya Öğrenmeli Vektör Kuantlama (Learning Vector Quantization; LVQ) takviyeli kavramaya örnek olarak verilebilir (Sağıroğlu ve diğ., 2003).

Gözetimli, gözetimsiz ve takviyeli kavrama yöntemlerini ve uygulama alanlarını Çizelge 2.4'teki gibi özetlenebilir (Jain ve Mao, 1996).

Çizelge 2.4 : Kavrama algoritmalarının sınıflandırılması (Jain ve Mao, 1996).

Kavrama Türü	Kavrama Kuralı	YSA Mimarisi	Kavrama Algoritması	Kullanıldığı Yerler
Gözetimli	Hata Uyarlama	Tek ya da çok Katmanlı	Perceptron Geriye Yayılım Adaline ve Madaline	Örüntü tanıma Tahmin Kontrol
Gözetimli	Boltzmann	Geri Dönüştü	Boltzmann öğrenmesi	Örüntü Sınıflandırma
Gözetimli	Hebbian	Çok Katmanlı İleri beslemeli	Lineer diskriminant	Veri analizi Sınıflandırma
Gözetimli	Rekabetli	Rekabetli	Vektör kuantalama öğrenmesi	Veri sıkıştırma Sınıflandırma
Gözetimli	Rekabetli	ART ağı	ART Haritalaması	Örüntü Sınıflandırma Sınıflandırma
Gözetimsiz	Hebbian	İleri beslemeli Rekabetçi	Principal Component Analizi	Veri Analizi Veri Sıkıştırma

Çizelge 2.4 (devam): Kavrama algoritmalarının sınıflandırılması (Jain ve Mao, 1996).

Kavrama Türü	Kavrama Kuralı	YSA Mimarisi	Kavrama Algoritması	Kullanıldığı Yerler
Gözetimsiz	Hebbian	Hopfield Ağı	Associative Memory Öğrenmesi	Associative Memory
Gözetimsiz	Rekabetli	Rekabetli	Vektör Kuantalaması	Sınıflandırma Veri Sıkıştırma
Gözetimsiz	Rekabetli	Kohonen SOM	Kohonen SOM	Sınıflandırma Veri Analizi
Gözetimsiz	Rekabetli	ART ağı	ART1, ART2	Sınıflandırma
Takviyeli	Hata uyarlama ve Rekabetli	Radyal Tabanlı	Radyal Tabanlı Öğrenme	Sınıflandırma Örüntü Tanıma Fonksiyon Yaklaşımı Tahmin Kontrol

Hata uyarlama kullanan kavrama metodları yapay sinir ağının ürettiği sonuç ile üretmesi gereken sonuç arasındaki farkı; yani hata miktarını dikkate alırlar. Bu kavrama kuralları hatayı yapay sinir ağının ağırlıklarına dağıtır ve ağırlıkları düzelterek hata miktarını azaltırlar.

Rekabete dayalı kavrama metodları yapay sinir ağının gizli katmanındaki işlem elemanlarının eğitimde kullanılan gözlemler için birbirleri ile rekabet etmesine ve kazanan işlem elemanın (ya da işlem elemanlarının) ağırlıklarının tekrar düzenlenmesine dayanır.

Boltzmann kavrama kuralı stokastik sinir hücrelerinden oluşan stokastik ikili makinelerin (stochastic binary machine) benzerlik fonksiyonu (likelihood function) en büyükmeye dayanır (Haykin, 1999).

Hebbian kavrama iki temel ilkeye dayanır. Bu ilkelerden ilki bir sinapsın iki ucundaki sinir hücreleri aynı anda aktive oluyorsa bu sinapsın güçlendirilmesi gerektiğidir. İkinci ilke ise birinci ilkenin tam tersi durumda; bir sinapsın iki ucundaki sinir hücreleri farklı zamanlarda aktive oluyorsa bu sinapsın zayıflatılması veya yok edilmesi gerektiğidir.

ART (Adaptive Resonans Theory) kavrama kuralı girdiler ile çıktı sinir hücrelerinin değerlerinin benzerliğine göre yapının değiştirilmesine dayanır. Eğer girdiler ile sinir hücrelerinin değerleri birbirine belli bir eşikten daha yakınsa, kazanan sinir hücresinin değerleri girdiye yakınlaştırılır. Eğer hiçbir sinir hücresi girdiye belli bir eşik değerden daha fazla benzemiyorsa girdiye benzeyen yeni bir sinir hücresi eklenir.

Radyal tabanlı fonksiyon ağırları m_0 boyutlu doğrusal olmayan yöntemlerle sınıflara ayrılabilen verinin, doğrusal olmayan bir dönüşümle m_1 boyutlu doğrusal olarak sınıflarına ayrılabilen bir veriye dönüştürmeye dayanır. Bu maksatla gizli katmandaki her bir işlem elemanı bir küme merkezinin temsil eder ve çan şeklindeki fonksiyonlar kullanılarak işlem yapılan gözlem ile temsil edilen küme merkezi arasındaki mesafeye bağlı bir çıktı üretilir. Gizli katmandan çıkan sonuçlar doğrusal olarak sınıflara ayrılabilir olduğundan, gizli katman ile çıktı katmanı arasındaki ağırlıklar doğrusal denklem sistemleri kullanılarak hesaplanır.

Çevrim içi kavrama, eğitim kümesindeki her bir örnek için ağırlık değişim miktarı hesaplandıktan sonra değişimin hemen uygulanmasıdır. Bu yöntemin avantajları arasında değişiklik hemen yapıldığı için fazladan bellek kullanımına ihtiyaç duyulmaması ve programlama yönünden daha kolay olması sayılabilir.

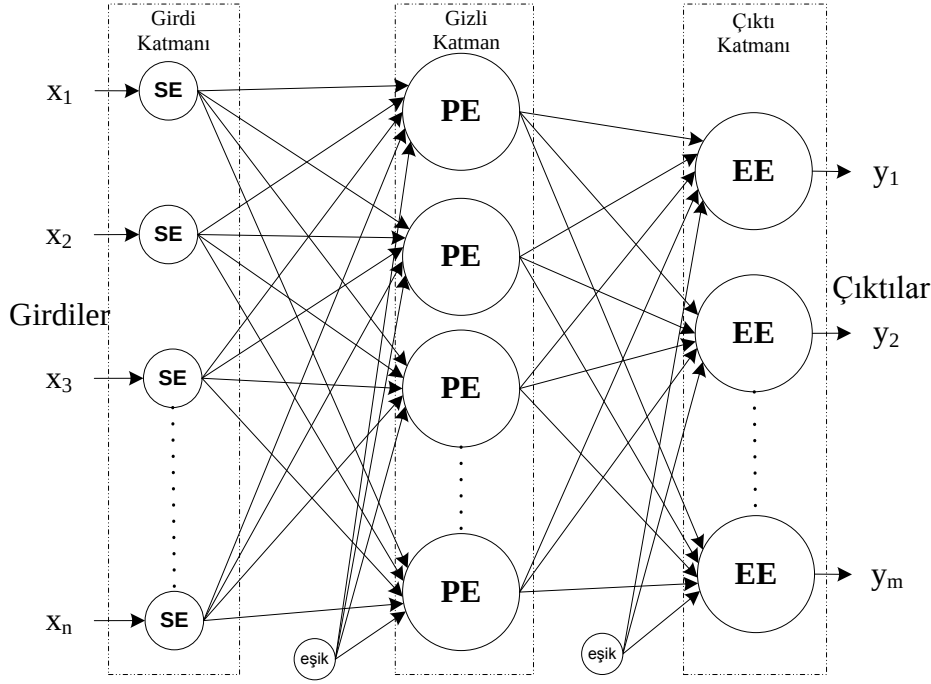
Toplu (batch) kavrama yönteminde ise tüm ağırlık değişiklikleri kaydedilir ve devir sonunda kümülatif bir değişim yapılır. Bu yöntemde amaç, örneklerin çoğunluğunu kapsayan bir ağırlık değişimi yapmaktır.

2.6 Çok Katmanlı Algılayıcılar (ÇKA)

Çok katmanlı algılayıcılar (Multi-Layer Perceptron) çok kullanılan ve günümüzde yapay sinir ağı dendiğinde ilk akla gelen yapılardır.

Çok katmanlı algılayıcılar (ÇKA) girdi ve çıktı katmanları arasında en az bir gizli katman kullanan yapıdır. Bu yapı girdi değerleri ile çıktı değerleri arasındaki doğrusal olmayan ilişkileri de tespit edebilmektedir. Temel olarak girdi katmanı, dış ortamdan gelen bilgileri alıp gizli katmana iletmede rol alır. Bu bilgiler gizli katmanda işlenir ve değişim işleminden sonra bir sonraki katmana iletilir. Bir sonraki katman başka bir gizli katman olabileceği gibi çıktı katmanı da olabilir. Çıktı katmanı ise dış ortama sonuçlar / tepkiler üretir.

Çok katmanlı algılayıcılar için tek gizli katmanlı örnek Şekil 2.8’de verilmiştir. Minsky ve Papert’in (1969) doğrusal olmayan bir ilişki gösteren “Özel Veya” (XOR) probleminden yola çıkarak doğrusal olmayan ilişkileri yakalamadaki başarısızlıklarını göstermesiyle algılayıcılar (perceptron) üzerindeki çalışmalar azalmıştır (Michie ve diğ., 1994). Bu başarısızlık üzerine çok katmanlı algılayıcılar (ÇKA) ve yeni öğrenme algoritmaları geliştirilmiştir.



Şekil 2.8 : YSA’da çok katmanlı algılayıcı yapısı.

ÇKA yapısının uygun parametrelerle sürekli çok değişkenli bir fonksiyonu modelleyebilmesinin ispatlanması tekrar bu alandaki çalışmaları hızlandırmıştır (Cybenko, 1989; Hornik ve diğ., 1989; Funahashi, 1989; Hornik ve diğ., 1990; Funahashi, 1990; Hornik, 1991; Hornik, 1993). Böylece yapay sinir ağları tarihinde önemli bir devreye girilmiş ve birçok çalışmada ÇKA’nın farklı kavrama algoritmalarının uygulanmasına imkân sağlaması nedeniyle tercih edilmiştir.

Çok katmanlı algılayıcılarda kavrama algoritmalarına hata yayma modeli veya geriye yayılım algoritması örnek verilebilir. Genelde Delta kuralı olarak adlandırılan kavrama kuralının yaygın olarak kullanılmasıyla bir çok yeni kavrama algoritması ÇKA yapısı üzerinde geliştirilmiştir.

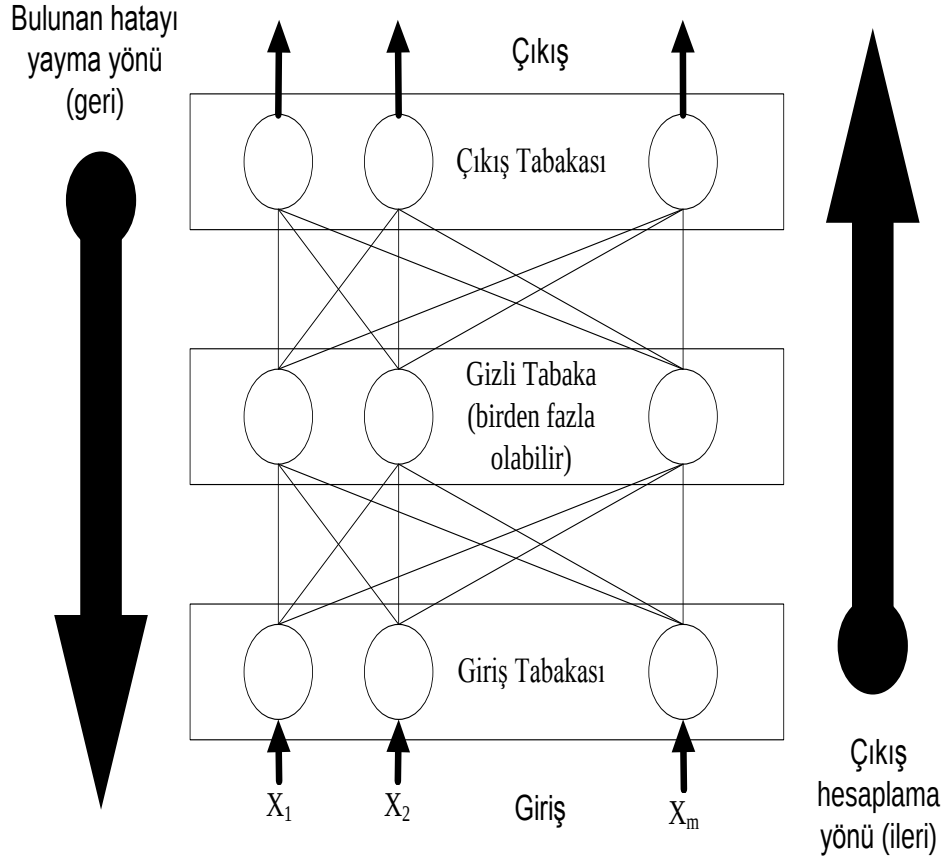
Delta kuralı, en küçük kareler yöntemi kullanarak, yapay sinir ağının hesapladığı çıktı değeri ile beklenen çıktı değeri arasındaki farkı azaltmaya dayanır.

2.6.1 Geri yayılım algoritması

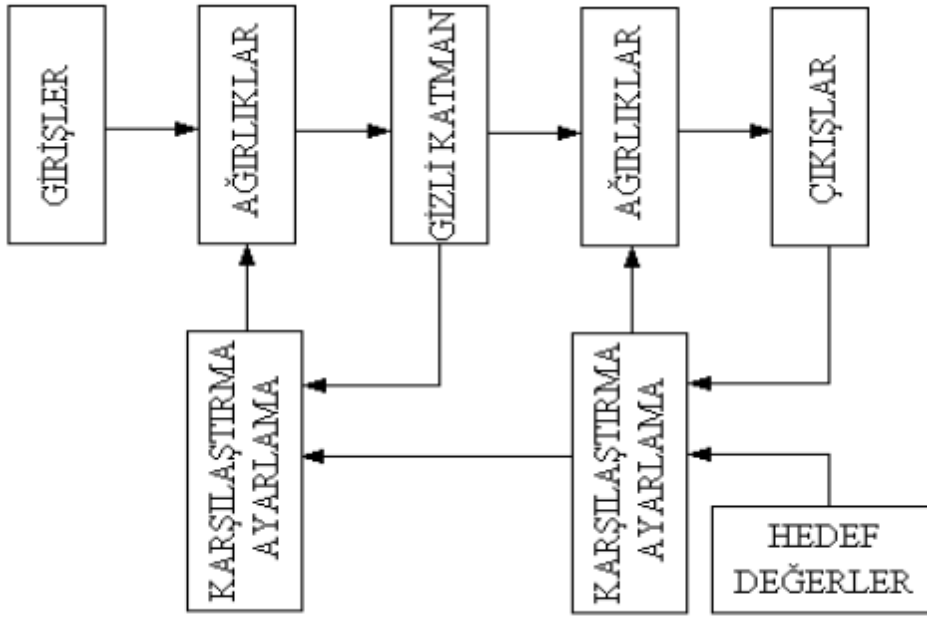
Çok katmanlı algılayıcılarda özellikle tercih edilen geri yayılım algoritması (GYA) temel öğrenme algoritması olarak adlandırılabilir. Bu algoritmaya dayanarak momentum ve adaptif kavrama oranı gibi değişik kavramların eklenmesiyle yeni algoritmalar geliştirilmiştir. Geri yayılım algoritmasında kavrama 3 temel aşamadan oluşur (Ocakoğlu, 2006):

- Girdi değerinin ileri beslenmesi,
- İlişkili hatanın geri yayılımı,
- Bağlantı ağırlıklarının güncellenmesi.

Bu üç adım Şekil 2.9'da ve Şekil 2.10'da gösterilmiştir.



Şekil 2.9 : İleri beslemeli geri yayılım yapısı (Gündoğdu, 2007).



Şekil 2.10 : Hatanın geriye yayılım akış diyagramı (Ocakoğlu, 2005).

Geri yayılım algoritması için t iterasyonunda j 'inci bitiş elemanına ait hata sinyali $e_j(t)$ ve anlık hata enerjisi $\xi_j(t)$, istenen çıktı $o^{des.}(t)$ ve hesaplanan çıktı $y_j(t)$ için aşağıdaki gibi tanımlanabilir (Haykin, 1999):

$$e_j(t) = o^{des.}(t) - y_j(t) \quad (2.12)$$

$$\xi_j(t) = \frac{1}{2} e_j^2(t) \quad (2.13)$$

$$\xi(t) = \sum_j \xi_j(t) = \frac{1}{2} \sum_j e_j^2(t) \quad (2.14)$$

Hesaplanan çıktı $y_j(t)$ ise j 'inci bitiş elemanı için aşağıdaki gibi ifade edilebilir:

$$v_j(t) = \sum_{i=0}^n w_{ji}(t) y_i(t) \quad (2.15)$$

$$y_j(t) = \varphi(v_j(t)) \quad (2.16)$$

Burada n gelen girdi sayısı, w_{ji} girdilerin ağırlıkları, w_{j0} ise sabit sinyal üreten ($y_0 = +1$) eşik değerinin ağırlığıdır. Hatanın ağırlığa göre değişimini kısmi türev ve zincir kuralı kullanılarak aşağıdaki gibi ifade edilebilir:

$$\frac{\partial \xi(t)}{\partial w_{ji}(t)} = \frac{\partial \xi(t)}{\partial e_j(t)} \frac{\partial e_j(t)}{\partial y_j(t)} \frac{\partial y_j(t)}{\partial v_j(t)} \frac{\partial v_j(t)}{\partial w_{ji}(t)} \quad (2.17)$$

Denklem 2.16'daki kısmî türevler yukarıda verilen denklemler yardımıyla hesaplanabilir. Denklem 2.14'te $e_j(t)$ ye göre kısmî türev alındığında aşağıdaki denkleme ulaşılır:

$$\frac{\partial \xi(t)}{\partial e_j(t)} = e_j(t) \quad (2.18)$$

Denklem 2.12'de $y_j(t)$ ye göre kısmî türev alındığında aşağıdaki denkleme ulaşılır:

$$\frac{\partial e_j(t)}{\partial y_j(t)} = -1 \quad (2.19)$$

Denklem 2.16'de $v_j(t)$ ye göre kısmî türev alındığında aşağıdaki denkleme ulaşılır:

$$\frac{\partial y_j(t)}{\partial v_j(t)} = \varphi'_j(v_j(t)) \quad (2.20)$$

Denklem 2.15'te $e_j(t)$ ye göre kısmî türev alındığında aşağıdaki denkleme ulaşılır:

$$\frac{\partial v_j(t)}{\partial w_{ji}(t)} = y_i(t) \quad (2.21)$$

Denklem 2.18'den Denklem 2.21'e kadar olan denklemler Denklem 2.17'de yerine yazıldığında ise aşağıdaki denkleme ulaşılır:

$$\frac{\partial \xi(t)}{\partial w_{ji}(t)} = -e_j(t) \varphi'_j(v_j(t)) y_i(t) \quad (2.22)$$

Hataları azaltmak için uygulanacak ağırlık değişimi $\Delta w_{ji}(t)$ ise kavrama oranına (η) bağlı olarak şöyle tanımlanabilir (Haykin, 1999):

$$\Delta w_{ji}(t) = -\eta \frac{\partial \xi(t)}{\partial w_{ji}(t)} \quad (2.23)$$

Başka bir ifadeyle hatayı azaltmak için uygulanacak ağırlık değişimi :

$$\Delta w_{ji}(t) = \eta e_j(t) \phi'_j(v_j(t)) y_i(t) \quad (2.24)$$

$$\Delta w_{ji}(t) = \eta \delta_j(t) y_i(t) \quad (2.25)$$

olarak ifade edilebilir. Böylece geri yayılım algoritmasındaki ağırlık değişimleri şu şekilde özetlenebilir (Haykin, 1999):

$$\begin{pmatrix} \text{Ağırlık} \\ \text{düzeltmesi} \\ \Delta w_{ji}(t) \end{pmatrix} = \begin{pmatrix} \text{öğrenme} \\ \text{oranı} \\ \eta \end{pmatrix} \times \begin{pmatrix} \text{yerel} \\ \text{gradyan} \\ \delta_j(t) \end{pmatrix} \times \begin{pmatrix} \text{gelen} \\ \text{sinyal} \\ y_i(t) \end{pmatrix} \quad (2.26)$$

Gizli katmana (işlem elemanlarına) gelen ağırlıkların güncellenmesi de aynı şekilde yapılır. Bu güncellemeler de j 'inci işlem elemanının gradyanı ise şu şekilde hesaplanır (Haykin, 1999):

$$\delta_j(t) = \phi'_j(v_j(t)) \sum_k \delta_k(t) w_{kj}(t) \quad (2.27)$$

Geri yayılım algoritmasının yakınsama hızını arttırmak için momentum (α) yöntemi şu şekilde kullanılır:

$$\Delta w_{ji}(t) = \alpha \Delta w_{ji}(t-1) + \eta \delta_j(t) y_i(t) \quad (2.28)$$

Geri yayılım algoritmasında ağırlıkların yakınsamasını hızlandırmak için kullanılan bir diğer yöntem de kavrama oranının (η) kavramanın hızına göre değiştirilmesidir. Bunun için hata düşüş hızı arttıkça kavrama oranı sabit bir miktar artırılırken hata düşüş hızı yavaşladığında ise kavrama oranı geometrik olarak azaltılır. Bu değişim aşağıdaki denklemle ifade edilebilir (Alpaydın, 2004):

$$\Delta \eta = \begin{cases} +a & E^{t+c} < E^t \\ -b\eta & \text{diğ.} \end{cases} \quad (2.29)$$

Denklem 2.29’da a , b ve c sabit olup E^t ise son t iterasyondaki hata maliyetinin ortalamasıdır.

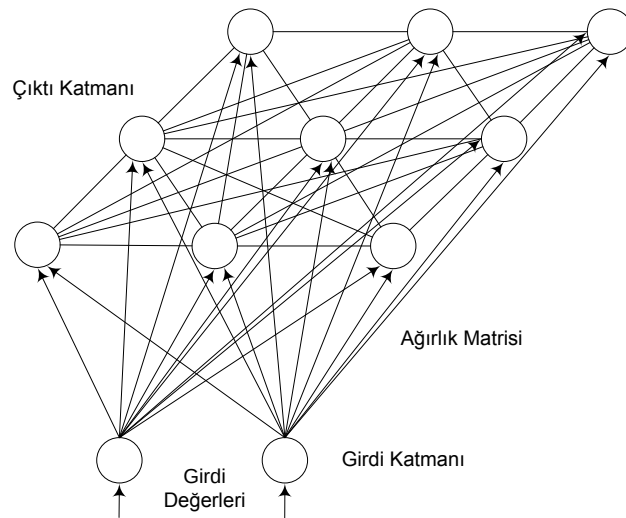
2.7 Öz-örgütlemeli Haritalar

Öz-örgütlemeli haritalar (Self Organizing Map; SOM) gözetimsiz kavrama kullanan yapay sinir ağı yapılarıdır. Bu yapılara Malsburg (1973), Hopfield (1982) ve Kohonen (1982a) tarafından geliştirilen yapılar örnek verilebilir. Zaman içerisinde Kohonen (1982b, 1984, 1988, 1989, 1991, 1993a, 1993b) tarafından geliştirilen öz-örgütlemeli haritalar (ÖÖH) araştırmacılar tarafından benimsenmiş ve ilk akla gelen ÖÖH yöntemi olmuştur. Bu çalışmada da öz örgütlemeli harita yapısı olarak Kohonen’in geliştirdiği ÖÖH kullanılmaktadır.

2.7.1 Kohonen öz-örgütlemeli haritası

Kohonen öz örgütlemeli haritası, beynin Neocortex tabakasında yaygın olan duysal haritalardan esinlenerek geliştirilmiş bir yapay sinir ağı yapısıdır (Kohonen, 1972). Özellikle kümeleme problemlerinin çözümünde yaygın olarak kullanılır.

Bu ağ bir girdi ve bir de çıktı katmanı olmak üzere iki katmandan oluşur. Girdi katmanında girdi boyutu (sayısı) kadar başlangıç elemanı (SE) yer alır. Çıktı katmanı ise iki boyutlu bir katmanda yer alan bitiş elemanlarından (EE) oluşur. Normalden farklı olarak iki boyutlu bir çıktı katmanı olduğundan bitiş elemanlarının sıralaması ve komşulukları buna göre değerlendirilir. Kohonen ağının yapısı Şekil 2.11’de gösterilmiştir.



Şekil 2.11 : Kohonen öz örgütlemeli harita yapısı (Frohlich, 1997).

Kohonen öz örgütlemeli haritası için öğrenme adımlarını Çizelge 2.5'teki gibi sıralamak mümkündür. Kohonen öz örgütlemeli haritası gözetimsiz ve rekabetçi bir öğrenme ile eğitilir.

Çizelge 2.5 : Kohonen ağında kavrama işlem adımları.

İşlem Adımları	Açıklama
Adım 1	Çıktı katmanında yer alan bitiş elemanlarının bütün referans vektörler rassal olarak atanır.
Adım 2	Bir giriş deseni belirlenir.
Adım 3	Bitiş elemanları arasından seçilen desene en yakın olan bitiş elemanı yarışmayı kazanan eleman olarak belirlenir.
Adım 4	Kazanan bitiş elemanı komşusu olan bitiş elemanlarının ağırlıkları değiştirilir.
Adım 5	Önceden tanımlanmış durma koşullarına göre belirgin bir değişiklik oldukça Adım 2'den itibaren adımlar tekrarlanır.

Kazanan bitiş elemanı giriş desenine en yakın referans vektörüne sahip elemandır. Bu durumda t 'inci tekrarda $\mathbf{w}_j(t)$ ağırlık vektörü, $\mathbf{x}(t)$ giriş desen vektörü ve $i(\mathbf{x})$ kazanan bitiş elemanı olmak üzere en kısa uzaklıktaki kazanan bitiş elemanı aşağıdaki gibi tanımlanabilir:

$$i(\mathbf{x}) = \arg \min_j \|\mathbf{x}(t) - \mathbf{w}_j(t)\|, \quad j = 1, 2, \dots, n \quad (2.30)$$

Kazanan bitiş elemanı ve komşuları için ağırlıklar aşağıdaki gibi güncellenir:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta(t) h_{j,i(\mathbf{x})}(t) (\mathbf{x}(t) - \mathbf{w}_j(t)) \quad (2.31)$$

Burada t 'inci tekrarda $\eta(t)$ kavrama oranı, $h_{j,i(\mathbf{x})}(t)$ komşuluk fonksiyonudur.

$h_{j,i(\mathbf{x})}(t)$ kazanan bitiş elemanı için “1” değerini alırken tanımlanan komşuluk türüne göre komşular için “0” a doğru gider. Komşu olarak tanımlanmamış bitiş elemanları için “0” değerini alarak komşu olmayan bitiş elemanlarının ağırlıklarını güncellemesini engeller. Komşuluk ilişkileri kavrama sırasında ağırlık vektörünün güncellenmesiyle zamanla değişebilir.

3. YSA YAPILANDIRMA ALGORİTMALARI

Verilen dinamik yapay sinir ağları tanımına uyan yapay sinir ağlarının literatürde bulunup bulunmadığını incelemek için yapay sinir ağlarının yapısını (SE, PE, EE) değiştiren veya sayılarını tespit etmede kullanılan algoritmaları irdelemek gerekmektedir. Literatürdeki algoritmaları yapıcı (constructive) algoritmalar, budama (pruning) algoritmaları, düzenleme (regularization) algoritmaları ve evrimsel (evolutionary) algoritmalar olarak sınıflayabiliriz (Kwok ve Yeung, 1997a).

3.1 Yapıcı Algoritmalar

Yapıcı algoritmalar görece küçük YSA yapılarıyla başlayıp sistematik olarak bu yapıları büyüten algoritmalarlardır. Bu algoritmaların özellikleri ayrıntılı olarak daha önceki yayınlarımızda verilmiştir (Güreşen ve Kayakutlu, 2012; 2013b).

Yapısal büyüme sağlaması nedeniyle yapıcı algoritmalar dinamik sinir ağlarıyla kolayca karıştırılabilirler. Ayrıca bu algoritmalarından bir kısmı yapılan tanımlara uygun olarak dinamik yapay sinir ağı oluşturma potansiyeline de sahip olduğundan ayrıntılı olarak irdelenmelidir. İlerleyen kısımlarda bu tip yapıcı algoritmalar irdelenecek ve en az bir yeni bitiş elamanını (EE) mevcut yapıya ekleyip eklemedikleri incelenecektir.

3.1.1 Dinamik düğüm yaratma algoritması

Dinamik Düğüm Yaratma (Dynamic Node Creation) algoritmaları ilk olarak Ash (1989) tarafından ortaya atılmıştır. Temelde bu algoritmalar gizli katmana her seferinde yeni bir PE ekler ve tüm YSA tekrar eğitilir (Kwok ve Yeung, 1997a). Azimi-Sadjadi ve diğ. (1993) hem ağırlıkları güncelleyen hem de gizli katmandaki PE sayısını tek tek arttıran bir algoritma oluşturmuştur. Khorasani ve Weng (1994) ise bir budama algoritması ile dinamik düğüm yaratma algoritmasını karşılaştırmıştır. Zhang (1994) dinamik düğüm yaratma benzeri SELF adındaki algoritmayı geliştirmiştir ve bu algoritma ile en iyi gizli katman PE sayısını bulmaya çalışmaktadır.

3.1.2 İzdüşüm takip regresyonu

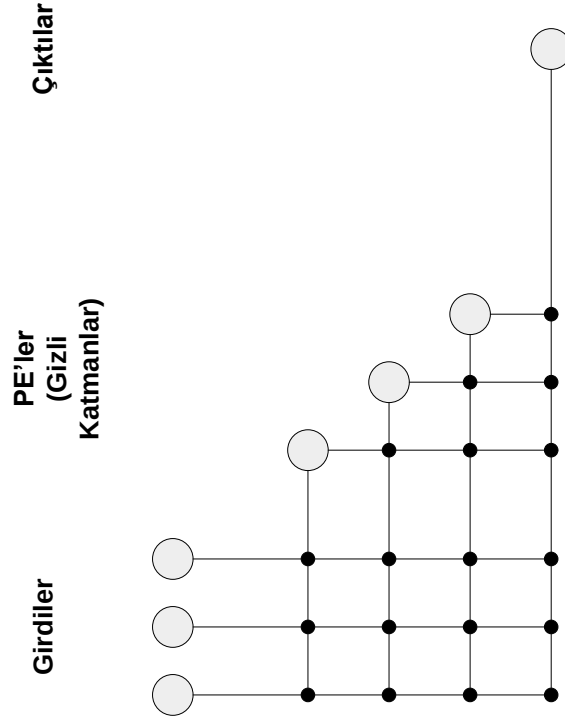
İzdüşüm Takip Regresyonu (Projection Pursuit Regression) algoritmaları çıkış olarak Friedman ve Stuetzle (1981) bu konudaki çalışmaları temel alırlar. Bu algoritmalar çok değişkenli veriyi iki katmanlı bir ağ yardımıyla çok boyutlu bir uzaydan izdüşümlerle daha az boyutlu bir uzaya taşırlar (Hwang ve diğ., 1994). Mevcut yazında Lay ve diğ. (1994) gizli katmandaki bir işlem elemanını dahil etmeden önce bir transfer fonksiyonu havuzu oluşturup yeni işlem elemanını bu havuzdaki polinomlardan seçmişlerdir. Saha ve diğ. (1993) ise radyal temelli fonksiyonlar kullanılan problemlerin gözlem/boyut oranını düşürmek için çoklu lineer izdüşüm kullanmışlardır. Shin ve Ghosh (1995) Ridge Polinom Ağlarını, izdüşüm takip regresyonunun özel bir durumu, her zaman aynı fonksiyonu kullanan versiyonu olarak özetlemiştir. Verkooijen ve Daniels (1994) izdüşüm takip regresyonunda kullanılacak fonksiyonları ayrı ayrı yapay sinir ağı olarak ele almış ve bu ağları eğiterek uygun fonksiyonu bulmuştur. Verkooijen ve Daniels (1994) algoritması ilk bakışta farklı yapay sinir ağlarının ortaklaşa çalışması ve yeni bitiş elemanları (EE) eklemesi gibi gözüксе de ilk yapay sinir ağı açısından yeni bir bitiş elemanı eklememiş ve ilk yapay sinir ağına yeni bir işlev kazandırmamıştır.

3.1.3 Basamak-korelasyon algoritması

Basamak-korelasyon algoritması (Cascade-Corelation Algorithm) ilk olarak Fahlman ve Lebiere (1990) tarafından geliştirilmiştir. Bu algorithmada her seferinde yeni bir PE eklenmektedir. Bu yeni eklenen PE diğer tüm gizli katmandaki PE'lere bağlanmaktadır. Böylece aslında her eklenen PE, bir PE'den oluşan bir gizli katman oluşturarak (Kwok ve Yeung, 1997a) Şekil 3.1'de görülen basamak yapısını oluşturmaktadır. Bu algorithmada Fahlman ve Lebiere (1990) her seferinde bir adet PE ekleyip onu sabitlemekte ve ekleyecekleri PE'yi de kalan hata ile en yüksek korelasyona sahip çıktı üreten PE olarak seçmektedirler.

Fang ve Lacher (1994), basamak-korelasyon algoritmasını ufak değişikliklerle kullanmışlardır. Phatak ve Koren (1994), basamak-korelasyon algoritmasını her bir gizli katman birden fazla PE alabilecek şekilde değiştirmişler; böylece tüm SE'lerin (başlangıç elemanı) tüm PE'lere bağlantısı olmamasını sağlamışlardır. Sjogaard (1992) ikinci dereceden basamak-korelasyon algoritması kullanarak eklenen tüm PE'lerin aynı gizli katmanda ve birbirlerinden etkilenmeyecek (birinin çıktısı

diğerinin girdisi olmayacak) şekilde toplamıştır. Kwok ve Yeung (1997b) farklı korelasyon tabanlı fonksiyonları kullanarak farklı zaman ve uzay ihtiyaçlarını karşılaştırarak performans analizleri yapmışlardır.



Şekil 3.1 : Basamak-Korelasyon yapısı (Kwok ve Yeung, 1997a).

Şekil 3.1’de görülen yapıda içi boş ve büyük daireler düğümleri (SE’leri, PE’leri ve EE’leri) göstermektedir. Yatay çizgiler düğümlerin çıktılarını göstermektedir. İçi dolu noktalar ise bulundukları yatay çizgideki düğümün çıktısının dikey çizgiyle bağlandıkları düğüme girdi olarak verileceğini göstermektedir. Böylece dikey çizgi boyunca yataydaki düğümlerin çıktıları ağırlıklarıyla çarpılarak toplanır ve dikeyde bulunan düğümlere girdi olarak verilir.

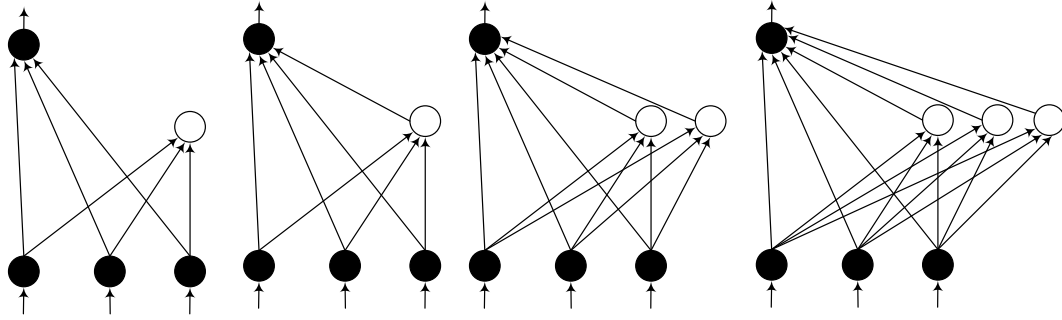
Basamak-korelasyon temelli algoritmalar incelendiğinde açıkça görülmektedir ki bu algoritmalar SE ve EE’leri en baştan sabitleyip aradaki PE’leri ve bunların bağlantılarını tespit etmektedirler.

3.1.4 Kaynak dağıtım ağları

Kaynak dağıtım ağları (Resource Allocation Networks; RAN) her seferinde bir adet PE’yi gizli katmana eklemektedirler ve diğer yapıcı algoritmalardan temel farkı öğrenme izlerini hatırlayarak bunları kullanmalarındadır (Kwok ve Yeung, 1997a). Sadece gizli katmandaki PE’leri artırarak ya da azaltarak adaptasyon yeteneğini artırır.

İlk defa Platt (1991) tarafından ortaya atılan bu algoritma iki katmanlı bir radyal tabanlı fonksiyon (RTF) ağıdır (Li ve diğ., 2010).

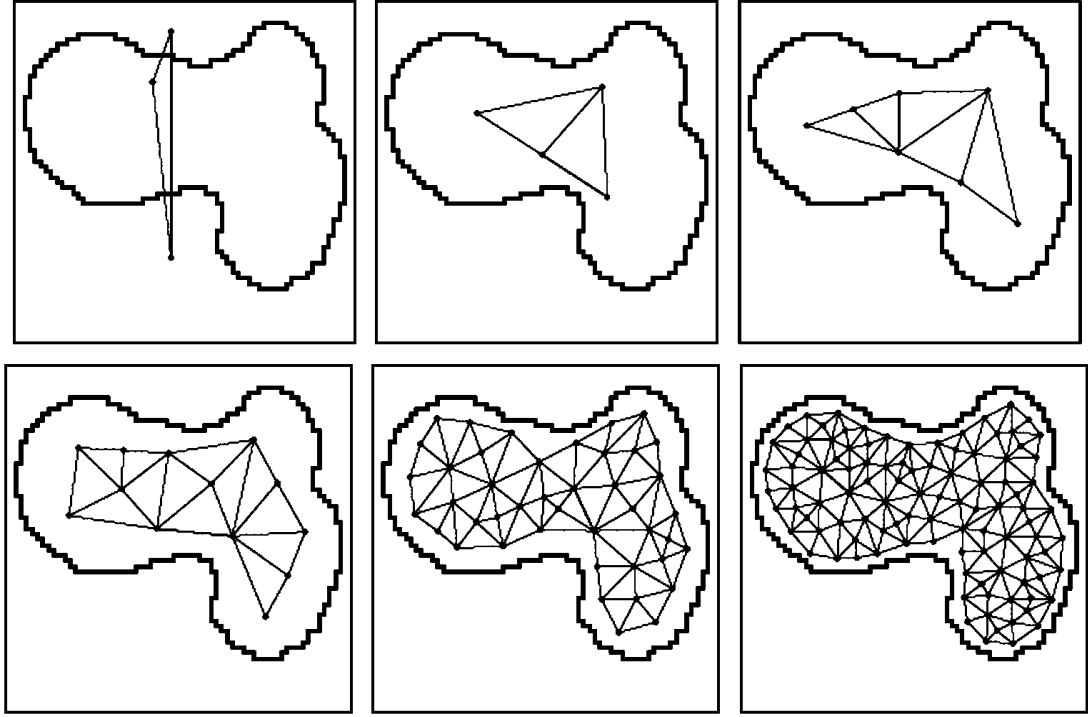
Şekil 3.2’de görülen örnekte 3 girdi (başlangıç elemanı, SE) ve bir çıktı (bitiş elemanı, EE) vardır. Tüm SE’ler mevcut EE’ye sürekli olarak bağlıdır. Bunun yanında kaynak dağıtım algoritması gizli katmana radyal tabanlı fonksiyona sahip bir PE ekler. Buradaki beklenti PE’nin eğitim kümesindeki verilerden aynı sınıfa ait bir kümeyi kalanlardan ayırt edebilmesidir. PE’ler yerel olarak girdi-çıkı eşlemesini yaparken EE ise global olarak girdi-çıkı eşlemesini yapmaktadır (Li ve diğ., 2010). RTF ve ileride anlatılacak sınırlandırılmış coulomb enerji ağından farkı ise girdi olarak hem SE’lerden gelen bağlantıyı kullanması hem de PE’lerden gelen bağlantıyı kullanmasıdır.



Şekil 3.2 : Kaynak dağıtım ağlarının oluşturulması (Li ve diğ., 2010).

Fritzke (1993; 1994a; 1994b) yaptığı çalışmada hem gözetimli hem de gözetimsiz eğitim için büyüyen hücre yapıları (growing cell structures) adlı algoritmayı geliştirmiştir. Daha sonra Fritzke (1995a; 1995b) bu algoritmayı büyüyen sinir gaz ağları (growing neural gas network) ve büyüyen ızgara (growing grid) olarak adlandırmıştır.

Kaynak dağıtım ağları içerisinde sınıflandırabileceğimiz (Kwok ve Yeung, 1997a) bu algoritmaların temel farkı Şekil 3.3’te adımlar da görüleceği gibi bir uzayı kaplayıncaya kadar PE eklemesidir. Her eklemekten sonra oluşan yapı tekrar eğitilmekte ama zaten eğitilmiş olan hücrelerin parametreleri bir önceki eğitimden kaldığı şekilde bırakılması ve yeniden rassal olarak oluşturulmasıdır. Daha önce eğitilen kısım sıfırlanmadığı için kaynak dağıtım ağları kategorisinde değerlendirilmektedir.



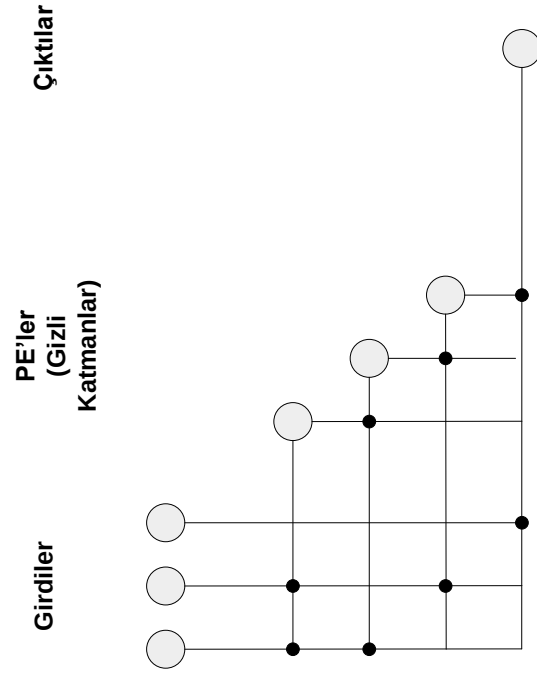
Şekil 3.3 : Büyüyen hücre yapısı örneği (Fritzke, 1993).

3.1.5 Grup metoduyla veri işleme

Grup metoduyla veri işleme (Group Method of Data Handling) Ivakhnenko'nun (1984) çalışması temel alınarak geliştirilmiş metotlardır. Diğer algoritmalarından farkı her katmanda sabit miktarda bağlantı olması ama bu bağlantıların girdi ve mevcut PE'lerin çıktılarından herhangi bir kombinasyonundan oluşmasıdır (Kwok ve Yeung, 1997a). Dolayısıyla her bir PE ekleme işlemi birçok ağ alternatifini gündeme getirir. Bu alternatif ağ seçimi sırasında çok farklı yöntemler kullanılabilir.

Tenorio ve Lee (1990) bu seçimi benzetim tavlama kullanarak yapmışlardır. Parker ve Tummala (1992) ise belli bir değerin üzerinde performans gösteren PE'lerin hepsini ekleyerek yapmışlardır.

Şekil 3.4'te grup metoduyla veri işleme yapısı örneği verilmiştir. Şekilden de görüleceği üzere her PE sadece sabit miktarda girdi almakta ve bu işlem kendisinden önceki PE'lerin çıktılarıyla SE'lerin (başlangıç elemanlarının) çıktılarından farklı kombinasyonları olarak gerçekleşmektedir.

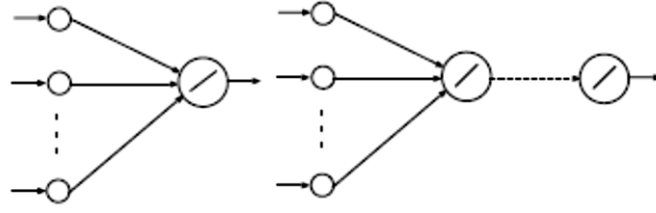


Şekil 3.4 : Grup metoduyla veri işleme yapısı örneği (Kwok ve Yeung, 1997a).

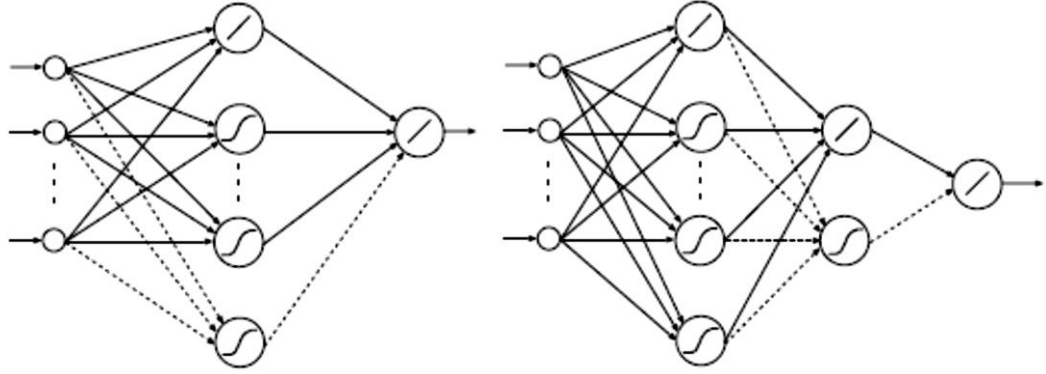
3.1.6 Ma ve Khorasani'nin yapıcı stratejisi

Ma ve Khorasani (2003) tarafından yeni bir yapıcı strateji YSA'larının yapısını belirlemek için önerilmiştir.

Ma ve Khorasani (2003) tarafından önerilen algoritma öncelikle tüm girdileri lineer transfer fonksiyonuna sahip bir EE ile eğitmektedirler (Şekil 3.5, a). Eğer oluşan hata istenilen düzeyden daha düşük değilse Şekil 3.5 (b)'de görüldüğü gibi eski EE arada kalacak ve PE olacak şekilde yeni bir lineer transfer fonksiyonuna sahip bir EE ağı eklenmektedir. Daha sonra Şekil 3.6 :(a)'da görüldüğü gibi yeni PE'ler gizli katmana eklenmektedirler. Buraya eklenen PE'lerin lineer olmayan transfer fonksiyonuna sahip olması istenir, bu da lineer ilişkilerin zaten lineer transfer fonksiyonuna sahip PE tarafından yeterince öğrenildiği kalan hatanın lineer olmayan ilişkilerden kaynaklandığı dolayısıyla lineer olmayan transfer fonksiyonuna sahip yeni PE eklemenin hatayı azaltacağı varsayımına dayanır (Ma ve Khorasani, 2003).



Şekil 3.5 : (a) Başlangıçtaki lineer ağ (b) İlk gizli katmanın oluşumu (Ma ve Khorasani, 2003).



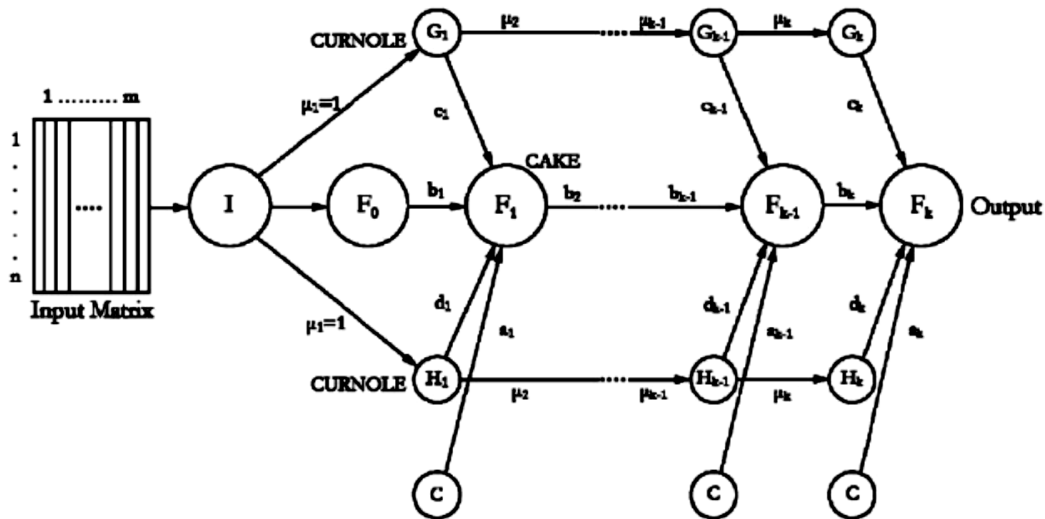
Şekil 3.6 : (a) İlk gizli katmana PE'lerin eklenmesi (b) İkinci gizli katmanın oluşturulması ve PE'lerin eklenmesi (Ma ve Khorasani, 2003).

Ma ve Khorasani (2003) geliştirdiği algoritma hata seviyesi düşmeye devam ettiği sürece aynı gizli katmana PE eklemeye devam eder. Hata seviyesi istikrarlı bir şekilde korunmaya başladığı zaman Şekil 3.6 : (b)'deki gibi eski EE'yi PE yapacak şekilde yeni bir EE eklerler ve yeni oluşan gizli katmana yine lineer olmayan PE'ler eklemeye başlarlar. Eğer bu aşamada hata seviyesi kararlı bir şekilde korunursa son eklenen PE ve EE'ler çıkartılarak ağ çalışır haldeki tek gizli katmanlı haline döndürülür. Eğer hata seviyesinde düşme sağlanırsa tek tek PE'ler eklenir. Bu algoritma hata seviyesi kararlı olarak belli bir seviyede kalıncaya kadar devam edilir. Bu algorithmada sürekli yeni eklenen elemanlar ve ilgili bağlantılar eğitilir, diğer bağlantılar ve elemanlar ise dondurulur. Bu algoritmanın amacı temel bir elemanın öğrenmenin çoğunu yapması ve kalan PE'lerin bu hatayı azaltacak şekilde davranmalarıdır, böylece her katmanda ağın mevcut hatasının artmayacağından emin olunmaktadır (Ma ve Khorasani, 2003).

3.1.7 DAN2

Bitiş elemanını değiştiren diğer bir yöntemde Ghiassi ve Saidane'in (2005) geliştirdiği ve Ghiassi ve diğ. (2005), Ghiassi ve diğ. (2006), Gomes ve diğ. (2006), Ghiassi ve diğ. (2008), Ghiassi ve Nangoy (2009), Ghiassi ve Burnley (2010), Wang

ve diğ., (2010), Velásquez ve Franco, (2010) ve Guresen ve diğ., (2011) tarafından kullanılan DAN2 (Dynamic Architecture for Neural Networks) yapısıdır. Temel işleyiş mantığı Ma ve Khorasani (2003) gibi ağdaki bir lineer düğümün (elamanın) öğrenmenin büyük kısmını alması ve diğer lineer olmayan kısımların ise hata miktarını azaltma amaçlı kullanılmasıdır. Ma ve Khorasani (2003) ayrılan en önemli özelliği saf ileri beslemeli olup, geri bildirim kullanmamasıdır. Bu durum aslında birçok sorunu da beraberinde getirmektedir (Guresen ve diğ., 2011). Ayrıca Velásquez ve Franco (2012) öğrenme algoritmasının net olmadığını belirtmişlerdir. DAN2 Guresen ve Kayakutlu (2011a)'ya göre ise yapay sinir ağı olarak adlandırılmaz. DAN2 yapısının yapay sinir ağı olup olmadığı bu çalışmanın araştırma kapsamı dışındadır, her durumda yapıcı bir algoritma olduğundan ve mevcut bir YSA'ya EE eklediği için bu çalışma kapsamında incelenmiştir. DAN2 yapısı Şekil 3.7'de sunulmuştur.



Şekil 3.7 : DAN2 yapısı (Ghiassi ve diğ., 2005).

DAN2 algoritması saf ileri beslemeli bir algoritmadır; bu yüzden geriye dönük hata düzeltme yapmaz. Dolayısıyla her gözlemi tek tek eğitimde kullanmak yerine girdi matrisini bir kerede gözlemde kullanarak hesaplamalarını yapar. İlk adım olan F_0 düğümünde çok değişkenli lineer regresyon kullanarak veri kümesindeki lineer ilişkili yakalamaya çalışır (Ghiassi ve Saidane, 2005). Eğer hata istenilen seviyeye düşmediyse bu kümede lineer olmayan ilişkilerin de olduğu düşünülerek G_1 ve H_1 düğümleri ile birlikte F_1 düğümü eklenir. G ve H düğümlerinde tüm girdilerin bir referans vektörü ile arasındaki açığı $\sin(\cdot)$ ve $\cos(\cdot)$ fonksiyonlarında kullanır. F_1

düğümünde ise bir önceki F_0 ile G_1 ve H_1 düğümlerinin çok değişkenli lineer regresyon ile çıktı kümesine ilişkilendirir. Şekil 3.7’de görülen C düğümleri ile lineer regresyondaki sabittir (β_0). Algoritma hata istenilen seviyeye ininceye kadar her seferinde bir gizli katman eklemek suretiyle devam eder. Böylece her adımda bir önceki EE (DAN2’nin F düğümleri) gizli katmandaki PE’ye dönüşür.

Ghiassi ve Burnley (2010) DAN2 algoritmasının tek çıktı üretebilmesi problemini aşmak için sınıflandırma problemlerinde kullanılmak üzere bir hiyerarşik prosedür geliştirmişlerdir. Bu prosedürde her aşamada gözlemin seçili sınıfa ya da kalan sınıflardan birine ait olduğuna karar veren bir adet DAN2 algoritması kullanmışlardır. Dolayısıyla eğitim tamamlandığında n adet sınıfı belirlemek için birbiri ardınca çalışan $n-1$ adet DAN2 ağı kullanmışlardır. Burada kullanılan eğitim algoritması DAN2 ile aynı olmakla beraber, hangi adımda hangi sınıfın ayrılacağını belirlemek için ise yeni bir yaklaşım öne sürmüşlerdir. Bu yaklaşımda öncelikle tüm veri seti kullanılarak n adet sınıflama yapmak için her biri bir sınıfı diğerlerinden ayıracak şekilde n adet DAN2 ağı oluşturulur ve aralarından en düşük hataya sahip olan ilk ayrımı yapılacak sınıf (ζ_i) olarak belirlenir ve o sınıfı ayıran DAN2 yapısı saklanır. Bir sonraki adım için mevcut veri kümesinden ζ_i sınıfına ait tüm gözlemler çıkarılır ve kalan $n-1$ sınıf için her biri ayrı bir sınıf diğer sınıflardan ayırt edecek şekilde $n-1$ adet DAN2 ağı oluşturulur ve yine en düşük hata ile sınıflama yapan DAN2 ağı saklanıp bir sonraki adım için veri kümesinden ilgili sınıfın verileri çıkartılır. Bu adımlar en son birbirinden ayrılması gereken iki sınıf ve sadece bu iki sınıfa ait gözlemler kalıncaya kadar devam eder. Son olarak da bu iki sınıfa ait gözlemleri birbirinden ayıran DAN2 ağı oluşturulup daha önce anlatılan hiyerarşik prosedürde bu yeni DAN2 ağı kullanılır. Böylece eğitim aşamasında ise deneme yanılma yoluyla n adet sınıfı birbirinden ayırt etmek için

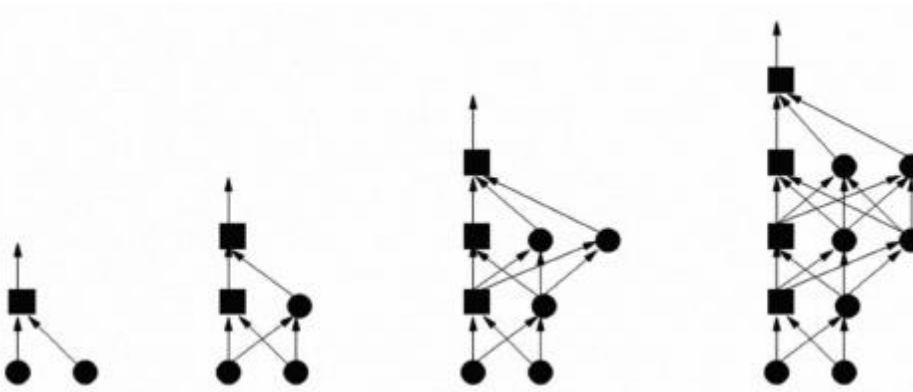
$$n + (n-1) + (n-2) + (n-3) + \dots + 3 + 2 = \left(\frac{n}{2}\right)(n+1) - 1 \quad (3.1)$$

adet DAN2 ağının oluşturulması gerekir.

3.1.8 Kiremit algoritması

Kiremit algoritması (Tiling Algorithm) ilk defa Mezard ve Nadal (1989) tarafından önerilmiştir. Donnelly ve diğ. (1999) bağlı elemanları farklı stratejiler kullanarak eğitmişler ve daha iyi öğrenme performansı sağlamışlardır.

Kiremit algoritmasında ihtiyaç olan yere kiremitlerin konduğu gibi ihtiyaç oldukça yeni eleman eklendiğinden bu adı almıştır (Mezard ve Nadal, 1989). Bu algorithmada iki tip eleman vardır; usta (master) ve bağlı (ancillary). Usta elemanlar bir katmana ilk eklenen elemanlardır. Eğer usta elemanlar yeterli sınıflama performansı sağlamazsa bağlı elemanlar usta elemanın bulunduğu katmana eklenir ve yeni bir usta yeni bir gizli katman oluşturacak şekilde eklenir (Şekil 3.8). Bağlı elemanlar böl ve yönet (divide and conquer) mantığıyla her biri mevcut ağda temsil edilemeyen bir izi temsil edecek şekilde eklenir. Bunu istenilen performans elde edilinceye kadar tekrar ederek yapay sinir ağının yapısı otomatik olarak oluşturulur.



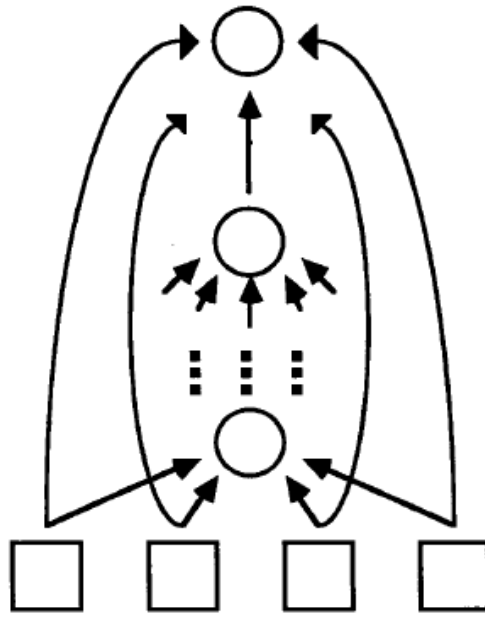
Şekil 3.8 : Kiremit algoritmasını işleyişi (Li ve diğ., 2010).

Parekh ve diğ. (2000) MTiling-Real adı altında reel değerli, çok çıktılı bir kiremit algoritması sunmuştur.

3.1.9 Kule algoritması

Kule algoritması (tower algorithm) ilk olarak Gallant (1990) tarafından ortaya atılmıştır. Bu algoritma basamak-korelasyon algoritmaları ile karıştırılabilir çünkü kule algoritması basamak-korelasyon algoritmalarındaki gibi her yeni eklediği düğüme (elemana) girdi düğümlerinden (başlangıç elemanlarından) birer bağlantı ekler. Ama her adım da bir önceki EE'yi PE'ye çevirip yeni bir EE ekleyerek yapay sinir ağının yapısını otomatik olarak belirlemektedir.

Kule algoritması adını Şekil 3.9’da görüldüğü gibi kule yapısında yapay sinir ağı yapısı oluşturmasından almaktadır. Şekil 3.9’da görülen kareler girdilerin alındığı başlangıç elemanlarını, (SE) yuvarlaklar ise işlem elemanlarını (PE) ve bitiş elemanlarını (EE) göstermektedir. Algoritma önce SE’ler ve tek bir EE ile başlar. İstenilen hata seviyesi yakalanamazsa mevcut EE, PE’ye dönüşecek şekilde yeni bir EE eklenmektedir. Bu yeni eklenen EE sadece kendisinden bir önce eklenmiş PE’den (bir önceki EE’den) bir bağlantı ve girdilerden tüm bağlantıları alır. Bu şekilde algoritmanın istenilen hata seviyesine kadar çalıştırılması ile Şekil 3.9’da görülen kule yapısı oluşur.

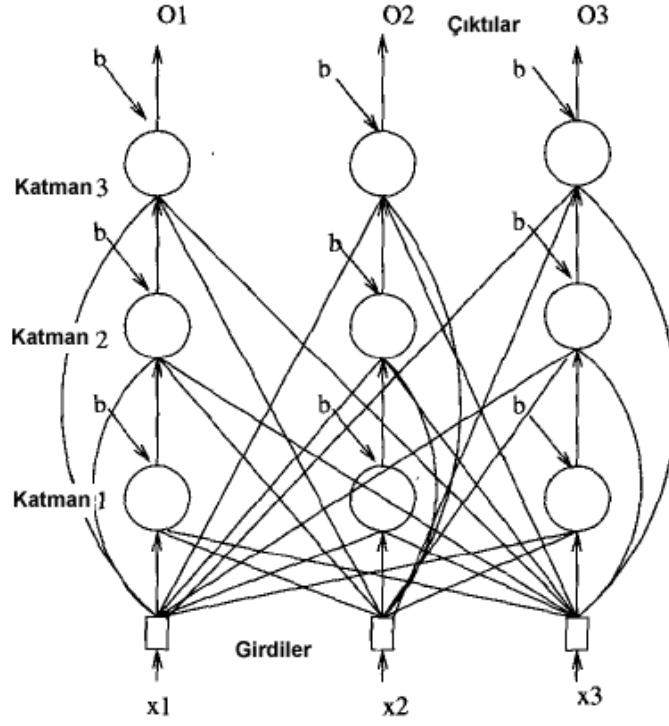


Şekil 3.9 : Kule algoritmasının gösterimi (Gallant, 1990).

Kule algoritmasının sadece iki sınıfın olduğu problemlerde sınıflama için kullanılabilmektedir. Bu kısıtı aşmak için Nagabhushana ve diğ. (2003) kule algoritmasını genişletmiştir (Şekil 3.10). Burada Nagabhushana ve diğ. (2003) bu kategorideki diğer algoritmaların da temel sorunu olduğu gibi aslında n adet çıktı için birbirinden bağımsız n adet ağ kullanarak genişletmiştir.

Şekil 3.10’da rahatlıkla görebileceği üzere sadece ortak başlangıç elemanları (SE’ler) kullanıp birbirinden bağımsız PE’ler ve EE’ler kullanması neden olmuştur. SE’lerde sadece verilerin veri kümesinden okunduğunu, ağda bir akış başlattıklarını ve bu verileri kendi yapılarında değiştirmediklerini göz önüne alırsak Şekil 3.10’da görülen ağın aslında üç adet birbirinden ayrık ve bağımsız çalışan ağlar olduklarını

kolayca görülmektedir. Ayrıca Parekh ve diğ. (2000) MPyramid-Real adı altında reel değerli çok çıktılı bir kule algoritması sunmuştur.



Şekil 3.10 : Genişletilmiş kule algoritmasının gösterimi (Nagabhushana ve diğ., 2003).

Son anlatılan algoritmalarda bitiş elemanı (EE) PE'ye dönüştürülerek yeni bir EE eklemekte ve bu yeni EE'nin daha az hata veren sonuç üretmesi istemektedir. Bu kategoride Ma ve Khorasani (2003) yapıcı stratejilerini, Ghiassi ve Saidane (2005) DAN2'sini, Mezard ve Nadal (1989) kiremit algoritmasını ve Gallant (1990) kule algoritması değerlendirilebilir. Li ve diğ. (2010) bu kategoride incelenen kiremit ve kule algoritmalarının bitiş elemanını değiştirmek suretiyle veri kümesinde yanlış sınıflandırılmış veya hatası yüksek gözlemlerin eklenen PE'ler ile yeni bitiş elemanında düzeltilmeye çalışıldığını belirtmektedir. Diğer iki algoritma da incelendiğinde bu kategoride yer alan algoritmalarının temel motivasyonunun bir elemanın (düğümün) öğrenmenin çoğunu gerçekleştirmesi ve kalan PE'lerin bu elemanın hatasını azaltacak şekilde davranması olduğu görülmektedir. Bu sayede her yeni katmanda ağın mevcut hatasının artmayacağından emin olunabilmektedir. Bu kategoride yer alan algoritmaların diğer bir özelliği de genelde sadece bir EE'ye sahip olabilmeleridir. Literatürde bunların birden fazla EE'ye sahip olabileceği

belirtilmiş olsa da genellikle oluşturdıkları diğer EE'ler, PE'leri diğer EE'lerin PE'lerinden bağımsız ve aslında birbirinden ayrık birden fazla yapay sinir ağı oluşturmaktadır.

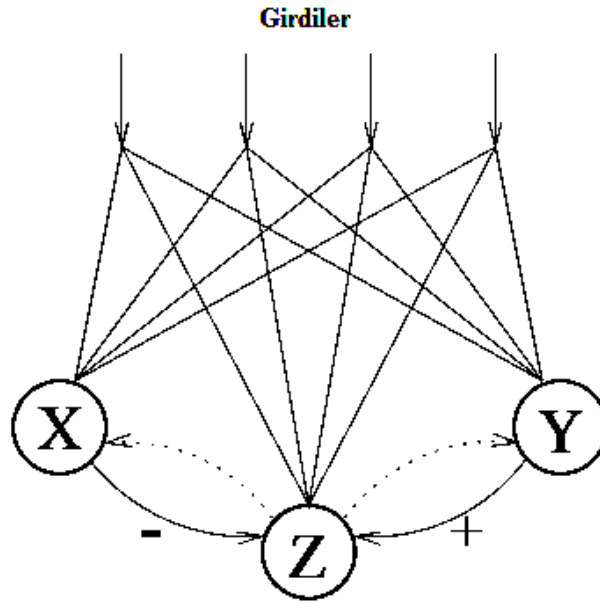
Bu tip algoritmalar ilk bakışta türetilmiş yapay sinir ağına yeni EE ekliyor gibi gözükmemektedir. Ancak burada dikkat edilmesi gereken nokta bu kategorideki algoritmaların hiç birinin yeni bir işleve sahip EE eklemeyeceği, kısaca yapay sinir ağına yeni bir işlev kazandırmadığıdır. Temel aldığımız YSA tanımına göre EE çıktı matrisine bir çıktı gözlemi için bir özellik üretmelidir. Her YSA'da EE sayısı kadar farklı çıktı özelliği vardır. Ama bu kategorideki algoritmalar çıktı matrisine yeni bir özellik eklemiyorlar, sadece var olan özelliği üreten düğümü değiştiriyorlar. Bu sebeple Ma ve Khorasani'nin yapıcı algoritması, DAN2, kiremit algoritması ve kule algoritmasını *bitiş elemanı değiştiren algoritmalar* olarak adlandırabilir.

3.1.10 Upstart algoritması

Upstart algoritması ilk olarak Freat (1990) tarafından oluşturulmuştur. Algoritmanın lineer olmayan verilerde de sağlıklı işleyip ağıdaki ağırlıkları sürekli hatayı azaltacak şekilde güncelleyebilmek için Gallant (1986) tarafından önerilen cep algoritmasını (pocket algoritma) kullanmıştır.

Upstart algoritması da sadece bir adet EE kullanmaktadır. EE değiştiren algoritmalarından temel farkı hatayı düzeltmek için EE'yi iki yeni PE ekleyerek düzeltmeye çalışmasıdır.

Upstart algoritması ikili (binary) sınıflama yapar ve çıktıyı Şekil 3.11'de görüldüğü gibi Z düğümü ile sağlar. Bu düğümde oluşan hatayı (yanlış sınıflamayı) düzeltmek için Z düğümünün (EE'in) önüne Şekil 3.11'de görülen X ve Y adında iki adet PE ekler. X düğümü ile yanlışlıkla "1" verilen sonuçlar düzeltilmeye çalışılırken Y düğümü ile de yanlışlıkla "0" verilen düğümler düzeltilmeye çalışılır. Z düğümü iki farklı duruma sahiptir; açık (on) ve kapalı (off). Açık olan Z düğümü "1" değeri üretirken kapalı olan EE "0" değeri üretir.



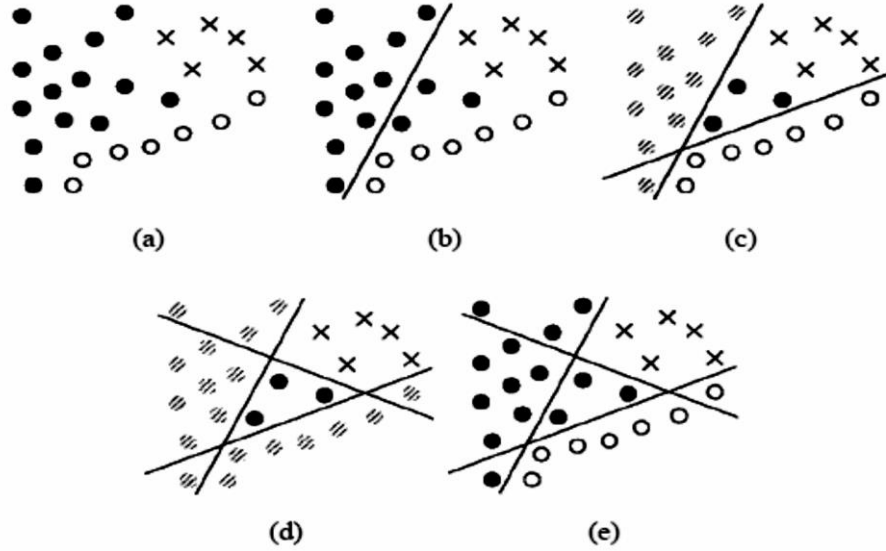
Şekil 3.11 : Upstart algoritmasında yeni PE'ler eklenmesi (Freaan, 1990).

Fanguy ve Kubat (2002) upstart algoritmasının sadece iki sınıfı birbirinden ayırabilmesi sınırlamasını aşmak için bir prosedür geliştirmişler ve bu prosedür Fanguy ve Bhutta (2003) tarafından tedarikçi seçiminde kullanılmıştır. Fanguy ve Kubat (2002) tarafından geliştirilen prosedür temelde n farklı sınıf için n adet EE (Z düğümü) kullanmaya dayanır. Her EE normal upstart algoritmasında olduğu gibi iki farklı duruma sahiptir; açık (on) ve kapalı (off). Açık olan EE “1” değeri üretirken kapalı olan EE “0” değeri üretir. EE sayısı n adet olduğunda ise doğru sınıfı temsil EE'nin açık olması diğer EE'lerin ise kapalı olması istenir. Eğer aksi bir durum söz konusuysa hata göz önüne alınarak mevcut ağ yapısı geliştirilir.

3.1.11 CARVE

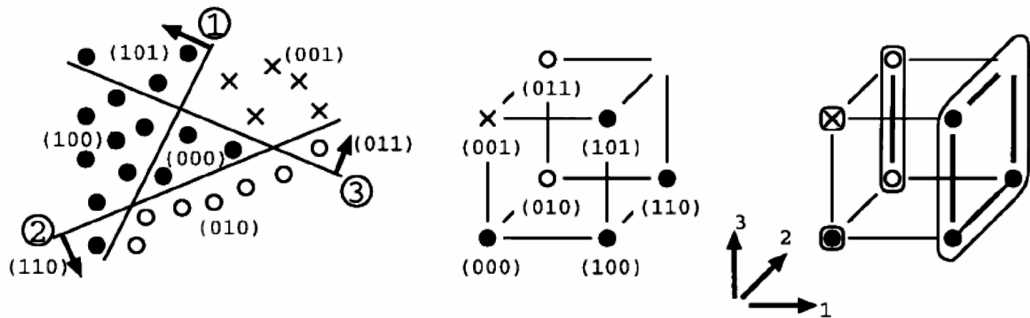
CARVE algoritması ilk defa Young ve Downs (1998) tarafından oluşturulmuştur. Bu algoritma ilk önce sadece başlangıç elemanları (SE) ve eğitim setindeki her bir sınıf için bir bitiş elemanı (EE) olacak şekilde hazırlanan bir YSA ile başlar. Bu durumda gizli katmanlarda hiç PE yoktur. Daha sonra aynı sınıfa ait bir veri kümesini diğer eğitim verilerinden ayırabilecek bir hiper-düzlem bulunur ve buna CARVE kümesi denir. Bunu sağlayacak bir tane PE gizli katmana eklenir. CARVE kümesi veri setinden çıkarılır. Bu adımlar veri setinde sadece bir sınıf kalıncaya kadar devam eder. Böylelikle gizli katmana PE ekleme işlemi biter ve gizli katman ile çıktı katmanı arasındaki ağırlıklar tespit edilir.

Bunu Şekil 3.12'deki (a) üç sınıflı örnek ile açıklarsak algoritma önce içi dolu noktaların çoğu ile diğerlerini ayıran (Şekil 3.12, b) bir doğru bulunur. Kalan örnekler incelenir. İçi boş daireler diğer gözlemleri (Şekil 3.12, c) birbirinden ayıran bir doğru çizilir. Kalan örnekler sadece içi dolu daireler ile çarpı işaretleridir ve bunlar da birbirinden bir doğru ile ayrılır (Şekil 3.12, d). Böylece 3 doğru ile Şekil 3.12 (e)'de görülen karar alanları ile sınıfların ayrılması sağlanmış olunur.



Şekil 3.12 : CARVE algoritması ile sınıfların ayrılması (Young ve Downs, 1998).

Şekil 3.12 (e)'de görülen karar alanları oluşturulması sırasında her doğruyu gizli katmana bir PE ekleyerek ifade ed3'tilir. Bu da aslında Şekil 3.13'te görülen birim küpte gösterilen PE yapısıyla ifade edilebilir. Örneğin Şekil 3.13'te doğruların okla gösterilen kısmı "1" diğer kısmı "0" değeri aldığını düşünürsek (000) koordinatında içi dolu dairelerin sınıfını ifade eder.



Şekil 3.13 : Gizli katmandaki PE'ler ile CARVE algoritmasının gösterimi (Young ve Downs, 1998).

CARVE algoritması sonucu oluşan YSA, en baştan sona kadar aynı işleve ve sayıya sahip EE barındırır, dolayısıyla geliştirme yapmaz.

3.1.12 İleri beslemeli sinir ağlarının çapraz geçerlilikle inşa edilmesi

Setiono'nun (2001) önerdiği N2C2S (Neural Network Construction with Cross-Validation Samples) algoritmasında 6 adım izlenmiştir. Bu algoritmanın adımları Şekil 3.14'te görülmektedir.

Girdiler;

T: Eğitim örneklerinin kümesi (x_p, y_p) , $p=1, 2, \dots, P1$

C: Çapraz geçerlilik kümesi (x_p, y_p) , $p=1, 2, \dots, P2$

Amaç; Genelleme özelliği yeterince iyi olan bir ileri beslemeli yapay sinir ağı oluşturmak

Adım 1. N1 ağı N adet girdi birimli (SE), M adet çıktı birimli (EE) ve H adet PE'li bir yapay sinir ağı olsun.

Adım 2. N1 ağının ağırlıklarını rassal olarak ata ve hatayı minimize edecek şekilde eğit. AT1 ve AC1, N1 ağının sırasıyla T ve C kümelerindeki tutarlılığı olsun.

Adım 3. N2 ağı N adet girdi birimli (SE), M adet çıktı birimli (EE) ve H+h adet PE'li bir yapay sinir ağı olsun.

Adım 4. N2 ağının ilk H adet PE'sinin ağırlıklarını N1 ağının eğitilmesi sonucu elde edilen ağırlıklar olarak belirle. Kalan h adet PE'sinin ağırlıklarını rassal olarak ata. AT2 ve AC2, N2 ağının sırasıyla T ve C kümelerindeki tutarlılığı olsun.

Adım 5.a. Eğer $(AT2 + AC2) > (AT1 + AC1)$ ise

1. $H \leftarrow H + h$

2. $N1 \leftarrow N2$; $AT1 \leftarrow AT2$; $AC1 \leftarrow AC2$

3. Eğer $H < \max H$ ise 3. adıma dön

Adım 5.b. Değil ise

1. Yeni bir N3 ağının N adet girdi birimli (SE), M adet çıktı birimli (EE) ve H+h adet PE'li bir yapay sinir ağı olacak şekilde kur. Ağırlıkların tamamını rassal olarak ata ve eğit. AT3 ve AC3, N3 ağının sırasıyla T ve C kümelerindeki tutarlılığı olsun.

2. Eğer $(AT2 + AC2) > (AT1 + AC1)$ ise

(a). $H \leftarrow H + h$

(b). $N1 \leftarrow N2$; $AT1 \leftarrow AT2$; $AC1 \leftarrow AC2$

(c). Eğer $H < \max H$ ise 3. adıma dön

Adım 6. N1 yapay sinir ağını çıktı olarak ver.

Şekil 3.14 : N2C2S algoritması sözde programı (Setiono, 2001).

Bu algoritma her adımda SE ve EE sayısını sabit tutarak aradaki PE sayısını ve ağırlıklarını tespit etmeye çalışmaktadır ve yeni bir EE eklememektedir.

3.1.13 STRIP ağ büyüme algoritması

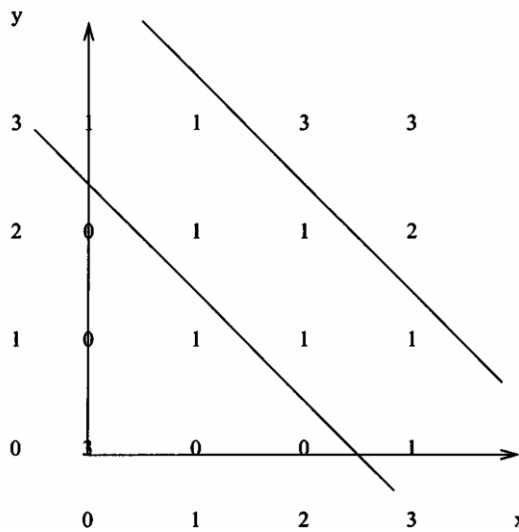
STRIP tabanlı yapay sinir ağı algoritması Ngom ve diğ. (2001) tarafından ortaya atılmıştır. Bu algoritma *strip* adı verilen iki hiper düzlem arasında kalan aynı sınıfa ait nokta kümelerinden en büyüğünü bulmaya dayanır.

Ngom ve diğ. (2001) bu en büyük strip'i bulmak için genetik algoritmalarla arama yapmışlardır. Yaptıkları her arama sonucu buldukları en büyük strip gizli katmandaki bir PE ile temsil ettirmişlerdir.

Algoritma önce SE ve EE'leri olan ama gizli katmanda hiç PE olmayan bir ağ yapısı oluşturur. Sonra eğitim kümesinde (E) hiç gözlem kalmayınca kadar şu işlemleri yapar;

1. Genetik algoritmaları kullanarak G alt kümesini veri setinde mevcut olan en büyük strip kümesi A 'ya en yakın olacak şekilde ara.
2. Yeni bir PE, G kümesini temsil edecek şekilde gizli katmana yerleştir.
3. E kümesinden G kümesini çıkararak yeni E kümesini oluştur.

CARVE'den farkı ve üstünlüğü CARVE'nin doğası gereği veri kümesinin sınırlarına değen hiperdüzlemlerle sınıfları birbirinden ayırmasıdır (Şekil 3.13). Böylece tek bir hiperdüzlemle bir sınıfa ait veri kümesiyle kalan eğitim verisinin hepsini birbirinden ayırabilmektedir. STRIP algoritması ise (genetik algoritma ile tarama yaptığından) en büyük strip'i bulmaya çalışır ve bunu küme veri setini sınırlarına dayanmış olmak zorunda değildir (Şekil 3.15).



Şekil 3.15 : En büyük strip örneği (Ngom ve diğ., 2001).

STRIP algoritması sonucu oluşan YSA, en baştan sona kadar aynı işleve ve sayıya sahip EE barındırır.

3.1.14 Petrol bulma algoritması

Frattale-Mascioli ve Martinelli (1995) tarafından ortaya atılan petrol bulma algoritması (oil-spot algoritma) N -boyutlu uzayı 3 boyutlu birim küpe eşlemeye dayanır. Bu algoritma CARVE algoritmasının çıkış noktası niteliğindedir. İşlem sırasında CARVE’de olduğu gibi hiperdüzlemlerle uzayı kesimler yapıp aynı sınıfa ait bir alt küme arar. Her bir alt küme ilk gizli katmandaki bir PE ile temsil edilir. Daha sonra bir katman daha eklenerek aynı sınıfı belirleyen PE’lerin bu katmanda “ya da” mantıksal operatörü ile bağlanmış gibi çalışması sağlanır.

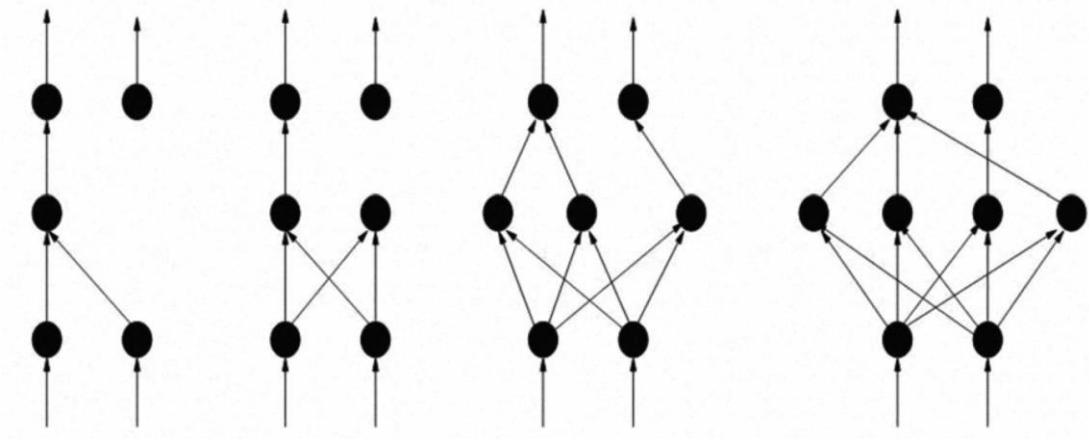
Bu algoritma mevcut bir YSA’ya yeni bir işlev kazandıran yeni bir EE eklemez.

3.1.15 Sınırlandırılmış Coulomb enerji ağı

Sınırlandırılmış Coulomb enerji ağı Reilly ve diğ. (1982) tarafından ortaya atılmıştır. Bu metot aslında bugün radyal tabanlı fonksiyon ağları dediğimiz yapıdır (Li ve diğ., 2010). *Radyal Tabanlı Fonksiyon Ağları (RTFA)* ise gizli katmanında H adet taban fonksiyonu bulunduran çok katmanlı algılayıcılardır (Broomhead ve Lowe, 1988; Moody ve Darken, 1989). RTF ağları Gaussian olarak adlandırdığımız m_h ve s_h ’nin sırasıyla merkezi ve sapmayı belirttiğimiz h nolu PE için şu fonksiyonu kullanırlar (Alpaydın, 2004);

$$p_h = \exp \left[-\frac{\|x^t - m_h\|^2}{2s_h^2} \right] \quad (3.2)$$

Sınırlandırılmış coulomb enerji ağı tek gizli katmanı olan, PE’leri Denklem 3.4’teki fonksiyonu kullanan, her bir PE’nin s parametresinin belli bir değerden başlayıp sadece aynı kümeye ait verileri kapsayınca kadar azaltıldığı bir ağıdır (Li ve diğ., 2010). İz bulma yöntemiyle yeni PE’lerin merkez parametresi (m) tespit edilir ve onlar içinde (s) parametresi hesaplanır. Tüm eğitim verisi PE’ler tarafından sınıflanabildiğinde PE eklenmesi durur ve aynı sınıfa ait verileri temsil eden PE’ler “ya da” mantıksal kapılarıyla birleştirilerek ağ yapısı Şekil 3.16’da görüldüğü gibi oluşturulur.

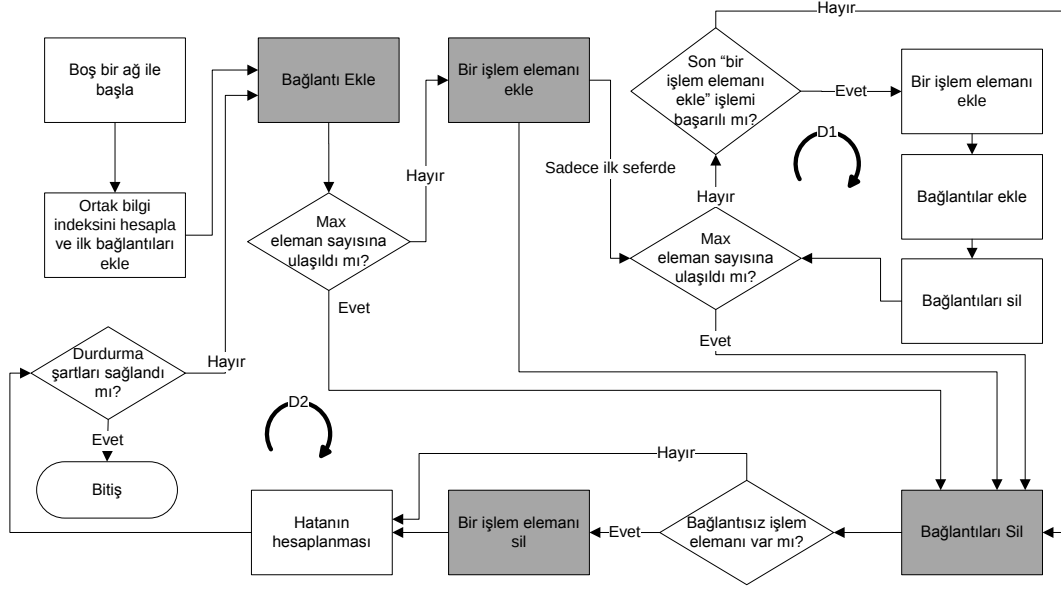


Şekil 3.16 : Sınırlandırılmış Coulomb Enerji Ağı'nın oluşturulması (Li ve diğ., 2010).

Şekil 3.16'da görülen örnekte önce 2 adet başlangıç elemanı (SE), 2 adet bitiş elemanı (EE) ve bir adet PE ile başlanır. Daha sonra tüm SE'lerden veri alan yeni bir PE eklenir ve parametreleri hesaplanır. Her PE sadece temsil ettiği sınıfa ait EE'ye bağlanır. Veri kümesinde PE'lerle temsil edilmeyen veri kalmadığında PE ekleme durur ve Şekil 3.16'da en sonda görülene benzer bir yapı elde edilir. Buradaki örnekte 4 PE kullanılmış 3 tanesi bir sınıfı belirlemede kullanıldığından sadece ilgili EE'ye bağlanmışlardır, diğer PE ise kalan sınıfı belirlemede kullanıldığından sadece diğer EE'ye bağlanmıştır, böylece “ya da” mantıksal kapıları oluşturulmuş olur. Daha önce bahsedilen kaynak dağıtım ağlarından farkı girdi katmanı ile çıktı katmanı arasında direkt bir bağlantı olmamasıdır.

3.1.16 Rasgele bağlantılandırılmış ileri beslemeli sinir ağlarını birleştirme

Puma-Villanueva ve diğ. (2012)'nin geliştirdiği bu algoritma sadece girdi ve çıktı katmanlarından oluşan ve gizli katmanı boş bir ağ yapısıyla başlar. Bu ağda girdi ve çıktı katmanları arasında bağlantı oluşturulmaz. Sonra ilk olarak ortak bilgi (mutual information) kullanılarak her başlangıç elemanı (dolayısıyla her bir özellik) için bir önem hesaplanır ve buna bağlı olarak bu başlangıç elemanları çıktı katmanındaki bitiş elemanlarına bağlanmaya başlanır. Daha sonra tanımlanan kurallar çerçevesinde yeni bağlantılar, yeni işlem elemanları (hidden neurons), eklenmeye veya tanımlanan şartlar sağlandıkça bu bağlantılar veya işlem elemanları silinir. Bu işlemler algoritmanın durdurma kriterlerine ulaşıncaya kadar devam eder. Bu kriter art arda 4 işleminde yapılamaması (bağlantı veya işlem elemanının eklememesi veya bağlantı veya işlem elemanının silinmesi) durumunda gerçekleşir. Algoritmanın genel akışı Şekil 3.17'de verilmiştir.



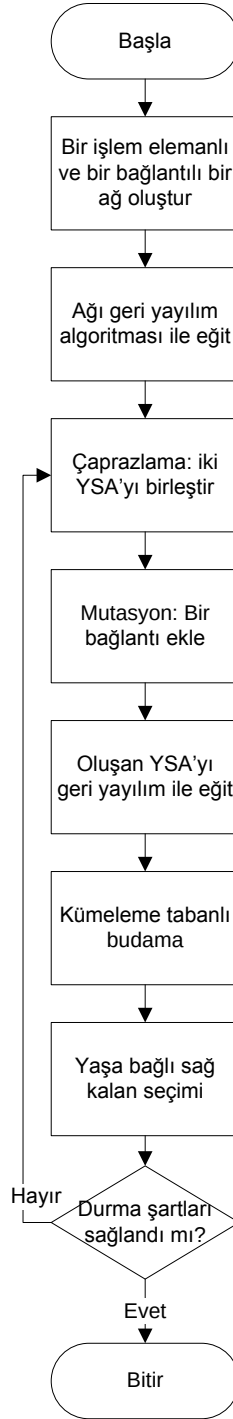
Şekil 3.17 : Rasgele bağlantılandırılmış ileri beslemeli sinir ağlarını birleştirme algoritması (Puma-Villanueva ve diğ.,2012).

Mevcut algoritma incelendiğinde, hiç bir adımda yeni bir bitiş elemanı eklenmediği sadece girdi ve çıktıları sabitleyip aradaki gizli katmanları ve aradaki bağlantıları oluşturduğu görülmektedir.

3.1.17 Evrimsel yapıcı ve budayıcı algoritma (ECPA)

Yang ve Chen (2012)'in geliştirdiği bu algoritma girdi katmanı, çıktı katmanı ve içerisinde bir adet işlem elemanının bulunduğu gizli katmana sahip ağlardan oluşan bir popülasyon ile başlamaktadır. Bu ağlar geri yayılım (back propagation) algoritması ile eğitilmektedir. Bu popülasyondan alınan iki YSA çapraz geçiş (crossover) ile birleştirilmektedir. Ardından mutasyon adımı bu ağa yeni bir bağlantı eklenerek YSA geliştirilmeye çalışmaktadır. Oluşan bu ağ geri yayılım algoritması ile eğitilmekte ve sonrasında gizli katmandaki işlem elemanları 2 kümeye; iyi ve kötü, ayrılmaktadır. Kötü kümesinde yer alan işlem elemanları, stokastik bir yöntemle aşırı işlem elemanı silmeyi önleyecek şekilde silinmektedir. Daha sonra en iyiler bir sonraki nesle aktarılmakta, bu yapılırken normalden farklı olarak bir bireyin kaç nesildir yaşadığı da göz önüne alınmaktadır. Bu işlem iki aşamadan oluşmaktadır; ilki klasik turnuva yöntemi ile en iyilerin seçilmesi, ikincisi ise yaşa bağlı olarak oluşturulan index ile uzun süre yaşamış olanların seçilmesi.

Çapraz geçiş, mutasyon, küme bazlı budama adımları ve yaşa bağlı yeni popülasyon seçimi istenilen nesil sayısına ulaşıncaya kadar tekrar edilir. Son nesilin en iyisi sonuç olarak verilir. Bu algoritmanın akış şeması Şekil 3.18’de görülmektedir.



Şekil 3.18 : ECPA Algoritması(Yang ve Chen, 2012).

ECPA algoritması dinamik yapay sinir ağları açısından incelendiğinde, hiç bir adımda yeni bir bitiş elemanı ekmediği sadece girdi ve çıktıları sabitleyip aradaki

gizli katmanları oluřturması nedeniyle dinamik yapay sinir ağı algoritması olmadığı görölmektedir.

3.1.18 İleri beslemeli sinir ağılarında bir yapı optimizasyon algoritması

Han ve Qiao (2013) tarafından geliştirilen bu algoritma (A structure optimization algorithm for feed forward neural network construction) hem yapıcı hem budayıcı özelliğı olan bir algoritmadır. Bu algoritma sadece bir gizli katmanı olan ileri beslemeli yapay sinir ağılarında çalışmaktadır. Bu çalışma dinamik yapay sinir ağıları açısından incelendiğinde bu algoritmanın girdi ve çıktı katmanlarını sabitleyip aradaki gizli katmanı ve gizli katmandaki işlem elemanlarını düzenlediğı görölmektedir. Karşılaştırma amacıyla kullandığı veri kümelerinden birisi de bu çalışmada kullanılan aslanagzı (iris) veri kümesidir. Bu kümede elde ettiğı en iyi sonuç ortalama % 95.35’dir.

3.1.19 Sinir ağıları için hızlı yapıcı-kaplayıcı algoritma

Wang (2008) tarafından geliştirilen Hızlı Yapıcı-Kaplayıcı Algoritma (Fast Constructive-Covering Algorithm for neural networks) girdi ve çıktı katmanlarını sabitleyip araya boş bir gizli katman koyarak başlamaktadır. Daha sonra gerektiğçe ara katmana işlem elemanı koyarak yapay sinir ağını oluşturmaya çalışmaktadır. Yeni işlem elemanı eklerken girdi hiper-uzayında genişleme sağlamaya çalışmaktadır. Karşılaştırma amacıyla kullandığı veri kümelerinden birisi de bu çalışmada kullanılan aslanagzı (iris) veri kümesidir. Bu kümede elde ettiğı en iyi sonuç iki eğitim ikide test verisinde olmak üzere 150 gözlemde dört hatalı sınıflamadır (%97,33 doğru sınıflama). Normal ÇKA ile aslanagzı veri kümesinde en iyi sonuçları elde etmiş ve iki eğitim verisinde bir adette test verisinde olmak üzere toplam üç hatalı sınıflama yapmıştır (%98 doğru sınıflama).

3.1.20 C-Mantec

Hata düzeltme ile rekabetçi çoğunluk ağı eğitim algoritması (C-mantec) Subirats ve diğ. (2012) tarafından geliştirilmiştir. Çoğunluk (majority) fonksiyonlarını kullanan bu algoritma tek gizli katmanlı ve tek çıktılı bir yapay sinir oluşturur. Çoğunluk fonksiyonu N adet girdiyi tek bir bit’e dönüřtüren mantıksal bir fonksiyondur. Bu algoritma eğitim sırasında “termal perceptron” (ısıya dayalı tek katmanlı tek çıktılı sinir ağı) öğrenimi adı verilen, benzetim tavlamasındaki sıcaklık değıřimlerine

benzeyen bir sıcaklık değişimine bağlı öğrenme kullanır. Algoritma, ufak bir yapay sinir ağıyla başlar ve yanlış sınıflandırılmış sınıfları tespit eder. Daha sonra büyüme etkenini hesaplar (g_{fac}) ve her bir işlem elemanı (PE) için hesapladığı ısı etkeni (T_{fac}) ile kıyaslar. Eğer en yüksek ısı etkeni büyüme faktöründen daha büyükse ait olduğu PE'nin ağırlıkları termal perceptron algoritmasına göre düzeltilir. Eğer ısı etkenlerinden en büyüğü, büyüme faktöründen daha küçük ise bu durumda algoritma yeni bir PE ekler. Algoritma yanlış sınıflama kalmadığında ise durur.

Daha yeni geliştirilmesi sebebiyle C-Mantec algoritmasının bir çok eksikliği vardır. Bu eksiklerden en göze çarpanlar ikili (binary) sistemi kullanması, tek çıktı vermesi ve dolayısıyla sadece iki sınıfı birbirinden ayırt etmede kullanılmasıdır. Dinamik YSA açısından incelendiğinde algoritmanın hiçbir şekilde mevcut bir yapay sinir ağına bir bitiş elemanı ekmediği göze çarpmaktadır

3.2 Budama Algoritmaları

Uygun yapay sinir ağı yapısını bulmada kullanılan önemli bir yöntem grubu da budama algoritmalarıdır. Bu algoritmalar yeterince büyük bir yapıyla başlarlar ve yavaş yavaş yapay sinir ağını ufaltarak daha uygun boyuta getirirler.

Mozer ve Smolensky (1989) bunu yapmak için daha az önemli PE'leri tespit edip bunları yok ederek yapay sinir ağı yapısını uygun boyuta getirmektedir. Karnin (1990) ise hata fonksiyonun mevcut PE'lerin silinmesine duyarlılığını hesaplayarak budama işlemini gerçekleştirmektedir. Castellano ve diğ. (1993) 3 farklı budama algoritmasını; Sietsma ve Dow (1988), Burkitt (1991) ve Pelillo ve Fanelli (1993) algoritmalarını karşılaştırmışlardır. Castellano ve diğ. (1997) ise problemten bağımsız tekrarlayan bir budama algoritması geliştirmişlerdir.

Budama algoritması kapsamındaki algoritmaların dinamik yapay sinir ağları oluşturmasını beklenmez. Çünkü tanım gereği bir yapay sinir ağının dinamik yapay sinir ağı olarak adlandırılması için mevcut bir YSA'ya sonradan en az bir adet EE eklenmiş olması gerekmektedir. Budama algoritmaları hiçbir zaman ekleme yapmadıklarından dinamik yapay sinir ağı oluşturmazlar.

3.3 Düzenleme Algoritmaları

Düzenleme algoritmaları (Regularization) görece büyük bir ağı yapısı ile başlar ve bu yapıdaki sonuca etkisi az olan ama hâlâ hatanın düşürülmesine yardımcı olan bağlantıları düzenlemeye çalışır (Ma ve Khorasani, 2003). Bu bağlantılar eğitim verisinde hatayı düşürse de yeni bir veri ile karşılaşıldığında genelleme yeteneklerini kaybettiklerinden hatanın artmasına sebep olurlar. Düzenleme algoritmaları, sözü geçen sorunu aşmak için düzenleme kuralları oluştururlar ve bu kurallara göre daha az önemli olan ağırlıkların sıfıra çekilmesini yani silinmesini sağlarlar. Düzenleme kuralları Thodberg (1996) ve Buntine ve Weigend (1991) de olduğu gibi Bayes kuramına dayalı yöntemlerle oluşturulabilir. Ancak Ma ve Khorasani (2003) bu kurallarda Bayes'e dayalı kanıtların olmadığını ve bazen yeterince iyi sonuçlar vermediğini belirtmişlerdir.

Yukarıda adı geçen çalışmalarda da görüldüğü üzere düzenleme algoritmaları yapay sinir ağının yapısını (mimarisini) belirlememektedirler. Dolayısıyla araştırmacı bu yapıyı kendisi verip düzenleme algoritmalarının gereksiz bağlantıları temizlemesini sağlamalıdır. Bu durumda yeni bir bitiş elemanı ya da bir yapay sinir ağına mevcut işlevlerinin dışında yeni bir işlev hiçbir zaman eklenmediğinden bu kategorideki algoritmalar dinamik yapay sinir ağı oluşturmazlar. Yapılan incelemelerde evrimsel algoritmaların bitiş elemanı eklemek maksadıyla kullanımına rastlanmamıştır.

3.4 Evrimsel Algoritmalar

Evrimsel algoritmalar yazında yapay sinir ağlarının parametre tespitinde kullanılmaktadır (Çınar, 2007). Kullanılan(veya kullanılması umulan) bu evrimsel algoritmaların çeşitleri şöyle sıralanabilir (Kiranyaz ve diğ., 2009); genetik algoritmalar (Goldberg, 1989), genetik programlama (Koza, 1992), evrimsel stratejiler (Back ve Kursawe, 1995), evrimsel programlama (Fayyad ve diğ., 1996), parçacık sürü algoritmaları (partial swarm optimization) (Kennedy ve Eberhart, 1995). Evrimsel algoritmalar popülasyon tabanlı stokastik süreçlerdir ve yerel optimumlara takılı kalmayı önlemek amacıyla kullanılmaktadır (Kiranyaz ve diğ., 2009).

Yapay sinir ağı parametrelerinden öğrenme oranını evrimsel algoritma kullanarak belirleyenlere Kim ve Bae (2005), Wang ve Huang (2007), ve Yi-Hui (2007) örnek

olarak verilebilir. Evrimsel algoritmalar ile başlangıç ağırlıklarını tespit edenlere ise Castillo ve diğ. (2000), Kim ve Bae (2005), Wang ve Huang (2007), ve Kuo (2001), gizli katman sayısını ve bu katmanlardaki işlem elemanı sayısını tespit edenlere ise Castillo ve diğ. (2000), Niska ve diğ. (2004), Ferentinos (2005), Kim ve Bae (2005), Wang ve Huang (2007), ve Yi-Hui (2007), işlem elemanlarının transfer fonksiyonunun türünü tespit edenlere ise Ferentinos (2005) örnek gösterilebilir. Evrimsel algoritmalarla ayrıca hangi girdilerin kullanılacağı araştırılmış yani başlangıç elemanları tespit edilmiştir. Bu tip çalışmalara ise Niska ve diğ. (2004), Sexton ve diğ. (2004) ve Doganis ve diğ. (2006) örnek verilebilir.

3.5 Modüler Yapay Sinir Ağları ve Komite Makineleri

Bir problemin çözümünde bir adet yapay sinir ağı kullanılması karmaşık problemlerin çözümünde bazı zorlukları da beraberinde getirmiştir. Bu sorunu aşmak birden fazla YSA beraber kullanılmış ve bunu gerçekleştirmek için literatürde iki farklı yaklaşım ortaya çıkmıştır (Azam, 2000). Bunlardan birincisi topluluk tabanlı (ensemble-based) (Sharkey, 1996); diğeri ise modüler yapıdır (Haykin, 1994).

Topluluk tabanlı (ensemble-based) yapıya komite makinesi (committee machine) de denilmektedir (Azam, 2000). Komite makineleri eğitilmiş ve hali hazırda çalışır durumdaki yapay sinir ağlarının sonuçlarının en iyi oranlarda birleştirilmesi için çalışır. Dolayısıyla komite makineleri, veri kümesini kullanarak eğitilmiş birçok farklı yapay sinir ağından oluşur. Her bir yapay sinir ağının güçlü olduğu (hatasının az olduğu) durumları birleştirmek için ise uzmanların karışımı (mixture of experts) ve topluluk ortalaması (ensemble averaging) gibi yöntemler kullanılır (Haykin, 1999).

Modüler yapay sinir ağlarında ise karmaşık bir problem daha basit alt problemlere bölünerek, her bir alt problem için uzmanlaşmış bir yapay sinir ağı kullanmaya dayanır (Azam, 2000). Genellikle Azam (2000)de kullandığı gibi böl ve yönet (divide and conquer) tabanlı algoritmalarla çalışırlar. Kotania ve diğ. (2004), Almasri ve Kaluarachchi (2005), Molinavilaplana ve diğ. (2007), Chris Tseng ve Almogahed (2009), Kourakos ve Mantoglou (2009), Zhao (2009) ve Zhao ve diğ. (2010)'nın çalışmaları karmaşık problemlerde başarıyla uygulanmış modüler yapay sinir ağı örnekleridir.

Komite makinelerinin ve modüler yapay sinir ağılarının mevcut bir YSA'ya geliştirme yaptırmadığı; yeni bir işlev kazandıracak şekilde EE ekleme yapmadığı görülmektedir.

4. DİNAMİK VE BÜTÜNLEŞİK YSA GELİŞTİRİLMESİ

Giriş bölümünde yapay zekâ modellerinde kullanılan farklı bilgi edinimi türleri tanımlanmış ve bu çalışmanın kapsamı olarak bitiş elemanı (EE) ekleyen algoritmalar olarak belirlenmişti. İkinci bölümde ise birçok yapay sinir ağı tanımının eksik yönleri irdelenmiş ve kapsamlı güncel bir yapay sinir ağı tanımı verilmişti. Üçüncü bölümde ise verilen bu tanım çerçevesinde yazındaki mevcut yapay sinir ağı algoritmalarının nasıl işlediği ve ne yaptıkları incelenmişti. Yapılan incelemeler sonucunda mevcut algoritmaların yeni işlev kazandıracak şekilde bitiş elemanı eklemediği dolayısıyla geliştirme yapmadıkları tespit edilmiştir. İncelenen bu algoritmalar geliştirme yapmadıkları için yeni sonuçlar üretebilme, yeni konularda bilgi sahibi olma kabiliyetine sahip değillerdir.

Bu bölümde önceki bölümlerde bahsedilen algoritma ve tanımlar ışığında yapay sinir ağlarının yapısal değişiklikleri ayrıntılı olarak tanımlanmıştır. Yapılan net tanımların varlığı yapay sinir ağlarını ve ilintili konuları daha kolay anlaşılmasına ve YSA araştırmalarının hangi yöne doğru gitmesi gerektiğinin belirlenmesine yardımcı olacaktır.

4.1 Gelişen (Improving Structured) YSA Gereksinimi

3. bölümde incelenen algoritmaların özeti Çizelge 4.1’de görülmektedir.

Çizelge 4.1 : Literatürdeki YSA algoritmalarının özeti.

Algoritma	Anlatıldığı Bölüm	Geliştirilme amacı	Kullanıldığı Alan	Bizim Amacımız İçin Yetersizliği
Dinamik Düğüm Yaratma	3.1.1	YSA yapısını oluşturma	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
İzdüşüm Takip Regresyonu	3.1.2	Problem boyutunu azalma	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
Basamak-Korelasyon	3.1.3	Gizli Katmandaki optimal işlem elemanı sayısını bulmak	Regresyon	Bitiş Elemanı (EE) eklemiyor
Kaynak Dağıtım Ağları	3.1.4	Girdi-çıkı eşlemesini sağlamak	Regresyon	Bitiş Elemanı (EE) eklemiyor
Grup Metoduyla Veri İşleme	3.1.5	Girdi-çıkı eşlemesini performansını arttırmak	Regresyon	Bitiş Elemanı (EE) eklemiyor
Ma ve Khorasani’nin Yapıcı Stratejisi	3.1.6	Girdi-çıkı eşlemesini sağlamak ve lineer olmayan ilişkileri yakalayarak performansı arttırmak	Regresyon	Bitiş Elemanı (EE) eklemiyor

Çizelge 4.1 (devam): Literatürdeki YSA algoritmalarının özeti.

Algoritma	Anlatıldığı Bölüm	Geliştirilme amacı	Kullanıldığı Alan	Bizim Amacımız İçin Yetersizliği
DAN2	3.1.7 3.1.10	Girdi-çıkı eşlemesini sağlamak ve lineer olmayan ilişkileri yakalayarak performansı arttırmak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
Kiremit Algoritması	3.1.8	Girdi-çıkı eşlemesini sağlamak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
Kule Algoritması	3.1.9	Girdi-çıkı eşlemesini sağlamak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
Upstart Algoritması	3.1.10	Hatasız sınıflama yapan YSA oluşturmak	Sınıflama	Bitiş Elemanı (EE) eklemiyor
CARVE	3.1.11	Sınıflama yapabilmek için problem boyutunu azaltmak	Sınıflama	Bitiş Elemanı (EE) eklemiyor
İleri Beslemeli Sinir Ağlarının Çapraz Geçerlilikle İnşa Edilmesi	3.1.12	Gizli Katmandaki optimal işlem elemanı sayısını ve başlangıç ağırlıklarını bulmak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
STRIP Ağ Büyüme Algoritması	3.1.13	Sınıflama yapabilmek için problem boyutunu azaltmak	Sınıflama	Bitiş Elemanı (EE) eklemiyor
Petrol Bulma Algoritması	3.1.14	Sınıflama yapabilmek için problem boyutunu azaltmak	Sınıflama	Bitiş Elemanı (EE) eklemiyor
Sınırlandırılmış Coulomb Enerji Ağı	3.1.15	Girdi-çıkı eşlemesini sağlamak için problemi ufak parçalara bölmek	Regresyon - Sınıflama	Bitiş Elemanı (EE) eklemiyor

Çizelge 4.1 (devam): Literatürdeki YSA algoritmalarının özeti.

Algoritma	Anlatıldığı Bölüm	Geliştirilme amacı	Kullanıldığı Alan	Bizim Amacımız İçin Yetersizliği
Rasgele Bağlantılandırılmış İleri Beslemeli Sinir Ağlarını Birleştirme	3.1.16	Girdi-çıkı eşlemesini yapıcı hem budayıcı bir yaklaşımla sağlamak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
Evrimsel Yapıcı ve Budayıcı Algoritma (ECPA)	3.1.17	Girdi-çıkı eşlemesini yapıcı hem budayıcı bir yaklaşımla sağlamak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
İleri Beslemeli Sinir Ağlarında Bir Yapı Optimizasyon Algoritması	3.1.18	Girdi-çıkı eşlemesini yapıcı hem budayıcı bir yaklaşımla sağlamak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
Sinir ağları için hızlı yapıcı-kaplayıcı algoritma	3.1.19	Her yeni işlem elemanı ile girdi hiper-uzayında genişleme sağlayarak girdi-çıkı eşlemesi yapmak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
C-mantec	3.1.20	N adet girdiyi tek bir bit'e dönüştüren fonksiyonlarla sınıflama yapmak	Sınıflama	Bitiş Elemanı (EE) eklemiyor, sadece iki sınıfı birbirinden ayırabiliyor
Budama Algoritmaları	3.2	YSA en iyi yapısını yeterince büyük bir yapıdan işlem elemanı azaltmak suretiyle bulmak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
Düzenleme Algoritmaları	3.3	YSA en iyi yapısını yeterince büyük bir yapıdan gereksiz bağlantıları sıfır olmaya zorlayarak oluşturmak	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor

Çizelge 4.1 (devam): Literatürdeki YSA algoritmalarının özeti.

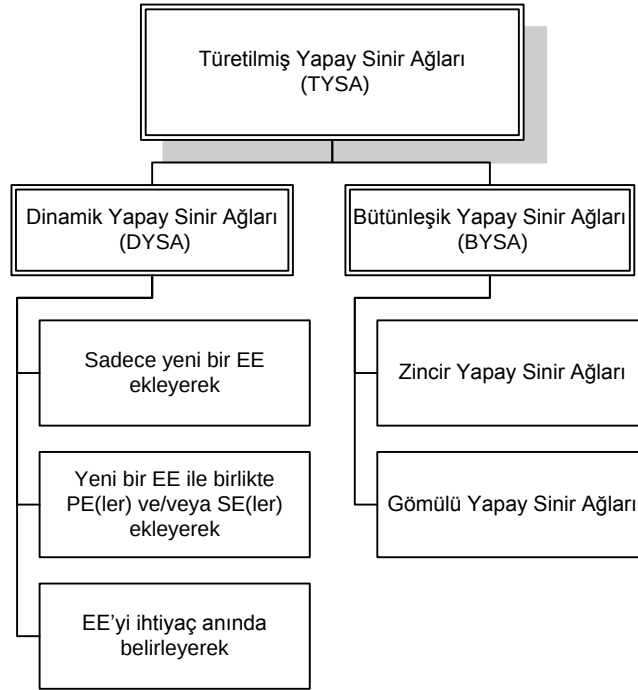
Algoritma	Anlatıldığı Bölüm	Geliştirilme amacı	Kullanıldığı Alan	Bizim Amacımız İçin Yetersizliği
Evrimsel Algoritmalar	3.4	YSA'nın tüm parametrelerini eniyilemek (öğrenme oranı, başlangıç ağırlıkları, PE sayısı vb.)	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor
Modüler Yapay Sinir Ağları	3.5	Girdi-çıkı eşlemesini sağlamak için problemi ufak parçalara bölmek ve karmaşık problemlere çözüm getirmek	Regresyon - Sınıflama	Bitiş Elemanı (EE) eklemiyor
Komite Makineleri	3.5	Girdi-çıkı eşlemesini sağlamak için problemi ufak parçalara bölmek ve karmaşık problemlere çözüm getirmek	Regresyon – Sınıflama	Bitiş Elemanı (EE) eklemiyor

4.1.1 Türetilmiş YSA

Yapay sinir ağlarının temel esin kaynağı olan hayvan beyninin en belirgin özelliklerinden birisi eski bilgiler üzerinden esinlenme yada ilham alma olarak adlandırılan yöntemle yeni sonuçlar (yeni çıktılar) üretebilmesidir. Mevcut YSA çalışmalarının biyolojik beyinden belki de en fazla uzaklaştığı ve zayıfladığı nokta birlikte çalışan veya yeni çıktılar üretebilen sinir ağlarıdır. Bu tip yapay sinir ağlarına yazında pek rastlanmamaktadır. Bu durumu daha iyi ifade edebilmek için daha önce açıklanan YSA tanımını temel alarak türetilmiş yapay sinir ağını şu şekilde tanımlayabiliriz;

Önceden eğitilmiş ve çalışır durumdaki bir YSA'nın mevcut yapısına en az bir yeni başlangıç elemanı (SE), işlem elemanı (PE) veya bitiş elemanı (EE) ekleyerek oluşturulan yeni modele *Türetilmiş Yapay Sinir Ağı (TYSA)* denir.

Türetilmiş YSA'ların potansiyel sınıflaması Şekil 4.1'de görülmektedir.



Şekil 4.1 : Türetilmiş yapay sinir ağlarının çeşitleri.

Türetilmiş yapay sinir ağları temel olarak bitiş elemanları ekleyip eklememesine ve birden fazla alt yapay sinir ağı bulundurup bulundurmadığına göre ikiye ayırabiliriz. Eğer mevcut bir yapay sinir ağına en az bir yeni bitiş elemanı ekleniyorsa, işlevsel

olarak yapay sinir ağı zaman içerisinde değişiyordur. Bu tip türetilmiş yapay sinir ağlarına *Dinamik Yapay Sinir Ağları (DYSA)* diyebiliriz. Eğer bitiş elemanı (EE) zaman içerisinde değişmiyorsa mevcut bir YSA'ya yeni bir bitiş elemanı eklemekten sadece mevcut yapıya SE'ler ve PE'ler ekleniyorsa veya birden fazla yapay sinir ağı ortak çalışıyorsa bu oluşan türetilmiş yapay sinir ağlarına da *Bütünleşik Yapay Sinir Ağları (BYSA)* diyebiliriz.

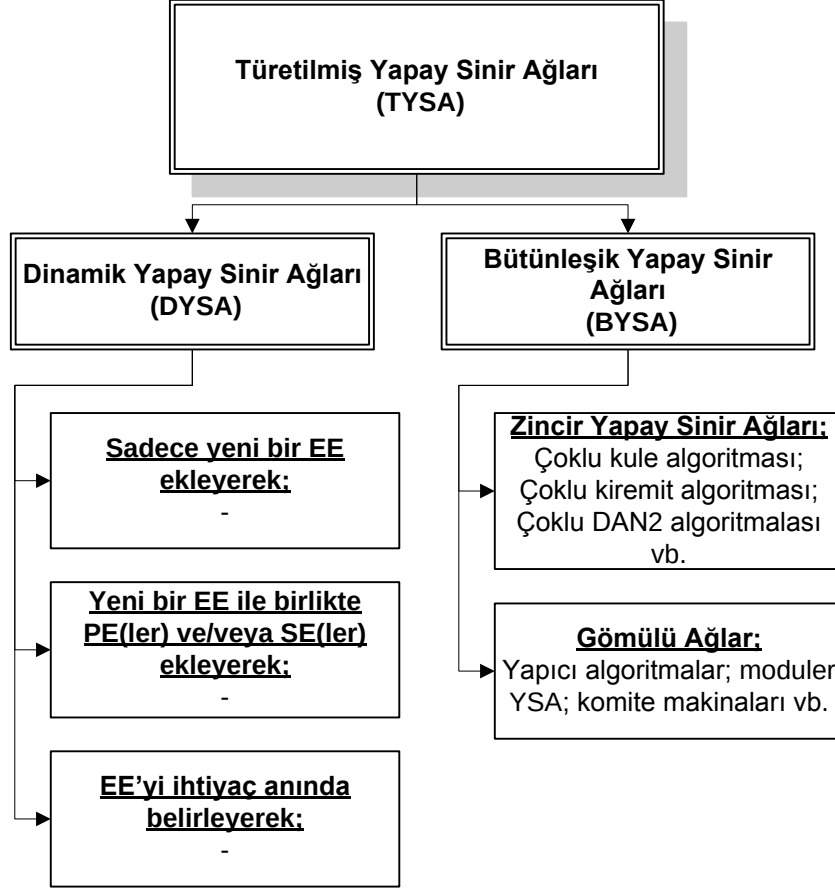
Dinamik yapay sinir ağları üç farklı şekilde oluşturabilirler;

1. Mevcut bir YSA'ya sadece yeni bir bitiş elemanı (EE) ekleyerek mevcut yapay sinir ağının yeni bir sonuç üretebilmesi, yeni bir işlev kazanması sağlanarak,
2. Mevcut bir YSA'ya yeni bir bitiş elemanı (EE) dışında PE(ler) ve/veya SE(ler) de ekleyerek mevcut yapay sinir ağının yeni bir sonuç üretebilmesi, yeni bir işlev kazanması sağlanarak,
3. Mevcut bir YSA bitiş elemanlarını (EE'leri) çevre koşullarına göre belirleyip, farklı koşullarda farklı EE'ler geliştirip zamanla farklı işlevler kazanarak.

Alavala'nın (2007) hibrit sistemleri sınıflamasını baz alarak sınıflandırırsak, iki farklı yapı kullanılarak bütünleşik yapay sinir ağlarını oluşturabiliriz;

1. Zincir Ağlar; tek başına çalışır haldeki farklı yapay sinir ağlarının su boruları gibi dizildiği ve bir yapay sinir ağının çıktısının diğer bir yapay sinir ağının girdisi olduğu yapılar,
2. Gömülü Ağlar; tek başına çalışır haldeki bir yapay sinir ağına EE eklemekten sadece SE(ler) ve/veya PE(ler) eklenerek yeniden oluşturulan yapılar.

Mevcut yazın, yukarıda verilen tanımlamalar kullanılarak yeni bir sınıflama halinde incelenebilir Şekil 4.2'deki yeni sınıflama ile açıkça görülmektedir ki dinamik olarak tanımladığımız yapay sinir ağı algoritmaları üzerinde çalışma yapılmamıştır.



Şekil 4.2 : Yapay sinir ağı algoritmalarının sınıflanması.

Yazındaki bu boşluğun belki de temel nedeni bu konu üzerinde daha çok bilgisayar bilimleri kökenli araştırmacıların çalışmış olmasıdır. Yazındaki birçok algoritma 0-1 ikili sistemde geliştirilmiş ve reel sistemlere genişletilmek suretiyle kullanılmıştır. Mevcut algoritmaların birçoğu aslında statik bir mimari üzerinde doğrusal veya doğrusal olmayan bileşenler kullanarak çözüm aramaktadırlar.

Bu algoritmalar çözümü ararken genellikle yapay sinir ağlarının başlangıç noktalarından biri olan paralel işlem (paralel computing) ve kara kutu davranışına zıt bir şekilde, ufak mantıklı parçalar ile yapay sinir ağlarını geliştirmeye çalışmaktadırlar. Bu da büyük resmin görülmeden yapay sinir ağlarının oluşturulmaya çalışılmasını, genelleme yapma yeteneği yerine tüm ayrıntıları kavramaya çalışmayı getirmektedir. Bu durum bazı problem türlerinde performans artışı sağlamıştır. Ancak bu özel ayrıntıları kavramaya yönelik yaklaşım doğal olarak farklı problemlerdeki genelleme yeteneğini sınırlayarak, pratikte uygulamalarda belirli alanlarla sınırlı kalmasına neden olabilmektedir.

Yapay sinir ağı algoritmaları incelendiğinde, evrimsel algoritmaların kullanılmaya başlaması ile parametre eniyilemesinde aşama kaydedilmiş ve yapay sinir ağındaki işlem birimleri problemin mantıklı ve daha ufak bir kısmını temsil etmeye zorlanmaktan çıkmıştır.

Yapılan tanım gereği türetilmiş yapay sinir ağları da tek başlarına bir yapay sinir ağıdır ve bir yapay sinir ağının taşıdığı tüm özelliklere sahiptir. Ancak, bir türetilmiş yapay sinir ağının yapay sinir ağına ek olarak aşağıdaki özellikleri de taşıması beklenmektedir:

- *Geçmiş Bilgilerin Kullanımı:* Bu özellik geçmiş bilgilerin üzerine yenilerinin eklenerek mevcut bilgi birikiminin artırılmasını ifade eder.
- *Yeni konular öğrenebilme:* Yeni yapay sinir hücreleri eklemek suretiyle mevcut YSA yeni bir YSA'ya dönüşebilmektedir. Bu da teorik olarak türetilmiş yapay sinir ağlarına sınırsız bir büyüme potansiyeli vermektedir.
- *Çıktı yapay sinir hücrelerinin işbirliğini engellememe:* Bu özellikle birlikte türetilmiş yapay sinir ağları içerisinde üretilen bir çıktı (yani anlamlı bir sonuç) diğerleri için girdi olarak kullanılabilir. Örnek olarak bir soğutma ünitesi ele alınabilir. Bu üniteye gelen malların cinsi bir YSA ile tespit edilip diğer bir YSA'ya o anki sıcaklık değeri gibi başka değişkenlerle birlikte girdi verilip soğutma sisteminin otomatik kontrolünü sağlanabilir.

Çalışmanın dördüncü bölümünde daha ayrıntılı açıklanacak olan türetilmiş yapay sinir ağları algoritmalarının kullanılmasının aşağıda belirtilen avantajları sağlaması beklenmektedir:

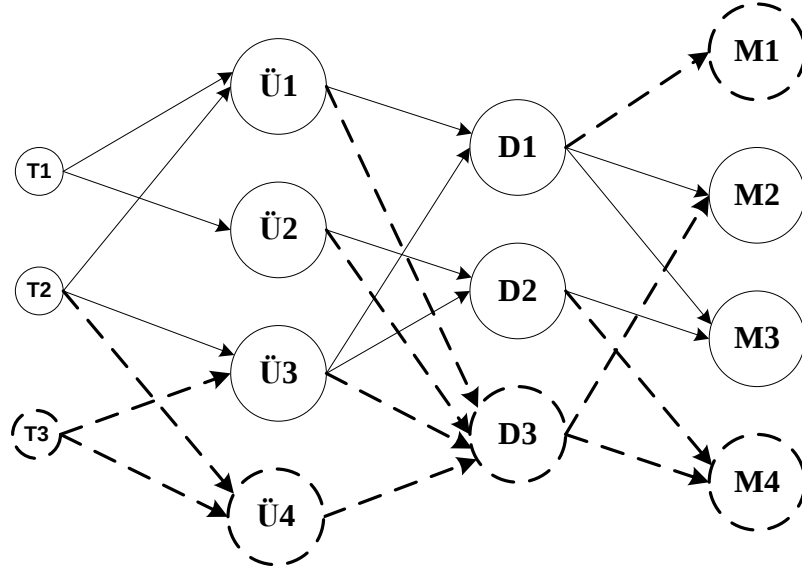
- Önceden kullanılmış bir YSA üzerine yeni bir tane oluşturmak uygun yapıyı arama sürecini kısıltacaktır (Ma ve Khorasani, 2004a, 2004b; Sharma ve Chandra (2010),
- Birçok ufak model yerine tek bir model kullanımı,
- Benzer veya aynı YSA'lar kullanıldığında daha basit bir yapı ile aynı işi yapabilme,

- YSA'lara yeni işlevler kazandırmak suretiyle yeni konularda bilgi ediniminin sağlanabilmesi,
- Farklı türlerdeki YSA'ların birlikte çalışarak daha karmaşık problemlere çözüm üretebilme potansiyeli (Azam, 2000),
- Biyolojik sinir sistemlerindeki yeni işlevler öğrenmenin, türetilmiş yapay sinir ağlarının canlı kanıtı olması (Haykin, 1999),
- Yapay sinir ağlarının öğrenme kapasitesinin arttırılması ve hata seviyesinin düşürülmesi (Qiang ve diğ., 2010),
- Karmaşık ilişkilerde, ilişkinin daha ufak ve daha basit kısımlarını ayrı YSA'lar kullanarak öğretip bunları bir YSA yardımıyla yönetmek (Haykin 1999; Azam 2000; Alpaydın, 2004).

Türetilmiş yapay sinir ağı çalışmalarının ileride potansiyel olarak karşılaşmasını beklenen muhtemel sorunları ise şöyle listeleyebiliriz:

- Önceden eğitilmiş bir YSA üzerinden devam edildiği için yerel optimumlara takılma olasılığı,
- Mevcut bir YSA'ya yeni bir çıktı yapay sinir hücresi eklediğinde hangi gelen ve giden bağlantıların ekleneceğinin belirlenmesi.

Bu tez kapsamındaki yapay sinir ağına genel yaklaşımı Şekil 4.3'teki gibi bir tedarik zinciri yapısına benzetebilir. Örneğin 2013 yılında şekildeki düz çizgilerle görülen yapıda bir tedarik zinciri olsun. T'ler tedarikçileri, Ü'ler üreticileri, D'ler dağıtım merkezlerini ve M'ler de müşterileri gösterebilir. Bu tedarik zinciri zamanla yeni tedarikçilerin, yeni üreticilerin, yeni dağıtım merkezlerinin ve yeni mağazaların piyasaya girmesiyle Şekil 4.3'te kesikli çizgilerle gösterildiği gibi genişleyebilir. Sistemdeki maliyetler değişebilir. Bu düşünce neden yapay sinir ağları oluşturulurken kullanılmıyor? Neden girdi ve çıktılar sabit tutulup sadece işlem elemanları, bazen de girdiler araştırılıyor? Neden araştırma yaparken çıktıları da zamanla değiştirerek mevcut yapay sinir ağları geliştirilmiyor?



Şekil 4.3 : Tedarik zinciri ağ gösterim örneği.

İşte tüm bu sorulara cevap verecek algoritmalar geliştirmek, önceki bölümlerde tanımlanan şekilde dinamik yapay sinir ağları oluşturan algoritmalar geliştirmek bu çalışmada üzerine yoğunlaşılan temel konudur.

4.1.2 Dinamik YSA

“*Dinamik*” kavramı yazında sıkça yer bulan ve değişkenliği ifade etmekte kullanılan bir kavramdır. Dinamik kavramı fizik bilimi içerisinde incelendiğinde dinamik olduğu ifade edilen nesnelerin zamana bağlı olarak değişim göstermesi gerekmektedir.

Bazı algoritmalar dinamik olduğunu ifade etmekle beraber bir kere YSA mimari yapısı oluşturulduktan sonra hiçbir şekilde mevcut yapıda değişiklik yapmamaktadır. Burada kullanılan dinamik kavramı tartışmaya son derece açıktır. Çünkü her nesne (ürün ya da model) kullanılabilir son halini almadan önce birtakım değişikliklere uğrar ama son halini aldıktan sonra genelde yapısını değiştirmez ve biz bu tip nesneleri (ürünleri ya da modelleri) dinamik olarak adlandırmayız. Örnek olarak bir gökdelenin inşaatını ele alalım, inşaat safhasında sürekli bina değişmekte, yeni katlar yeni odalar yapılmaktadır. Ancak bir kere binanın yapımı bittikten sonra (normalde) yeni katlar yeni odalar oluşturulmamaktadır. Burada binanın yapımı sırasında geçirdiği değişiklikleri göz önüne alıp hiç bir zaman “*dinamik mimarili gökdelen*” olarak adlandırılmamaktadır. Ya da *dinamik mimarili gökdelen* denildiğinde beklenen durum odaların ve/veya katların

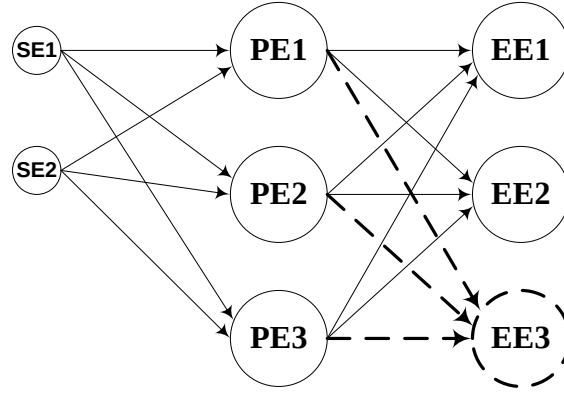
sürekli olarak değişmesi (metrekare, dekor, şekil vb. açılardan) beklenmektedir. Bu yaklaşımla incelendiğinde yazındaki mevcut yapay sinir ağı mimari yapılarını oluşturulurken yaşadığı değişiklikler nedeniyle dinamik olarak adlandırılmaması gerekir.

Dinamik yapay sinir ağlarındaki dinamik kavramı başlı başına bir tartışma konusudur ve bu tezin kapsamı dışındadır. Ancak bu çalışmada kullanılan dinamik kavramını yazındaki diğer kullanımlardan ayırmak amacıyla yukarıdaki örnek verilmiştir. Bu çalışmada bahsedilen dinamiklik bir yapay sinir ağının farklı zamanlarda farklı bitiş elemanlarına (EE) sahip olması sebebiyle zaman içerisinde farklı türde sonuçlar elde etmek için kullanılması ve zamanla değişen işlevlere sahip olmasıdır. Dolayısıyla bu çalışmada geçen dinamik yapay sinir ağını şu şekilde tanımlayabiliriz:

Çevre şartlarına veya ihtiyaçlara bağlı olarak bir YSA'da kullanılan bitiş eleman(lar)ı (EE'ları) sayı veya işlev olarak değişiyorsa bu tür YSA'lara *Dinamik Yapay Sinir Ağı (DYSA)* denir.

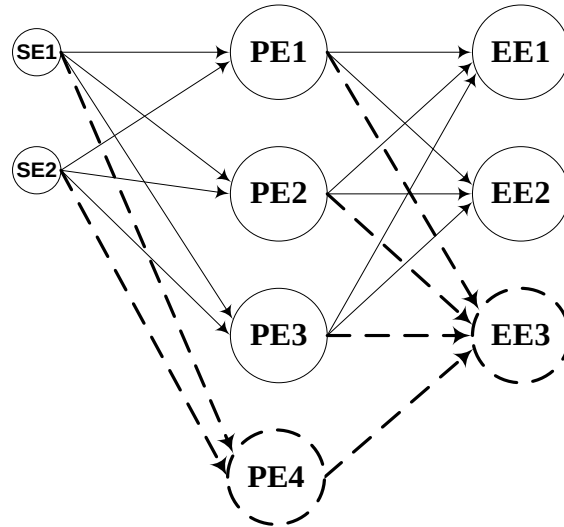
DYSA oluşturmak için kullanılabilecek ilk yol mevcut bir YSA'ya sadece yeni bir bitiş elemanı (EE) ekleyerek mevcut yapay sinir ağının yeni bir sonuç üretebilmesi, yeni bir işlev kazanmasını sağlamaktır. Şekil 4.4'te eğitilmiş bir YSA'ya yeni bir EE eklenmesi durumu gösterilmiştir. Bu tip eklemeler benzer çıktıların istendiği YSA'larında kullanılabilir.

Örneğin Şekil 4.4'teki yapı bir döviz kurunun bir saat (EE1) ve iki saat (EE2) sonraki değerini tahmin etmek üzere eğitilmiş olsun. SE1 ve SE2 sırasıyla şu anki döviz kuru değerini ve bir saat önceki döviz kuru değerini girdi olarak kullansın. PE1, PE2 ve PE3 gizli katmandaki işlem elemanları olsunlar. Kesikli çizgilerle gösterilen kısımlar ise sonradan eklenmiş parçaları gösterebilir. Yeni eklenen EE3 aynı döviz kurunun 3 saat sonraki değerini tahmin etmek üzere kullanılabilir, çünkü aynı girdileri ve benzer bir gizli katmanı kullanacaktır.



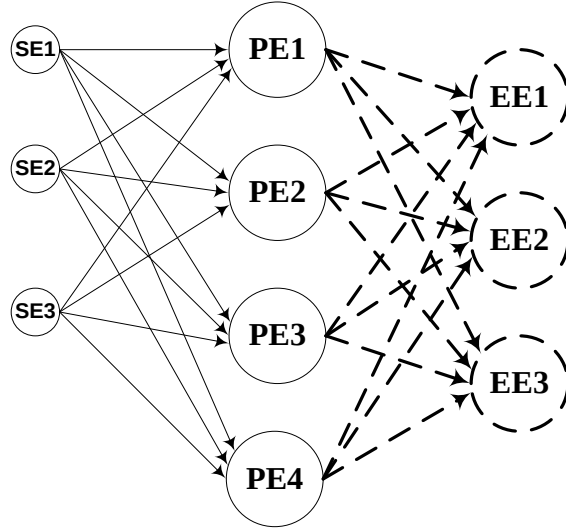
Şekil 4.4 : Eğitilmiş bir YSA'ya yeni bir EE ekleme örneği.

Şekil 4.4'teki örnekten devam edersek Şekil 4.5'teki yapı bir döviz kurunun bir saat (EE1) ve iki saat (EE2) sonraki değerini tahmin etmek üzere eğitilmiş olsun. SE1 ve SE2 sırasıyla şu anki döviz kuru değerini ve bir saat önceki döviz kuru değerini girdi olarak kullansın. PE1, PE2 ve PE3 gizli katmandaki işlem elemanları olsunlar. Kesikli çizgilerle gösterilen kısımlar ise sonradan eklenmiş parçaları gösterebilir. Yeni eklenen EE3'ü tahmin etmek için sadece bir tane EE eklemek yeterli olmayabilir. Belki aynı döviz kurunun 3 saat sonraki değerini tahmin etmek için üç saat önceki değeri gereklidir. Bu durumda Şekil 4.5'te görülen yeni yapı kullanılabilir, çünkü benzer girdileri ve benzer PE'ler kullanacaktır. Bu da DYSA oluşturmak için diğer bir yol olacaktır.



Şekil 4.5 : Eğitilmiş bir YSA'ya yeni bir EE yanında yeni bir PE ekleme örneği.

DYSA oluřturmanın dięer bir yolu da bitiř elemanlarına (EE'lere) kadar olan yapının tespit edilip son elemanın o anki ihtiyalara gre seilip aęırlıklarının tespit edilmesidir. Yine dviz kuru tahmininden rnek verilecek olursa; řekil 4.6'da grlen YSA'da SE1, SE2, SE3 sırasıyla řu anki, bir saat nceki ve iki saat nceki dviz kuru deęerlerini girdi olarak kullansın. PE1, PE2, PE3, PE4 gizli katmandaki iřlem elemanları olsunlar. Bu řekilde YSA eęitilip (rneęin girdilerdeki izler kullanılarak) sonrasında lazım olan tahmin bilgisine gre sonradan EE1, EE2 ve EE3 bitiř elemanları sırasıyla bir saat sonraki, iki saat sonraki ve  saat sonraki dviz kuru tahminlerini gstermek zere o an ihtiya duyulan EE'yi bir algoritmayla oluřturarak kullanılabilir.



řekil 4.6 : EE'nin ihtiyaa ve řartlara gre belirlendięi YSA rneęi.

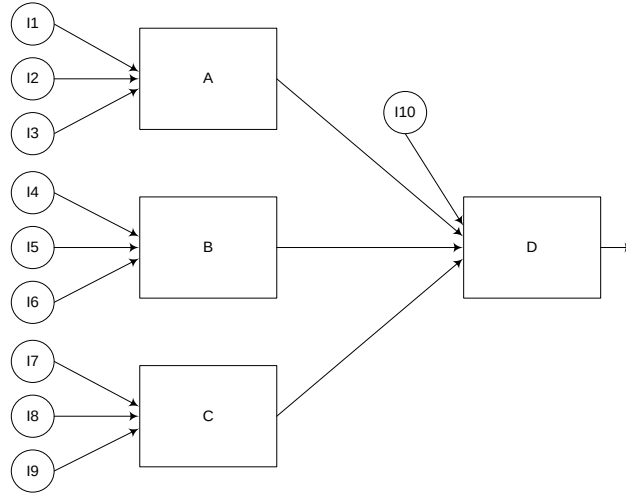
Dinamik yapay sinir aęı olarak deęerlendirebilecek yapılar yapay sinir aęının uygun mimarisinin (gizli katman sayısı, PE sayısı) tespitinde de kullanılabilir. rneęin iki girdi deęerini alıp  sonu retilmesi gereken bir problem verilmiř olsun. Bu durumda verilen yapı aslında $R^2 \rightarrow R^3$ bir haritalamadır (mapping). Mevcut yapay sinir aęı algoritmaları iki girdiyi ve  sonucu ele alıp arasındaki katmanları doldurmaya alıřır yani direkt $R^2 \rightarrow R^3$ 'e bir haritalama oluřturmaya alıřırlar. Dinamik olarak deęerlendirecek bir algoritma ise rneęin řekil 4.5'teki gibi nce $R^2 \rightarrow R^2$ 'ye bir haritalama yapıp sonra bunu $R^2 \rightarrow R^{2+1} \Rightarrow R^2 \rightarrow R^3$ olacak řekilde geliřtirebilir. Bu durumda ęrenme algoritması, alıřan ve $R^2 \rightarrow R^2$ 'ye bir haritalama yapan bir ysa'ya

kullanılmak üzere yeni bir işlev (ya da EE) ekleyerek bunu $R^2 \rightarrow R^3$ bir haritalamaya dönüştürdüğü için dinamik yapay sinir ağı oluşturmuş olacaktır.

4.1.3 Bütünleşik YSA

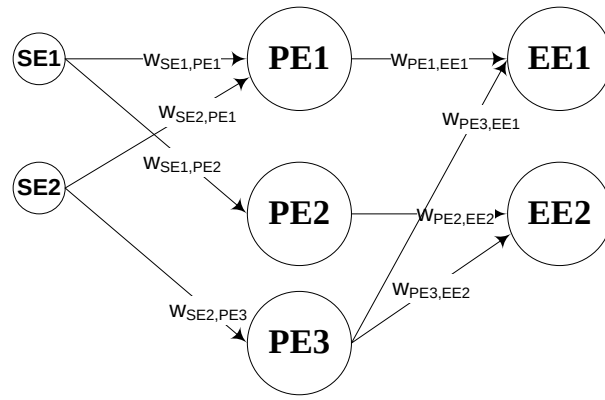
Hâlihazırda eğitilmiş ve kendi başına çalışır haldeki A ve B yapay sinir ağları bulunsun. A yapay sinir ağına B yapay sinir ağı birinin çıktıları diğerinin girdileri olacak şekilde birleştirilerek C yapay sinir ağı oluşturulsun veya mevcut A yapay sinir ağına hatayı azaltmak için yeni bir bitiş elemanı eklemekten yeni başlangıç veya işlem elemanları eklenerek C yapay sinir ağı oluşturulsun. Bu tip C yapay sinir ağlarına *Bütünleşik Yapay Sinir Ağı (BYSA)* denir. C yapay sinir ağının oluşturulması sırasında A ve B yapay sinir ağı birisinin çıktısı diğerinin girdisi olacak şekilde birleştirilmiş olabilir veya A yapay sinir ağına yeni PE'ları ve SE'ları kullanmak suretiyle geliştirilmiş olabilir. Eğer C yapay sinir ağı A ve B ağlarının birisinin çıktıları diğerinin girdileri olacak şekilde birleştirilmişse bu tip yapay sinir ağlarına *Zincir Yapay Sinir Ağı (ZYSA)* denir. Eğer C yapay sinir ağı A yapay sinir ağına yeni SE(ları) veya PE(ları) eklemek suretiyle elde edilmişse bu tip yapay sinir ağlarına *Gömülü Yapay Sinir Ağları (GYSA)* denir.

Basit bir şekilde Şekil 4.7'den zincir yapay sinir ağının örneğini görebiliriz. A, B, C ve D kendi başlarına çalışan yapay sinir ağlarını I'lar ise farklı girdileri göstermektedir. A, B, C ve D'nin bütünleşik olarak çalışmasıyla yeni bir model, Şekil 4.7'de görülen zincir yapay sinir ağları oluşturulmuştur.

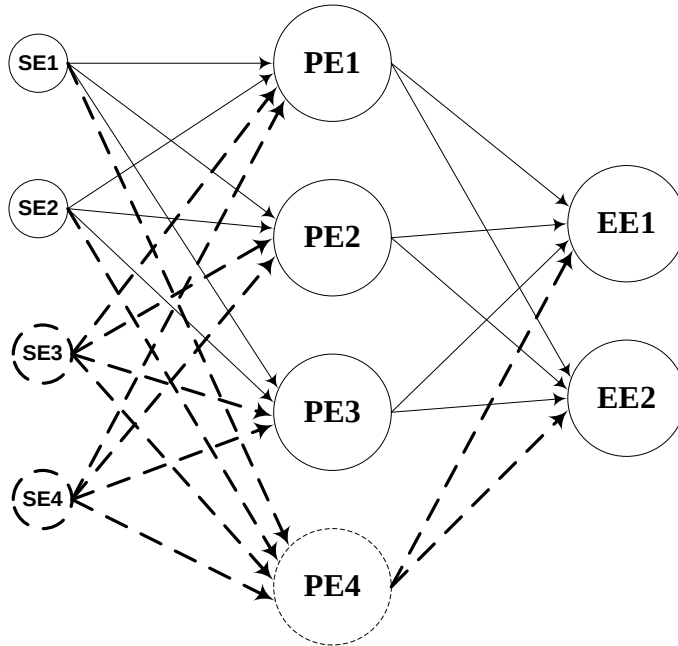


Şekil 4.7 : Zincir yapay sinir ağı örneği.

Şekil 4.8’de bulunan YSA’da SE_i i . başlangıç elemanını, w_{xy} ; x düğümünden y düğümüne olan bağlantının ağırlığını, PE_i i . işlem elemanını, EE_i i . bitiş elemanını gösterebilir. Her iki YSA’daki aynı numaralı PE’ler aynı transfer fonksiyonuna sahip işlem elemanları olsun. Bu YSA’ya hatayı azaltmak amacıyla yeni SE’ler ve PE’ler eklemek suretiyle Şekil 4.9’da görülen yeni YSA oluşturulabilir. Şekil 4.8’de görülen kesikli çizgiler Şekil 4.9’da görülen YSA’ya eklenen kısımları düz çizgiler ise Şekil 4.8’de görülen YSA yapısını göstermektedir. Böylece mevcut bir YSA’nın mevcut hata miktarı yeni eklenen girdilerle ve işlem elemanlarıyla azaltılabilir.

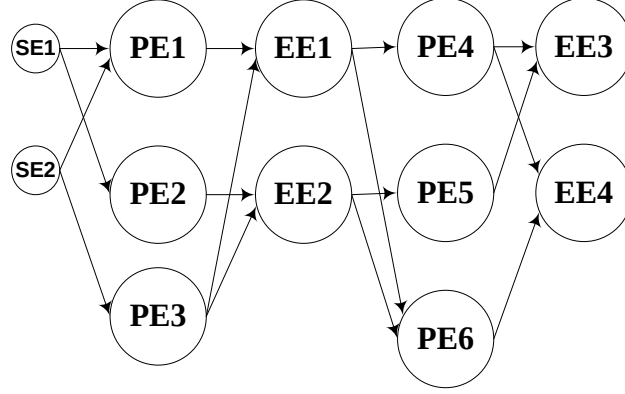


Şekil 4.8 : Örnek YSA.



Şekil 4.9 : Gömülü Ağ örneği.

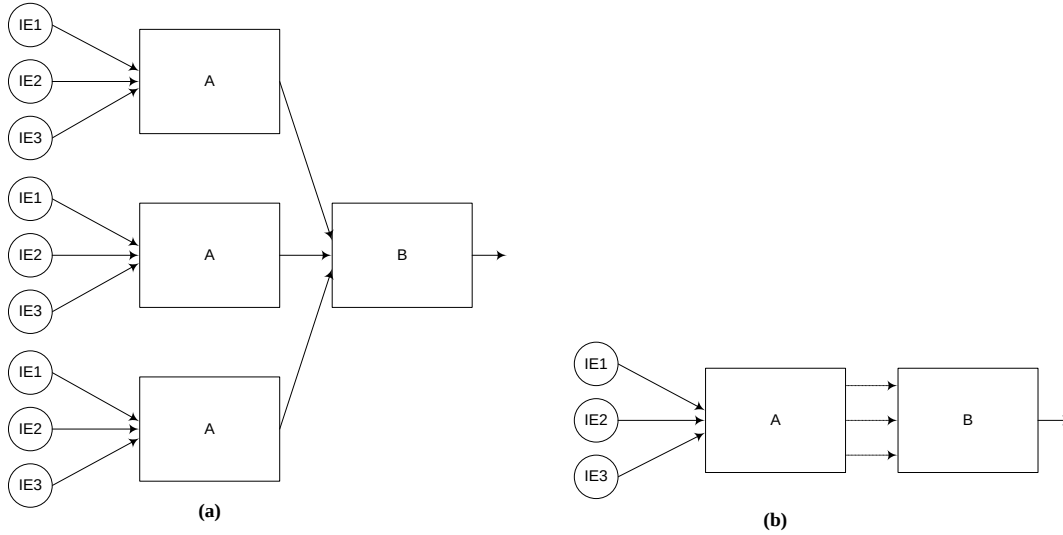
Zincir yapay sinir ağları ise paralel olabileceği gibi bir YSA'nın çıktısını başka bir YSA'da girdi olarak kullanmak üzere Şekil 4.10'daki gibi seri olarak da bağlanmasıyla da oluşturulabilir.



Şekil 4.10 : Seri olarak birleştirilmiş YSA örneği.

Zincir Yapay sinir ağlarını oluşturan ve aynı YSA'nın tekrarında kullanmak için *Geçici Bağlantılar* kullanabilir. Bu geçici bağlantılar YSA modellerinde her zaman bir uyarılmayla yapıda (çizgede) bir akış başlamasından ve en son bir çıktı üretilmesiyle yapıdaki tüm akışların kesilmesinden dolayı gereklidir.

Geçici bağlantılar birbirinin aynısı olan YSA'larının sadece bir tanesinin kullanılmasını sağlayarak ZYSA'nın yapısal karmaşıklığı azaltacaktır. Şekil 4.11'de geçici bağlantıların kullanımından bir örnek verilmiştir. Şekil 4.11'de bulunan BYSA (a) üç tane A ve bir tane B olmak dört adet A-YSA'dan oluşmaktadır. A yapay sinir ağı her seferinde aynı türde 3 girdiyi 3 farklı gözlem değerini kullanarak toplamda 3 farklı sonuç üreterek bunları B yapay sinir ağına girdi olarak vermektedir. Bunun yerine Şekil 4.11 (b)'de bulunan BYSA ise sadece bir adet A yapay sinir ağının geçici bağlantılarla B yapay sinir ağına bağlanmasından oluşmuştur. Dolayısıyla A yapay sinir ağı her seferinde bir geçici bağlantıyı kullanarak B yapay sinir ağına 3 adet akış gönderecektir.



Şekil 4.11 : Benzer YSA yapısının (a) geçici bağlantılarla elde edilmesi (b).

4.2 Dinamik Yapay Sinir Ağı Algoritmaları

Daha önce yapılan tanımlar gereği bir dinamik yapay sinir ağını sadece yapısına yeni bir bitiş elemanı ekleyerek elde edilebileceği üzerinde durulmuştu. Bu bölümden itibaren bu bitiş elemanı ekleme işleminin nasıl yapılabileceğine dair çalışmalar yapılacak ve bunları sağlayan algoritmalar önerilecektir.

Önerilen algoritmalar temel olarak çok katmanlı algılayıcılar üzerinde çalışacak veya mevcut yapıyı tek gizli katmanlı ÇKA kullanarak bu ÇKA yapısı üzerinde dinamik yapay sinir ağını oluşturacaktır.

ÇKA yapısının seçilme nedenlerini şöyle sıralayabiliriz;

1. Yapay sinir ağı denildiğinde birçok yazında ileri beslemeli ve geri yayılım algoritması ile kavrama yapılmış olan çok katmanlı algılayıcıların (ÇKA) kastedilmesi,
2. ÇKA'ların tercihinde genel kabul görmüş yapı olmalarının yanında şahsi deneyim ve gözlemler,
3. ÇKA yapılarının biyolojik sinir ağları ile yüksek benzerliği,
4. Tek gizli katmanlı ÇKA yapısının uygun parametrelerle herhangi bir sürekli çok değişkenli fonksiyonu modelleyebilmesi (Cybenko, 1989; Hornik ve diğ., 1989,

Funahashi, 1989; Hornik ve diğ., 1990; Funahashi, 1990;Hornik, 1991; Hornik, 1993),

5. ÇKA yapısında sonucu en çok etkileyen bağlantıların son iki katman (son gizli katman ile çıktı katmanı) arasında olması (Principe ve diğ., 1999) nedeniyle DYSA için ciddi bir potansiyel oluşturmaları.

Bu çalışma kapsamında Şekil 4.1’de verilen ve Şekil 4.2’de mevcut yazındaki çalışma olmadığı gösterilen her bir tür DYSA için algoritma önerilmiştir.

Önerilen ilk DYSA algoritması *Temel Algoritma 01 Verisyon 01* (TA01V01) mevcut bir YSA’ya sadece yeni bir bitiş elemanı (EE) ekleyerek mevcut yapay sinir ağının yeni bir sonuç üretebilmesi, yeni bir işlev kazanmasını sağlamaya çalışmaktadır.

Önerilen ikinci DYSA algoritması *Temel Algoritma 02 Verisyon 01* (TA02V01) mevcut bir YSA’ya yeni bir bitiş elemanı (EE) dışında PE(ler) ve/veya SE(ler) de ekleyerek mevcut yapay sinir ağının yeni bir sonuç üretebilmesi, yeni bir işlev kazanmasını sağlamaya çalışmaktadır.

Önerilen üçüncü DYSA algoritması *Temel Algoritma 03 Verisyon 01* (TA03V01) mevcut bir YSA bitiş elemanlarını (EE’leri) çevre koşullarına göre belirleyip, farklı koşullarda farklı EE’ler geliştirip zamanla farklı işlevler kazandırma amacına yönelik olarak geliştirilmiştir. Bu amaçla öncelikle gözlemlerdeki belli izleri tespit etmek maksadıyla özellikle Kohonen’in (1984) özörgütlemeli haritası ile kümeleme yapılmıştır. Oluşan bu özörgütlemeli harita yapısı son katmanı ÇKA’nın gizli katmanı olacak şekilde ÇKA’ya dönüştürülmüştür. Böylece çıktı katmanı boş bir ÇKA elde edilmiş ve bu yapı TA01V01 algoritmasına başlangıç yapısı olarak verilmiştir.

Önerilen dördüncü DYSA algoritması *Temel Algoritma 03 Verisyon 02* (TA03V02) de üçüncü algoritmaya benzer şekilde mevcut bir YSA bitiş elemanlarını (EE’leri) çevre koşullarına göre belirleyip, farklı koşullarda farklı EE’ler geliştirip zamanla farklı işlevler kazandırma amacına yönelik olarak geliştirilmiştir. TA03V01’e benzer şekilde Kohonen’in (1984) özörgütlemeli haritası ile kümeleme yapılarak oluşan yapı ÇKA yapısına dönüştürülmüştür. Böylece çıktı katmanı boş bir ÇKA elde edilmiş ve bu yapı TA02V01 algoritmasına başlangıç yapısı olarak verilmiştir.

Böylece TA03V01 algoritması TA01V01 algoritmasını kullanarak sadece EE eklerken, TA03V02 algoritması TA02V01 algoritmasını kullanmak suratiyle hem EE hem de PE ekleyebilmektedir.

4.2.1 Temel algoritma 1: sadece bitiş elemanı ekleme

Temel Algoritma 1 Versiyon 1 (TA01V01) ile yapay sinir ağlarına sonradan çıktı sinir hücreleri ekleyebilecek, veya yapay sinir ağı yapısı oluşturulurken çıktı sinir hücresi sayısını en baştan sabit tutmak yerine, eğitim aşamaları sırasında tek tek ekleyen bir algoritma olarak sunulmuştur (Guresen ve Kayakutlu, 2013a).

TA01V01 algoritması mevcut eğitilmiş ve çalışır haldeki bir yapay sinir ağına yeni bitiş ekleyebilecek özelliktedir. Bu çalışmada çok katmanlı algılayıcılar (Multi-Layer Perceptron) ve geri yayılım algoritması (backpropagation) kullanılarak gözetimli eğitim için TA01V01 için yapılmıştır. Şekil 4.12’de TA01V01 algoritmasının sözde programı verilmiştir.

Adım 1: Öğrenmenin gerçekleşeceği bir yapay sinir ağı al/ yoksa çıktı katmanı boş bir ÇKA oluştur

Adım 2: Veri kümesinden K’nın L’li çapraz geçerlilik veri kümelerini oluştur

Adım 3: Algoritmanın durma koşullarını tanımla

Adım 4: Eğitim ve çapraz geçerlilik veri kümelerini hazırla

Adım 5: **Seçili YSA’ya** Eklenecek EE için Temel Algoritma 1 Aday Havuzu hazırla

Adım 6: Adaylarının uygunluğunu K’nın L’li çapraz geçerliği için hesapla

Adım 7: En uygun EE adayını aday havuzu içinden seç ve mevcut yapay sinir ağı yapısına ekle

Adım 8: Yeni oluşan yapay sinir ağını GYA ile eğit

Adım 9: Durma koşullarının sağlanıp sağlanmadığını kontrol et

Adım 10: Oluşan YSA’yı seç durma koşulları sağlanmıyorsa Adım 4’e git, sağlanıyorsa Adım 11’e git

Adım 11: Seçili yapay sinir ağını sonuç olarak ver

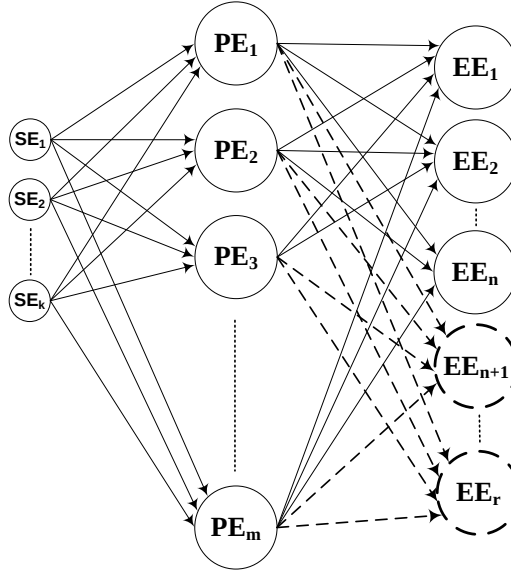
Şekil 4.12 : Temel Algoritma 1’in sözde programı.

Şekil 4.12’de belirtildiği üzere algoritma temel olarak Adım 4 ile Adım 10 arasında döngüye girmekte ve her seferinde mevcut YSA’ya yeni bir EE eklemektedir.

Bu döngü sırasında K 'nın L 'li çapraz geçerliliği için verileri hazırlamaktadır. K 'nın L 'li çapraz geçerliliği ilerleyen bölümlerde ayrıntılı olarak ele alınacaktır. Adım 5'te algoritmanın yapısına katılacak aday bitiş elemanları için bir aday havuzu oluşturup bunları hazırlamış olduğu K 'nın L 'li çapraz geçerliliği verileriyle değerlendirmektedir. Daha sonra en uygun adayı seçip GYA ile eğitmektedir. GYA ile eğitimde ise her seferinde L adet çapraz geçerlilik kümesi hariç kalan tüm eğitim alt kümeleri ile eğitim yapmakta L adet kümeyi ise birleştirip çapraz geçerlilikte kullanmaktadır.

Son olarak durma koşulları kontrol edilmekte, durma koşulları sağlanmamış ise Adım 4'e geri dönmekte, durma koşulları sağlanmış ise oluşan yapay sinir ağı sonuç olarak verilmektedir.

TA01V01 algoritmasının çalışma sistemi Şekil 4.13'te görülmektedir. Burada kesikli çizgilerle gösterilen yapılar sonradan TA01V01 algoritması ile eklenen yapılardır. Bir yapay sinir ağının n adet bitiş elemanı olduğu varsayıldığında TA01V01 sırayla n 'inci bitiş elemanından itibaren her iterasyonda yeni bir bitiş elemanı olmak üzere istenilen miktarda yeni bitiş elemanı ekleyebilmektedir.



Şekil 4.13 : TA01V01 algoritmasının çalışma örneği.

4.2.2 Temel algoritma 2: bitiş elemanı ve işlem elemanı ekleme

Temel Algoritma 2 Versiyon 1 (TA02V01) ile yapay sinir ağlarına sonradan çıktı sinir hücreleri ekleyebilecek veya yapay sinir ağı yapısı oluşturulurken çıktı sinir hücresi

sayısını en baştan sabit tutmak yerine, eğitim aşamaları sırasında tek tek ekleyen bir algoritma sunulmuştur. TA01V01'den temel farkı bitiş elemanı eklerken aynı zamanda gizli katmana işlem elemanı da ekleyebilmesidir.

TA02V01 algoritması mevcut eğitilmiş ve çalışır haldeki bir yapay sinir ağına yeni bitiş elemanı ve işlem elemanı ekleyebilecek özelliktedir. Bu çalışmada çok katmanlı algılayıcılar (Multi-Layer Perceptron) için geri yayılım algoritması (backpropagation) kullanılarak gözetimli eğitim için TA02V01 algoritması önerilmiştir. Şekil 4.14'te TA02V01 algoritmasının sözde programı verilmiştir.

Adım 1: Öğrenmenin gerçekleşeceği bir yapay sinir ağı al/ yoksa çıktı katmanı boş bir ÇKA oluştur

Adım 2: Veri kümesinden K' 'nin L' 'li çapraz geçerlilik veri kümelerini oluştur

Adım 3: Algoritmanın durma koşullarını tanımla

Adım 4: Eğitim ve çapraz geçerlilik veri kümelerini hazırla

Adım 5: **Seçili YSA'ya** Eklenecek EE ve PE.ler için Temel Algoritma 2 Aday Havuzu hazırla

Adım 6: Adaylarının uygunluğunu K' 'nin L' 'li çapraz geçerliliği için hesapla

Adım 7: En uygun EE ve PE.ler grubunu aday havuzu içinden seç ve mevcut yapay sinir ağı yapısına ekle

Adım 8: Yeni oluşan yapay sinir ağını GYA ile eğit

Adım 9: Durma koşullarının sağlanıp sağlanmadığını kontrol et

Adım 10: Oluşan YSA'yı seç durma koşulları sağlanmıyorsa Adım 4'e git, sağlanıyorsa Adım 11'e git

Adım 11: Seçili yapay sinir ağını sonuç olarak ver

Şekil 4.14 : Temel Algoritma 2'nin sözde programı.

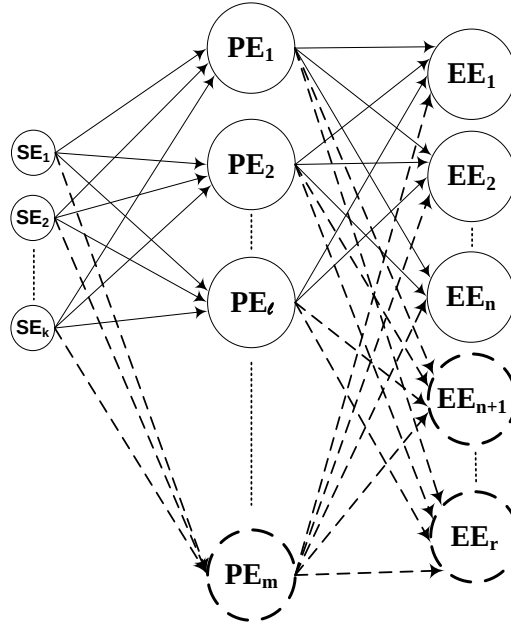
Şekil 4.14'te belirtildiği üzere algoritma temel olarak Adım 4 ile Adım 11 arasında döngüye girmekte ve her seferinde mevcut YSA'ya yeni bir EE eklemektedir.

Bu döngü sırasında K' 'nin L' 'li çapraz geçerliliği için verileri hazırlamaktadır. sırasında K' 'nin L' 'li çapraz geçerliliği ilerleyen bölümlerde ayrıntılı olarak ele alınacaktır. Adım 5'te algoritmanın yapısına katılacak aday bitiş elemanları ve aday bitiş elemanlarıyla birlikte eklenebilecek işlem elemanları için bir aday havuzu oluşturup bunları hazırlamış olduğu K' 'nin L' 'li çapraz geçerliliği verileriyle değerlendirmektedir. Aday havuzu ve aday havuzu oluşturma yöntemlerinin ayrıntıları ilerleyen bölümlerde belirtilecektir.

Adım 7’de en uygun adayı seçip GYA ile eğitmektedir. GYA ile eğitimde ise her seferinde L adet çarpaz geçerlilik kümesi hariç kalan tüm eğitim alt kümeleri ile eğitim yapmakta L adet kümeyi ise birleştirip çarpaz geçerlilikte kullanmaktadır.

Son olarak durma koşulları kontrol edilmekte, durma koşulları sağlanmamış ise Adım 4’e geri dönmekte, durma koşulları sağlanmış ise oluşan yapay sinir ağı sonuç olarak verilmektedir.

TA02V01 algoritmasının çalışma sistemi Şekil 4.15’te görülmektedir. Burada kesikli çizgilerle gösterilen yapılar sonradan TA02V01 algoritması ile eklebilen yapılardır. Bir yapay sinir ağının n adet bitiş elemanı olduğu varsayıldığında TA02V01 sırayla n ’inci bitiş elemanından itibaren her iterasyonda yeni bir bitiş elemanı olmak üzere istenilen miktarda yeni bitiş elemanı ekleyebilmektedir. İhtiyaç duyulması halinde gizli katmandaki l adet işlem elemanını, bitiş elemanı eklerken arttırabilmektedir. TA02V01 algoritması her defasında bir bitiş elemanı eklemekte, eğer performans göstergelerini iyileştiriyorsa bu bitiş elemanı ile birlikte işlem elemanları da eklemektedir.



Şekil 4.15 : TA02V01 algoritmasının çalışma örneği.

4.2.3 Temel algoritma 3: bitiş elemanını ihtiyaç anında belirleme

Önerilen üçüncü DYSA algoritması *Temel Algoritma 03 Versiyon 01* (TA03V01) ve dördüncü DYSA algoritması *Temel Algoritma 03 Versiyon 02* (TA03V02) benzer

özellikler taşımaktadırlar. Her iki algoritma da gözlemlerdeki belli izleri tespit etmek maksadıyla öncelikle Kohonen'in (1984) özörgütlemeli haritası ile kümeleme yapmaktadır.

Öz örgütlemeli haritalar (ÖÖH) ile kümeleme yapılırken kaç küme ya da kaç kaçlık $(m \times n)$ bir kümeleme matrisi kullanılması gerektiği sorusu gündeme gelmektedir. Bu çalışma kapsamında öz örgütlemeli haritanın boyutunun en fazla $m \times n \leq z$ olacak şekilde 1×2 'den $(m \times n)$ 'e kadar tüm boyutları değerlendirilmeye alınmaktadır. Burada m özörgütlemeli haritanın birinci boyutu, n özörgütlemeli haritanın ikinci boyutu ve z istenen bitiş elemanı sayısı ile başlangıç elemanı sayısının toplamı olarak alınmıştır.

z değerinin istenen bitiş elemanı sayısı ile başlangıç elemanı sayısının toplamı olarak alınmasında kullanılan sezgisel yaklaşım oluşturulacak haritalamanın en fazla bitiş elemanı sayısı kadar boyuta gizli katmanda indirilmesinin/çıkarılmasının yeterli olacağı varsayımına dayanmaktadır. İşlemsel olarak çok yük getirmeyeceği ve en uygun yapıyı arama sırasında arama süresini düşürebileceği değerlendirildiğinden gizli katmanda izin verilen max boyut (işlem elemanı sayısı) bitiş elemanı sayısı ve başlangıç elemanı sayısının toplamına gevşetilmiştir.

Oluşan bu özörgütlemeli haritalar yapısı son katmanı ÇKA'nın gizli katmanı olacak şekilde ÇKA'ya dönüştürülmüştür. Böylece çıktı katmanı boş bir ÇKA elde edilmiş ve bu yapı algoritmaların kalan kısmına başlangıç yapıları olarak verilmiştir.

TA03V01 için kalan kısımda TA01V01 algoritması ile sadece bitiş elemanı ekleme işlemi yapılmış TA03V02 için ise kalan kısımda TA02V01 algoritması ile bitiş elemanı ve işlem elemanı ekleme işlemi yapılmıştır. Şekil 4.16'da TA03V01 algoritmasının sözde programı verilmiştir.

Adım 1: ÖÖH'lerin alabileceği en büyük boyutu tespit et.

Adım 2: Veri kümesini kullanarak 1×2 'den en büyük boyuta kadar farklı boyularda ÖÖH oluştur.

Adım 3: Oluşan ÖÖH'leri ÖÖH havuzuna ekle.

Adım 4: ÖÖH havuzundaki YSA'ları ÇKA'ya çevir.

Adım 5: Oluşan ÇKA'ları ÇKA başlangıç aday havuzuna ekle.

Adım 6: ÇKA başlangıç AH'ndaki her bir ÇKA'yı TA01V01 için başlangıç YSA olarak ver.

Adım 7: Her bir TA01V01 sonucunda oluşan YSA'yı ÇKA AH'na ekle.

Adım 8: Adaylarının uygunluğunu K' 'nin L' 'li çapraz geçerliliği için hesapla

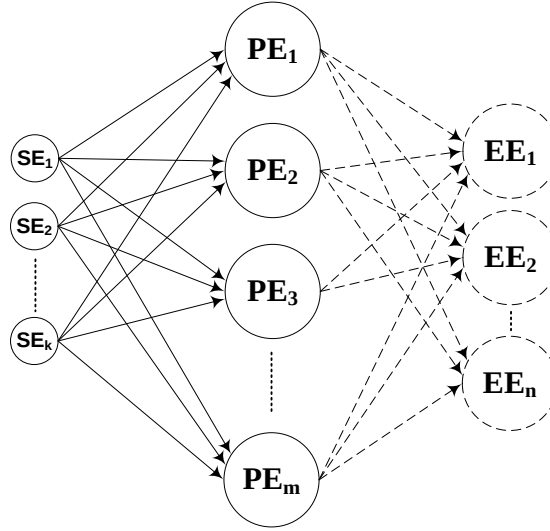
Adım 9: En uygun adayı sonuç olarak ver

Şekil 4.16 : Temel Algoritma 3 Versiyon 1'in sözde programı.

Şekil 4.16'da belirtildiği üzere algoritma önce öz örgütlemeli haritalar için müsaade edilen en büyük boyutu hesaplamakta daha sonra 1×2 boyutundan müsaade edilen en büyük boyuta kadar belirtilen miktarda öz örgütlemeli harita oluşturmaktadır. Oluşan bu ÖÖH'lar daha sonra ÇKA yapısına dönüştürülerek ÇKA başlangıç aday havuzuna eklenmektedir.

Daha sonra TA03V01 algoritması başlangıç aday havuzundaki her bir ÇKA'yı kullanarak TA01V01 algoritmasını çalıştırmakta ve çıkan sonucu saklamaktadır. Başlangıç aday havuzundaki tüm ÇKA'lar kullanıldıktan sonra TA01V01 sonucunda oluşan tüm yapılar K' 'nin L' 'li çapraz geçerliliği verileriyle değerlendirmektedir. Son olarak oluşan yapay sinir ağı sonuç olarak verilmektedir.

TA03V01 algoritmasının çalışma sistemi Şekil 4.17'de görülmektedir. Burada kesikli çizgilerle gösterilen yapılar TA03V01 algoritması ile ikinci aşamada eklenen yapılardır. TA03V01 algoritması ilk aşamada k adet başlangıç elemanı ve m adet işlem elemanını Kohonen öz örgütlemeli haritası ile oluşturmaktadır. Daha sonra bu öz örgütlemeli haritalar ikinci katmanı gizli katman olacak şekilde çok katmanlı algılayıcılara dönüştürülmektedir. Bu aşamada oluşan çıktı katmanı boş ÇKA ile TA01V01 algoritmasını uygulanarak bitiş elemanları eklenmektedir.



Şekil 4.17 : TA03V01 algoritmasının çalışma örneği.

Şekil 4.18’de TA03V02 algoritmasının sözde programı verilmiştir. Şekilde belirtildiği üzere algoritma önce öz örgütlemeli haritalar için müsaade edilen en büyük boyutu hesaplamakta daha sonra 1x2 boyutundan müsaade edilen en büyük boyuta kadar belirtilen miktarda öz örgütlemeli harita oluşturmaktadır. Oluşan bu ÖÖH’lar daha sonra ÇKA yapısına dönüştürülerek ÇKA başlangıç aday havuzuna eklenmektedir.

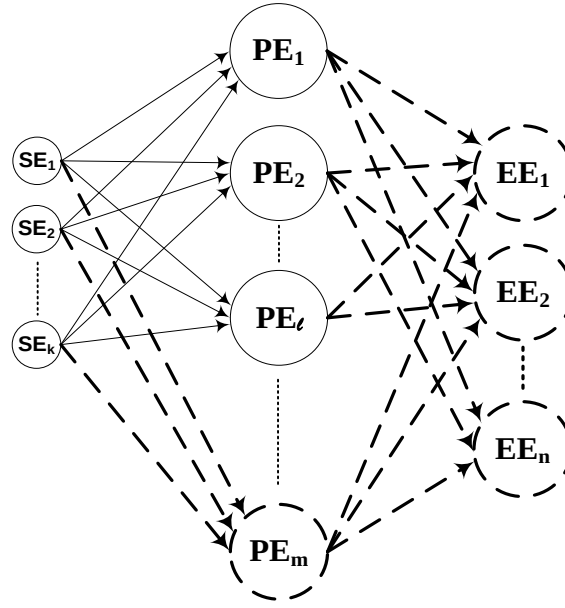
- | |
|---|
| <p>Adım 1: ÖÖH.ların alabileceği en büyük boyutu tespit et.</p> <p>Adım 2: Veri kümesini kullanarak 1x2’den en büyük boyuta kadar farklı boyularda ÖÖH oluştur.</p> <p>Adım 3: Oluşan ÖÖH’leri ÖÖH havuzuna ekle.</p> <p>Adım 4: ÖÖH havuzundaki YSA’ları ÇKA.ya çevir.</p> <p>Adım 5: Oluşan ÇKA’ları ÇKA başlangıç aday havuzuna ekle.</p> <p>Adım 6: ÇKA başlangıç AH’ndaki her bir ÇKA’yı TA02V01 için başlangıç YSA olarak ver.</p> <p>Adım 7: Her bir TA02V01 sonucunda oluşan YSA’yı ÇKA AH.na ekle.</p> <p>Adım 8: Adaylarının uygunluğunu K'’nin L'’li çapraz geçerliği için hesapla</p> <p>Adım 9: En uygun adayı sonuç olarak ver</p> |
|---|

Şekil 4.18 : Temel Algoritma 3 Versiyon 2’nin sözde programı.

Daha sonra algoritma başlangıç aday havuzundaki her bir ÇKA’yı kullanarak TA02V01 algoritmasını çalıştırmakta ve çıkan sonucu saklamaktadır. Başlangıç aday havuzundaki tüm ÇKA’lar kullanıldıktan sonra TA02V01 sonucunda oluşan tüm yapılar K' ’nin L' ’li

çapraz geçerliliği verileriyle değerlendirmektedir. Son olarak oluşan yapay sinir ağı sonuç olarak verilmektedir.

TA03V02 algoritmasının çalışma sistemi Şekil 4.19’da görülmektedir. Burada kesikli çizgilerle gösterilen yapılar TA03V02 algoritması ile ikinci aşamada eklenen yapılardır. TA03V02 algoritması ilk aşamada k adet başlangıç elemanı ve l adet işlem elemanını Kohonen öz örgütlemeli haritası ile oluşturmaktadır. Daha sonra bu öz örgütlemeli haritalar ikinci katmanı gizli katman olacak şekilde çok katmanlı algılayıcılara dönüştürülmektedir. Bu aşamada oluşan çıktı katmanı boş ÇKA ile TA02V01 algoritmasını uygulanarak bitiş elemanları eklenmektedir. TA03V02 algoritması ikinci aşamada TA02V01 algoritmasını kullandığından eğer performans göstergelerini iyileştiriyorsa bitiş elemanı ile birlikte işlem elemanları da eklemektedir.



Şekil 4.19 : TA03V02 algoritmasının çalışma örneği.

4.3 Aday Havuzu Oluşturma Yöntemleri

Daha önceki bölümlerde bahsedilen üç temel algoritma için aday havuzlarının oluşturulması gerekmektedir. Bu aday havuzları içerisinde belirlenmiş uygunluk kriterlerine en çok uyan düğüm ve bağlantılar (PE, EE, ve sinaptik ağırlıklar) seçilecektir. Seçilen bu yapılar mevcut YSA'nın yapısına eklenerek GY algoritması ile eğitilecek ve bir sonraki aşamaya hazır hale geleceklerdir.

Aday havuzlarına yeni bitiş elemanı ve yeni işlem elemanı oluşturmak için dokuzar adet yöntem geliştirilmiş ve bu yöntemler numaralandırılmıştır. Birinci temel algoritma sadece bitiş elemanı eklediği için aday havuzunda sadece bitiş elemanı ekleyen yöntemler kullanılarak adaylar oluşturulmaktadır.

İkinci temel algoritma ise bitiş elemanı ile birlikte gizli katmana yeni bir işlem elemanı eklenmesine müsaade etmektedir. Bunu sağlamak için sadece yeni bitiş elemanı ekleme yöntemleri kullanılarak ön aday havuzu oluşturulmaktadır. Oluşturulan bu ön aday havuzundaki adaylar kullanılarak önceden tanımlanmış belli bir sayıya kadar yeni işlem elemanı eklenmesiyle ayrı ayrı yeni aday yapılar oluşturulmaktadır.

Ön aday havuzundaki adayların da aday havuzuna katılmasıyla aday havuzu son halini alır. Burada hem sadece bitiş elemanı eklenmiş yapılar hem de bu yapıların öncesine farklı işlem elemanları eklenmiş yapılar bulunur. Burada temel amaç problemi anlık bitiş elemanı ya da işlem elemanı eklemeye indirmek yerine mevcut kaynaklar çerçevesinde ikisini birden değerlendirerek global en iyi çözümü aramaktır.

4.3.1 Kopyalama ile tek EE ekleme yöntemi (#EE00)

Çıktı katmanında bulunan ve bu yöntemde kullanılmak üzere seçilen EE'nin transfer fonksiyonu $\phi^{sec}(\cdot)$, gizli katmandaki i 'nci PE ile arasındaki bağlantının ağırlığı w_i^{sec} , yeni eklenen EE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$ ve gizli katmandaki i 'nci PE ile arasındaki bağlantının ağırlığı w_i^{yeni} olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{sec}(\cdot) \quad (4.1)$$

$$w_i^{yeni} = w_i^{sec}, \forall i \quad (4.2)$$

olarak tanımlanabilir.

4.3.2 Rassal tek EE ekleme yöntemi(#EE01)

Aday havuzunun transfer fonksiyonu türü $\phi^{AH}(\cdot)$, çıktı katmanına yeni eklenen EE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$ ve yeni eklenen EE'nin gizli katmandaki i 'nci PE ile arasındaki bağlantının ağırlığı w_i^{yeni} olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{AH}(\cdot) \quad (4.3)$$

$$f(w_i^{yeni}) = \begin{cases} \frac{1}{2} & -1 < w_i^{yeni} < 1 \\ 0 & \text{diğer} \end{cases} \quad (4.4)$$

$$P\{-1 < w_i^{yeni} < 1\} = \int_{-1}^1 f(w_i^{yeni}) dw_i^{yeni} \quad (4.5)$$

olarak tanımlanabilir.

4.3.3 Ayırma ile tek EE ekleme yöntemi (#EE02)

Mevcut durumda çıktı katmanına bağlantısı olan i 'nci PE ile seçilen bir EE arasındaki bağlantının ağırlığı $w_{i1}^{seç.}$, bu ağırlığın yeni değeri $w_{i2}^{seç.}$, yeni eklenen EE arasındaki bağlantının ağırlığı w_i^{yeni} , seçilen EE'nin transfer fonksiyonu $\phi^{seç.}(\cdot)$, yeni eklenen EE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$ ve $p \in (0,1)$ için ağırlık ikililerinin olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{seç.}(\cdot) \quad (4.6)$$

$$g(w_{i2}^{seç.}, w_i^{yeni}) = \begin{cases} p & w_{i2}^{seç.} = w_{i1}^{seç.} \wedge w_i^{yeni} = c, \forall i \\ 1-p & w_{i2}^{seç.} = c \wedge w_i^{yeni} = w_{i1}^{seç.} \end{cases} \quad (4.7)$$

$$c = 0 + \varepsilon \quad (4.8)$$

olarak tanımlanabilir. Burada C sabit bir sayı, ε ise yeterince küçük bir sayıdır.

4.3.4 Paylaştırma ile tek EE ekleme yöntemi (#EE03)

Mevcut durumda çıktı katmanına bağlantısı olan i 'nci PE ile seçilen bir EE arasındaki bağlantının ağırlığı $w_{i1}^{seç.}$, bu ağırlığın yeni değeri $w_{i2}^{seç.}$, yeni eklenen EE arasındaki bağlantının ağırlığı w_i^{yeni} , seçilen EE'nın transfer fonksiyonu $\phi^{seç.}(\cdot)$, yeni eklenen EE'nın transfer fonksiyonu $\phi^{yeni}(\cdot)$ ve $p, r \in (0,1)$ için ağırlık ikililerinin olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{seç.}(\cdot) \quad (4.9)$$

$$g(w_{i2}^{seç.}, w_i^{yeni}) = \begin{cases} p & w_{i2}^{seç.} = r w_{i1}^{seç.} \wedge w_i^{yeni} = (1-r) w_{i1}^{seç.} \\ 1-p & w_{i2}^{seç.} = (1-r) w_{i1}^{seç.} \wedge w_i^{yeni} = r w_{i1}^{seç.} \end{cases}, \forall i \quad (4.10)$$

olarak tanımlanabilir.

4.3.5 Ters işaretli ağırlıkla EE ekleme yöntemi (#EE04)

Çıktı katmanında bulunan ve bu yöntemde kullanılmak üzere seçilen EE'nın transfer fonksiyonu $\phi^{seç.}(\cdot)$, gizli katmandaki i 'nci PE ile arasındaki bağlantının ağırlığı $w_i^{seç.}$, yeni eklenen EE'nın transfer fonksiyonu $\phi^{yeni}(\cdot)$ ve gizli katmandaki i 'nci PE ile arasındaki bağlantının ağırlığı w_i^{yeni} olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{seç.}(\cdot) \quad (4.11)$$

$$w_i^{yeni} = -w_i^{seç.}, \forall i \quad (4.12)$$

olarak tanımlanabilir.

4.3.6 Toplama ile EE ekleme yöntemi (#EE05)

Mevcut durumda çıktı katmanına bağlantısı olan i 'nci PE ile seçilen bir EE arasındaki bağlantının ağırlığı $w_{i1}^{seç.}$, seçilen diğer bir EE arasındaki bağlantının ağırlığı $w_{i2}^{seç.}$, yeni eklenen EE arasındaki bağlantının ağırlığı w_i^{yeni} , seçilen EE'lerinin transfer fonksiyonları sırasıyla $\phi_1^{seç.}(\cdot), \phi_2^{seç.}(\cdot)$, yeni eklenen EE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$ olmak üzere:

$$\phi^{yeni}(\cdot) = \phi_1^{seç.}(\cdot) \quad (4.13)$$

$$w_i^{yeni} = w_{i1}^{seç.} + w_{i2}^{seç.}, \forall i \quad (4.14)$$

olarak tanımlanabilir.

4.3.7 Birleştirme ile EE ekleme yöntemi (#EE06)

Mevcut durumda çıktı katmanına bağlantısı olan i 'nci PE ile seçilen bir EE arasındaki bağlantının ağırlığı $w_{i1}^{seç.}$, seçilen diğer bir EE arasındaki bağlantının ağırlığı $w_{i2}^{seç.}$, yeni eklenen EE arasındaki bağlantının ağırlığı w_i^{yeni} , seçilen EE'lerinin transfer fonksiyonları sırasıyla $\phi_1^{seç.}(\cdot), \phi_2^{seç.}(\cdot)$, yeni eklenen EE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$ ve $p \in (0,1)$ için ağırlık olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi^{yeni}(\cdot) = \phi_1^{seç.}(\cdot) \quad (4.15)$$

$$g(w_i^{yeni}) = \begin{cases} p & w_i^{yeni} = w_{i1}^{seç.} \\ 1-p & w_i^{yeni} = w_{i2}^{seç.} \end{cases}, \forall i \quad (4.16)$$

olarak tanımlanabilir.

4.3.8 Ayırma ile iki aday EE ekleme yöntemi (#EE07)

Mevcut durumda çıktı katmanına bağlantısı olan i 'nci PE ile seçilen bir EE arasındaki bağlantının ağırlığı $w_i^{seç.}$, yeni eklenecek EE adayları arasındaki bağlantının ağırlığı w_{i1}^{yeni} ,

yeni eklenecek diğer EE adayı arasındaki bağlantının ağırlığı w_{i2}^{yeni} seçilen EE'nin transfer fonksiyonu $\phi^{sec.}(\cdot)$, yeni eklenen aday EE'lerin transfer fonksiyonları sırasıyla $\phi_1^{yeni}(\cdot), \phi_2^{yeni}(\cdot)$ ve $p \in (0,1)$ için ağırlık olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi_1^{yeni}(\cdot) = \phi_2^{yeni}(\cdot) = \phi^{sec.}(\cdot) \quad (4.17)$$

$$g(w_{i1}^{yeni}, w_{i2}^{yeni}) = \begin{cases} p & w_{i1}^{yeni} = w_i^{sec.} \wedge w_{i2}^{yeni} = c \\ 1-p & w_{i1}^{yeni} = c \wedge w_{i2}^{yeni} = w_i^{sec.}, \forall i \end{cases} \quad (4.18)$$

$$c = 0 + \varepsilon \quad (4.19)$$

olarak tanımlanabilir. Burada c sabit bir sayı, ε ise yeterince küçük bir sayıdır.

4.3.9 Paylaştırma ile iki aday EE ekleme yöntemi (#EE08)

Mevcut durumda çıktı katmanına bağlantısı olan i 'nci PE ile seçilen bir EE arasındaki bağlantının ağırlığı $w_i^{sec.}$, yeni eklenecek EE adayı arasındaki bağlantının ağırlığı w_{i1}^{yeni} , yeni eklenecek diğer EE adayı arasındaki bağlantının ağırlığı w_{i2}^{yeni} seçilen EE'nin transfer fonksiyonu $\phi^{sec.}(\cdot)$, yeni eklenen aday EE'lerin transfer fonksiyonları sırasıyla $\phi_1^{yeni}(\cdot), \phi_2^{yeni}(\cdot)$ ve $p, r \in (0,1)$ için ağırlık olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi_1^{yeni}(\cdot) = \phi_2^{yeni}(\cdot) = \phi^{sec.}(\cdot) \quad (4.20)$$

$$g(w_{i1}^{yeni}, w_{i2}^{yeni}) = \begin{cases} p & w_{i1}^{yeni} = r w_i^{sec.} \wedge w_{i2}^{yeni} = (1-r) w_i^{sec.} \\ 1-p & w_{i1}^{yeni} = (1-r) w_i^{sec.} \wedge w_{i2}^{yeni} = r w_i^{sec.}, \forall i \end{cases} \quad (4.21)$$

olarak tanımlanabilir.

4.3.10 Kopyalama ile tek PE ekleme yöntemi (#PE00)

Gizli katmanında bulunan ve bu yöntemde kullanılmak üzere seçilen PE'nin transfer fonksiyonu $\phi^{sec.}(\cdot)$, bir önceki katmandaki i 'nci eleman (E) ile arasındaki bağlantının ağırlığı $w_i^{sec.}$, bir sonraki katmandaki j 'nci E ile arasındaki bağlantının ağırlığı $w_j^{sec.}$,

yeni eklenen PE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$, bir önceki katmandaki i 'nci E ile arasındaki bağlantının ağırlığı w_i^{yeni} ve bir sonraki katmandaki j 'nci E ile arasındaki bağlantının ağırlığı w_j^{yeni} olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{seç.}(\cdot) \quad (4.22)$$

$$w_i^{yeni} = w_i^{seç.}, \forall i \quad (4.23)$$

$$w_j^{yeni} = w_j^{seç.}, \forall j \quad (4.24)$$

olarak tanımlanabilir.

4.3.11 Rassal tek PE ekleme yöntemi (#PE01)

Aday havuzunun transfer fonksiyonu türü $\phi^{AH}(\cdot)$, gizli katmana yeni eklenen PE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$, yeni eklenen PE'nin bir önceki katmandaki i 'nci eleman (E) ile arasındaki bağlantının ağırlığı w_i^{yeni} , ve bir sonraki katmandaki j 'nci E ile arasındaki bağlantının ağırlığı w_j^{yeni} olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{AH}(\cdot) \quad (4.25)$$

$$f(w_i^{yeni}) = \begin{cases} \frac{1}{2} & -1 < w_i^{yeni} < 1 \\ 0 & \text{diğer} \end{cases} \quad (4.26)$$

$$f(w_j^{yeni}) = \begin{cases} \frac{1}{2} & -1 < w_j^{yeni} < 1 \\ 0 & \text{diğer} \end{cases} \quad (4.27)$$

$$P\{-1 < w_i^{yeni} < 1\} = \int_{-1}^1 f(w_i^{yeni}) dw_i^{yeni} \quad (4.28)$$

$$P\{-1 < w_j^{yeni} < 1\} = \int_{-1}^1 f(w_j^{yeni}) dw_j^{yeni} \quad (4.29)$$

olarak tanımlanabilir.

4.3.12 Ayırma ile tek PE ekleme yöntemi (#PE02)

Mevcut durumda gizli katmana bağlantısı olan bir önceki katmandaki i 'nci E ile seçilen bir PE arasındaki bağlantının ağırlığı $w_{i1}^{seç.}$, bu ağırlığın yeni değeri $w_{i2}^{seç.}$, yeni eklenen PE arasındaki bağlantının ağırlığı w_i^{yeni} , seçilen gizli katmanla bağlantısı olan bir sonraki katmanda bulunan j 'nci E ile seçilen PE'nın arasındaki bağlantının ağırlığı $w_{j1}^{seç.}$, bu ağırlığın yeni değeri $w_{j2}^{seç.}$, yeni eklenen PE arasındaki bağlantının ağırlığı w_j^{yeni} , seçilen PE'nın transfer fonksiyonu $\phi^{seç.}(\cdot)$, yeni eklenen PE'nın transfer fonksiyonu $\phi^{yeni}(\cdot)$ ve $p \in (0,1)$ için ağırlık ikililerinin olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{seç.}(\cdot) \quad (4.30)$$

$$g(w_{i2}^{seç.}, w_i^{yeni}) = \begin{cases} p & w_{i2}^{seç.} = w_{i1}^{seç.} \wedge w_i^{yeni} = c \\ 1-p & w_{i2}^{seç.} = c \wedge w_i^{yeni} = w_{i1}^{seç.} \end{cases}, \forall i \quad (4.31)$$

$$g(w_{j2}^{seç.}, w_j^{yeni}) = \begin{cases} p & w_{j2}^{seç.} = w_{j1}^{seç.} \wedge w_j^{yeni} = c \\ 1-p & w_{j2}^{seç.} = c \wedge w_j^{yeni} = w_{j1}^{seç.} \end{cases}, \forall j \quad (4.32)$$

$$c = 0 + \varepsilon \quad (4.33)$$

olarak tanımlanabilir. Burada c sabit bir sayı, ε ise yeterince küçük bir sayıdır.

4.3.13 Paylaştırma ile tek PE ekleme yöntemi (#PE03)

Mevcut durumda gizli katmanına bağlantısı olan bir önceki katmandaki i 'nci E ile seçilen bir PE arasındaki bağlantının ağırlığı $w_{i1}^{seç.}$, bu ağırlığın yeni değeri $w_{i2}^{seç.}$, yeni eklenen PE arasındaki bağlantının ağırlığı w_i^{yeni} , seçilen gizli katmanla bağlantısı olan bir sonraki katmanda bulunan j 'nci E ile seçilen PE'nın arasındaki bağlantının ağırlığı $w_{j1}^{seç.}$, bu ağırlığın yeni değeri $w_{j2}^{seç.}$, yeni eklenen PE arasındaki bağlantının ağırlığı w_j^{yeni} , seçilen PE'nın transfer fonksiyonu $\phi^{seç.}(\cdot)$, yeni eklenen PE'nın transfer fonksiyonu $\phi^{yeni}(\cdot)$ ve $p \in (0,1)$ için ağırlık ikililerinin olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$\phi^{yeni}(\cdot)$ ve $p, r \in (0,1)$ için ağırlık ikililerinin olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{seç.}(\cdot) \quad (4.34)$$

$$g(w_{i2}^{seç.}, w_i^{yeni}) = \begin{cases} p & w_{i2}^{seç.} = r w_{i1}^{seç.} \wedge w_i^{yeni} = (1-r) w_{i1}^{seç.} \\ 1-p & w_{i2}^{seç.} = (1-r) w_{i1}^{seç.} \wedge w_i^{yeni} = r w_{i1}^{seç.} \end{cases}, \forall i \quad (4.35)$$

$$g(w_{j2}^{seç.}, w_j^{yeni}) = \begin{cases} p & w_{j2}^{seç.} = r w_{j1}^{seç.} \wedge w_j^{yeni} = (1-r) w_{j1}^{seç.} \\ 1-p & w_{j2}^{seç.} = (1-r) w_{j1}^{seç.} \wedge w_j^{yeni} = r w_{j1}^{seç.} \end{cases}, \forall j \quad (4.36)$$

olarak tanımlanabilir.

4.3.14 Ters işaretli ağırlıkla PE ekleme yöntemi (#PE04)

Gizli katmanında bulunan ve bu yöntemde kullanılmak üzere seçilen PE'nin transfer fonksiyonu $\phi^{seç.}(\cdot)$, bir önceki katmandaki i 'nci E ile arasındaki bağlantının ağırlığı $w_i^{seç.}$, bir sonraki katmandaki j 'nci E ile arasındaki bağlantının ağırlığı $w_j^{seç.}$, yeni eklenen EE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$, bir önceki katmandaki i 'nci E ile arasındaki bağlantının ağırlığı w_i^{yeni} ve bir sonraki katmandaki j 'nci E ile arasındaki bağlantının ağırlığı w_j^{yeni} olmak üzere:

$$\phi^{yeni}(\cdot) = \phi^{seç.}(\cdot) \quad (4.37)$$

$$w_i^{yeni} = -w_i^{seç.}, \forall i \quad (4.38)$$

$$w_j^{yeni} = -w_j^{seç.}, \forall j \quad (4.39)$$

olarak tanımlanabilir.

4.3.15 Toplama ile PE ekleme yöntemi (#PE05)

Seçilen gizli katmandan bir önceki katmanda bulunan i 'nci E ile seçilen bir PE arasındaki bağlantının ağırlığı $w_{i1}^{seç.}$, seçilen diğer bir PE arasındaki bağlantının ağırlığı $w_{i2}^{seç.}$, yeni eklenen PE arasındaki bağlantının ağırlığı w_i^{yeni} olsun. Seçilen gizli katmandan bir sonraki katmanda bulunan j 'nci E ile seçilen ilk PE arasındaki bağlantının ağırlığı $w_{j1}^{seç.}$, seçilen ikinci PE arasındaki bağlantının ağırlığı $w_{j2}^{seç.}$, yeni eklenen PE arasındaki bağlantının ağırlığı w_j^{yeni} olsun. Seçilen PE'lerinin transfer fonksiyonları sırasıyla $\phi_1^{seç.}(\cdot), \phi_2^{seç.}(\cdot)$, yeni eklenen PE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$ olmak üzere:

$$\phi^{yeni}(\cdot) = \phi_1^{seç.}(\cdot) \quad (4.40)$$

$$w_i^{yeni} = w_{i1}^{seç.} + w_{i2}^{seç.}, \forall i \quad (4.41)$$

$$w_j^{yeni} = w_{j1}^{seç.} + w_{j2}^{seç.}, \forall j \quad (4.42)$$

olarak tanımlanabilir.

4.3.16 Birleştirme ile PE ekleme yöntemi (#PE06)

Seçilen gizli katmandan bir önceki katmanda bulunan i 'nci E ile seçilen bir PE arasındaki bağlantının ağırlığı $w_{i1}^{seç.}$, seçilen diğer bir PE arasındaki bağlantının ağırlığı $w_{i2}^{seç.}$, yeni eklenen PE arasındaki bağlantının ağırlığı w_i^{yeni} olsun. Seçilen gizli katmandan bir sonraki katmanda bulunan j 'nci E ile seçilen ilk PE arasındaki bağlantının ağırlığı $w_{j1}^{seç.}$, seçilen ikinci PE arasındaki bağlantının ağırlığı $w_{j2}^{seç.}$, yeni eklenen PE arasındaki bağlantının ağırlığı w_j^{yeni} olsun. Seçilen PE'lerinin transfer fonksiyonları sırasıyla $\phi_1^{seç.}(\cdot), \phi_2^{seç.}(\cdot)$, yeni eklenen PE'nin transfer fonksiyonu $\phi^{yeni}(\cdot)$, ve $p \in (0,1)$ için ağırlık olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi^{yeni}(\cdot) = \phi_1^{sec.}(\cdot) \quad (4.43)$$

$$g(w_i^{yeni}) = \begin{cases} p & w_i^{yeni} = w_{i1}^{sec.} \\ 1-p & w_i^{yeni} = w_{i2}^{sec.} \end{cases}, \forall i \quad (4.44)$$

$$g(w_j^{yeni}) = \begin{cases} p & w_j^{yeni} = w_{j1}^{sec.} \\ 1-p & w_j^{yeni} = w_{j2}^{sec.} \end{cases}, \forall j \quad (4.45)$$

olarak tanımlanabilir.

4.3.17 Ayırma ile iki aday PE ekleme yöntemi (#PE07)

Seçilen gizli katmandan bir önceki katmanda bulunan i 'nci E ile seçilen PE arasındaki bağlantının ağırlığı $w_i^{sec.}$, yeni eklenen ilk PE arasındaki bağlantının ağırlığı w_{i1}^{yeni} , yeni eklenen ikinci PE arasındaki bağlantının ağırlığı w_{i2}^{yeni} olsun. Seçilen gizli katmandan bir sonraki katmanda bulunan j 'nci E ile seçilen PE arasındaki bağlantının ağırlığı $w_j^{sec.}$, yeni eklenen ilk PE arasındaki bağlantının ağırlığı w_{j1}^{yeni} , yeni eklenen ikinci PE arasındaki bağlantının ağırlığı w_{j2}^{yeni} olsun. Seçilen PE'nin transfer fonksiyonu $\phi^{sec.}(\cdot)$, yeni eklenen aday EE'lerin transfer fonksiyonları sırasıyla $\phi_1^{yeni}(\cdot), \phi_2^{yeni}(\cdot)$ ve $p \in (0,1)$ için ağırlık olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi_1^{yeni}(\cdot) = \phi_2^{yeni}(\cdot) = \phi^{sec.}(\cdot) \quad (4.46)$$

$$g(w_{i1}^{yeni}, w_{i2}^{yeni}) = \begin{cases} p & w_{i1}^{yeni} = w_i^{sec.} \wedge w_{i2}^{yeni} = c \\ 1-p & w_{i1}^{yeni} = c \wedge w_{i2}^{yeni} = w_i^{sec.} \end{cases}, \forall i \quad (4.47)$$

$$g(w_{j1}^{yeni}, w_{j2}^{yeni}) = \begin{cases} p & w_{j1}^{yeni} = w_j^{sec.} \wedge w_{j2}^{yeni} = c \\ 1-p & w_{j1}^{yeni} = c \wedge w_{j2}^{yeni} = w_j^{sec.} \end{cases}, \forall j \quad (4.48)$$

$$c = 0 + \varepsilon \quad (4.49)$$

olarak tanımlanabilir. Burada c sabit bir sayı, ε ise yeterince küçük bir sayıdır.

4.3.18 Paylaştırma ile iki aday PE ekleme yöntemi (#PE08)

Seçilen gizli katmandan bir önceki katmanda bulunan i 'nci E ile seçilen PE arasındaki bağlantının ağırlığı $w_i^{seç.}$, yeni eklenen ilk PE arasındaki bağlantının ağırlığı w_{i1}^{yeni} , yeni eklenen ikinci PE arasındaki bağlantının ağırlığı w_{i2}^{yeni} olsun. Seçilen gizli katmandan bir sonraki katmanda bulunan j 'nci E ile seçilen PE arasındaki bağlantının ağırlığı $w_j^{seç.}$, yeni eklenen ilk PE arasındaki bağlantının ağırlığı w_{j1}^{yeni} , yeni eklenen ikinci PE arasındaki bağlantının ağırlığı w_{j2}^{yeni} olsun. Seçilen PE'nin transfer fonksiyonu $\phi^{seç.}(\cdot)$, yeni eklenen aday EE'lerin transfer fonksiyonları sırasıyla $\phi_1^{yeni}(\cdot), \phi_2^{yeni}(\cdot)$ ve $p, r \in (0,1)$ için ağırlık olasılık kütle fonksiyonu $g(\cdot)$ olmak üzere:

$$\phi_1^{yeni}(\cdot) = \phi_2^{yeni}(\cdot) = \phi^{seç.}(\cdot) \quad (4.50)$$

$$g(w_{i1}^{yeni}, w_{i2}^{yeni}) = \begin{cases} p & w_{i1}^{yeni} = r w_i^{seç.} \wedge w_{i2}^{yeni} = (1-r) w_i^{seç.} \\ 1-p & w_{i1}^{yeni} = (1-r) w_i^{seç.} \wedge w_{i2}^{yeni} = r w_i^{seç.} \end{cases}, \forall i \quad (4.51)$$

$$g(w_{j1}^{yeni}, w_{j2}^{yeni}) = \begin{cases} p & w_{j1}^{yeni} = r w_j^{seç.} \wedge w_{j2}^{yeni} = (1-r) w_j^{seç.} \\ 1-p & w_{j1}^{yeni} = (1-r) w_j^{seç.} \wedge w_{j2}^{yeni} = r w_j^{seç.} \end{cases}, \forall j \quad (4.52)$$

olarak tanımlanabilir.

4.4 Aday Seçimi

Bu çalışma kapsamındaki tüm algoritmalarda aksi belirtilmedikçe aday seçimi için K 'nın L 'li çapraz geçerliliği kullanılmıştır. Mevcut yazında K -kat çapraz geçerlilik (K -fold cross validation) kullanılmaktadır. K -kat çapraz geçerlilik K 'nın L 'li çapraz geçerliliğinin $L=1$ olduğu özel bir durumdur (Guresen ve Kayakutlu, 2013).

K 'nın L 'li çapraz geçerliliği eldeki eğitim için kullanılacak gözlemlerin (X) ayrılış (partition) kümesidir. X 'in ayrılış kümesi $X_j, j=0, \dots, K-1$ için aşağıdaki şartlar geçerlidir:

$$X_i \neq \emptyset, \forall i \quad (4.53)$$

$$\bigcup_i X_i = X \quad (4.54)$$

$$X_i \cap X_j = \emptyset, \forall i, \forall j, i \neq j \quad (4.55)$$

Yukarıdaki ayrılış küme şartlarına ek olarak K 'nın L 'li çapraz geçerliliğinde aşağıdaki eleman sayılarının eşit veya yakın olma şartı da aranmaktadır:

$${}_s(X_i) \approx {}_s(X_j), \forall i, \forall j, i \neq j \quad (4.56)$$

K 'nın L 'li çapraz geçerliliği kısaca tüm eğitim verisinin rassal olarak K adet ayrık alt kümeye eleman sayıları yaklaşık olarak eşit olacak şekilde bölünmesidir.

4.4.1 K -kat çapraz geçerlilik

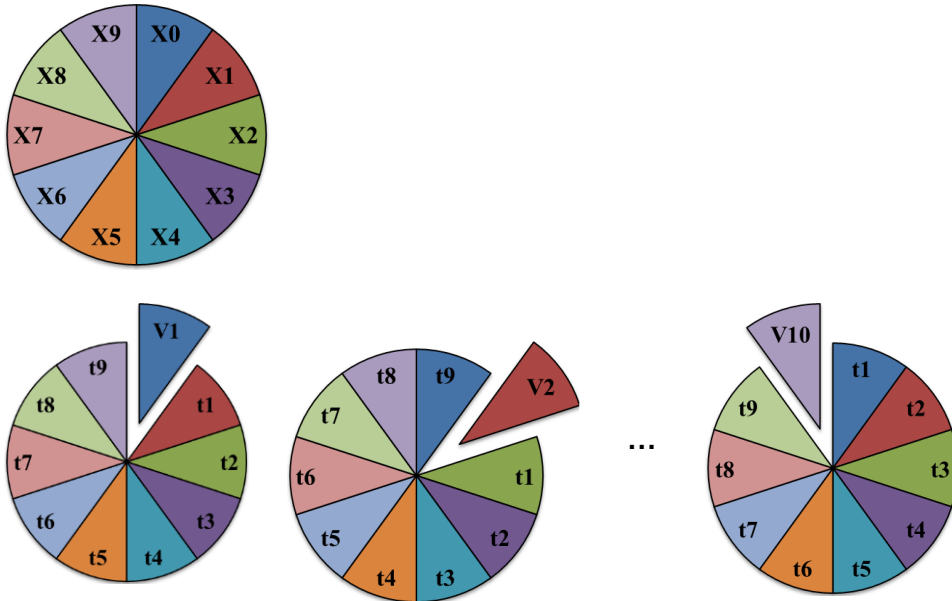
K -kat çapraz geçerlilikte tüm eğitim verisinin (X) rassal olarak K adet ayrık alt kümeye eleman sayıları yaklaşık olarak eşit olacak şekilde bölünmesiyle $X_j, j=0, \dots, K-1$ çapraz geçerlilik bölünmüş kümesi elde edilir. Eğitim kümelerini oluşturmak için ise K adet alt kümeden biri geçerleme için ayrılır. Kalan $K-1$ adet küme birleştirilerek eğitim kümesi oluşturulur. Bu durumda oluşan K adet eğitim kümesinden V geçerleme kümesini, T eğitim kümesini ve $A_i; i$ 'nci $(i=1, \dots, K)$ çapraz geçerlilik eğitim alternatifini göstermek üzere aşağıdaki gibi ifade edebiliriz:

$$A_i = \left\{ V_i = X_{i-1}, T_i = \bigcup_j (X_j \mid j \neq i-1) \right\} \quad (4.57)$$

Yukarıdaki ifadeyi açık olarak şu şekilde ifade edebiliriz:

$$\begin{aligned}
A_1 &= \{V_1 = X_0, T_1 = X_1 \cup X_2 \cup X_3 \cup \dots \cup X_{K-2} \cup X_{K-1}\} \\
A_2 &= \{V_2 = X_1, T_2 = X_2 \cup X_3 \cup \dots \cup X_{K-2} \cup X_{K-1} \cup X_0\} \\
A_3 &= \{V_3 = X_2, T_3 = X_3 \cup \dots \cup X_{K-2} \cup X_{K-1} \cup X_0 \cup X_1\} \\
&\vdots \\
A_K &= \{V_K = X_{K-1}, T_K = X_1 \cup X_2 \cup X_3 \cup \dots \cup X_{K-3} \cup X_{K-2}\}
\end{aligned} \tag{4.58}$$

K -kat çapraz geçerlilikte örnek olarak $K=10$ olduğunda $X_j, j=0, \dots, 9$ ayrık alt kümeler oluşturulur. Bu X_j alt kümelerinden bir tanesi V_i olarak geçerleme için kalan 9 tanesi $(t_j \mid j \neq i)$ birleştirilerek eğitimde kullanılır. Eğitim kümelerinin bileşimi bir defa eğitimde kullanıldıktan sonra geçerleme için kullanılan alt kümeden bir sonraki alt küme yeni geçerleme kümesi olarak belirlenir. Kalan 9 küme tekrar birleştirilerek yeni eğitim kümesi elde edilir. Her seferinde geçerlemede kullanılacak küme bir sonraki olacak şekilde kullanılır ve her alt küme bir defa geçerlemede kullanılmak üzere toplam 10 adet eğitim veri kümesi alternatifi oluşturulur. Her adımda farklı bir küme kullanılarak yapay sinir ağlarının ezberleme yapmasını engellenmeye çalışılır. Bu yapı Şekil 4.20’de görülmektedir.



Şekil 4.20 : $K=10$ için K -kat çapraz geçerlilik kümeleri.

4.4.2 K 'nın L 'li çapraz geçerliliği

K 'nın L 'li çapraz geçerliliği de tüm eğitim verisinin (X) rassal olarak K adet ayrık alt kümeye eleman sayıları yaklaşık olarak eşit olacak şekilde bölünmesiyle $X_j, j=0, \dots, K-1$ çapraz geçerlilik ayrılış kümesi elde edilir. K -kat çapraz geçerlilikten temel farkı geçerleme için bir adet küme yerine ard arda gelen L adet kümenin geçerleme için ayrılmasıdır. Dikkat edilmesi gereken nokta burada ayrılan L adet kümenin birleştirilmeden, ayrı ayrı çapraz geçerleme için kullanılmasıdır. Kalan $K-L$ adet küme ise birleştirilerek eğitim kümesi oluşturulur. Bu durumda oluşan K adet eğitim kümesinden V geçerleme kümesini, T eğitim kümesini, A eğitim alternatifleri ve $A_i; i$ 'nci $(i=1, \dots, K)$ çapraz geçerlilik eğitim alternatifini göstermek üzere aşağıdaki gibi ifade edebiliriz:

$$A = \bigcup_i A_i \quad (4.59)$$

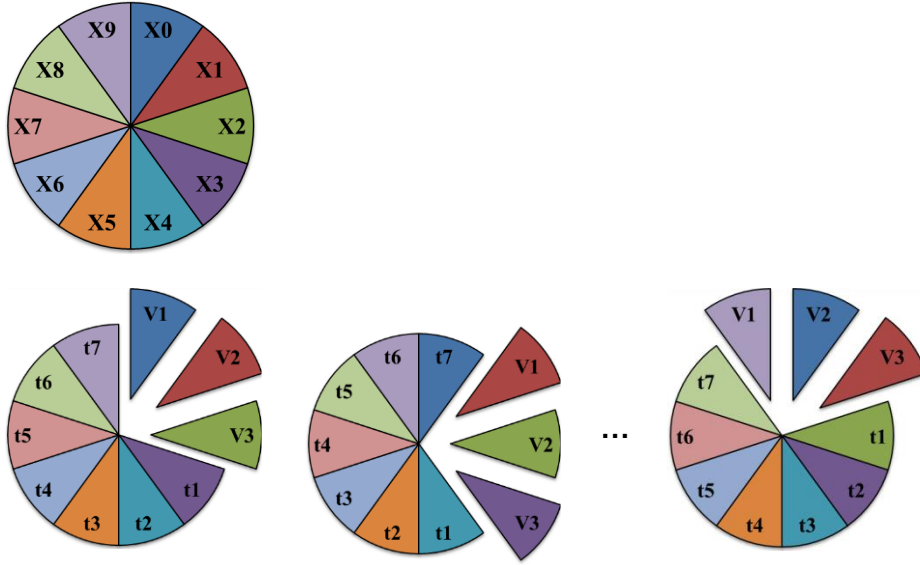
$$A_i = \left\{ T_i = \bigcup_j \left(X_j \mid X_j \notin \bigcup_{m=1}^L V_m \right), V_1 = X_{(i-1) \bmod K}, V_2 = X_{(i) \bmod K}, \dots, V_L = X_{(i+L-2) \bmod K} \right\} \quad (4.60)$$

Yukarıdaki ifadeyi açık olarak şu şekilde ifade edebiliriz:

$$\begin{aligned} A_1 &= \left\{ T_1 = \bigcup_j \left(X_j \mid X_j \notin \bigcup_{m=1}^L V_m \right), V_1 = X_{(0) \bmod K=0}, V_2 = X_{(1) \bmod K}, \dots, V_L = X_{(L-1) \bmod K} \right\} \\ A_2 &= \left\{ T_2 = \bigcup_j \left(X_j \mid X_j \notin \bigcup_{m=1}^L V_m \right), V_1 = X_{(1) \bmod K}, V_2 = X_{(2) \bmod K}, \dots, V_L = X_{(L) \bmod K} \right\} \\ &\vdots \\ A_K &= \left\{ T_K = \bigcup_j \left(X_j \mid X_j \notin \bigcup_{m=1}^L V_m \right), V_1 = X_{(K-1) \bmod K}, V_2 = X_{(K) \bmod K=0}, \dots, V_L = X_{(K+L-2) \bmod K} \right\} \end{aligned} \quad (4.61)$$

Yukarıdaki matematiksel ifadeleri Şekil 4.21'de görsel olarak ifade edebilir. K 'nın L 'li çapraz geçerliliği için Şekil 4.21'de görüldüğü gibi $K=10$ ve $L=3$ olarak alalım.

K -kat çapraz geçerliliğe benzer şekilde eğitim kümesi $X_j, j=0,...,9$ ayrık alt kümelerle bölünür. Bu kümelerden ardışık üç tanesi sırasıyla V_1, V_2 ve V_3 her biri ayrı ayrı geçerleme kümesi olacak şekilde alır ve kalan 7 ($K-L$) adet küme ile eğitimi yapılır. Eğitim kümelerinin bileşimi bir defa kullanıldıktan sonra geçerleme için kullanılan ilk alt kümeden bir sonraki alt küme olarak belirlenir. Ardından gelen iki alt veri kümesi de ayrı ayrı geçerleme kümesi olmak üzere toplam üç geçerleme kümesi belirlenir.



Şekil 4.21 : $K=10$ ve $L=3$ için K 'nın L 'li çapraz geçerlilik kümeleri.

Ayrılan L küme ile aday seçiminde ise her bir adayın her bir geçerleme kümesi için ayrı ayrı performans kriteri (ortalama hata kareleri – OHK, ikili fonksiyon doğru yüzdesi, doğru sınıflama yüzdesi) hesaplanır. Öncelikle tüm geçerleme kümelerinde iyileşme sağlayan aday seçilir. Eğer tüm kümelerde iyileşme sağlanamıyorsa en fazla kümede iyileşme sağlayan aday seçilir. Eğer en fazla kümede iyileşme sağlayan birden fazla aday varsa bu adaylardan OHK'ni en fazla düşüren aday seçilir. Daha önce bahsedildiği gibi K 'nın L 'li çapraz geçerliliği en uygun adayın seçilmesinde kullanılır.

Aday seçim işleminde K 'nın L 'li çapraz geçerliliği, ayrılan L adet kümenin her birinin tüm veri kümesini temsil edebilecek yeterlilikte birer örneklem olduğu,

dolayısıyla her geçerleme kümesinde ayrı ayrı iyileştirme sağlayan işlemin yapay sinir ağının genelleştirme özelliğini destekleyen bir ilerleme olacağı varsayımına dayanır.

K 'nın L 'li çapraz geçerliliği geri yayılım algoritmasında K -kat çapraz geçerlemede olduğu gibi kullanılır. Ancak çapraz geçerlilik hatası hesaplanırken tüm geçerleme kümelerindeki toplam OHK hesaplanır.

K -kat çapraz geçerlilik kullanan yapıcı algoritmalar, örneğin N2C2S algoritması (Setiono, 2001), mevcut ağa yeni bir eleman ya da bağlantı ekleyeceklerinde çapraz geçerlilik kümesine bakarak önlerindeki iki seçeneği değerlendirir;

1. Yeni bağlantıyı ekleme,
2. Yeni işlem elemanı ekleme

Bu algoritmalar bir aday havuzu içerisinden seçme yapamamaktadırlar, her seferinde yeni bir ekleme yapmaya çalışmaktadırlar. Bu tip algoritmalar eğer K -kat çapraz geçerlilik kullanarak bir aday havuzu içerisinden seçme yapmaya çalışırlarsa bunu yapmanın en olası yolu çapraz geçerlilik kümesinde en iyi iyileşmeyi sağlayanı almaktır. Bu durumun da tek bir alt kümenin özelliklerini ezberlemeye yönelik bir eğitim süreci sergilemesi muhtemeldir. K 'nın L 'li çapraz geçerliliği bu tip durumlarda YSA'nın yapısına genelleme yeteneğini arttırması muhtemel eklemeler yapmak için kullanılmaktadır.

N2C2S algoritması aday havuzu kullanmadığından oluşan bir ağ aynı sayıda ve aynı bağlantılara sahip ama farklı başlangıç ağırlıkları kullanılarak eğitilmiş bir yapay sinir ağı ile karşılaştırmakta ve eğer yeni olan ağ daha iyi performans sergilemişse onu kullanıp diğerini silmektedir. Eğer bu durumda K 'nın L 'li çapraz geçerliliği kullanılsaydı elde edilen ağ silinmeden önce daha farklı ekleme alternatifleri de değerlendirilmiş olacaktı. Böylece çözüm uzayında daha ayrıntılı ve potansiyel iyi çözümlere doğru aramalar yapılması sağlanabilecektir.

4.5 Algoritmaların Durma Koşulları

Durma koşulları temel algoritmalar ve geri yayılım algoritması için ayrı ayrı tanımlanmıştır. Geri yayılım algoritması için durma koşulları aşağıdaki gibidir:

- İterasyon sayısı $\geq$ İzin verilen maksimum iterasyon sayısı,
- Aynı hata seviyesinde kalınan iterasyon sayısı $\geq$ Sabit hata seviyesinde izin verilen maksimum iterasyon sayısı,
- Ulaşılan hata seviyesi $\leq$ İstenilen hata seviyesi,
- Ulaşılan sınıflama oranı $\geq$ İstenilen sınıflama oranı

Geri yayılım algoritmasının durmasında yukarıdaki koşulların sağlanıp sağlanmadığını kontrol için aşağıda verilen parametreler kullanılmaktadır.

- Maksimum iterasyon sayısı
- İstenen hata seviyesi / sınıflama seviyesi
- Hatanın değişmeden kalmasına müsaade edilen maksimum iterasyon sayısı
- Hata değişikliği duyarlılığı

Yukarıdaki parametrelerden hata değişiklik duyarlılığı mevcut hatanın seviyesinin değişmediği kabul edilecek değeri göstermektedir. Başka bir deyişle ardışık iki iterasyon arasında belirtilen duyarlılıktan daha az bir fark olursa bu iki iterasyon arasında hatanın değişmediği kabul edilmektedir.

5. DİNAMİK YAPAY SİNİR AĞI YAZILIMI

Önceki bölümlerde önerilen temel algoritmaların, aday havuzlarının, aday oluşturma yöntemlerinin ve çapraz geçerlilik yöntemlerinin uygulanabilmesi için bir yazılıma ihtiyaç duyulmaktadır. Bu bölümde yazılım ihtiyacı, bu ihtiyaçların sebepleri ve yeni önerilen algoritma ve yöntemlerin nasıl uygulandığı açıklanacaktır.

5.1 DYSA İçin Yazılım İhtiyacının Tespiti

Bu çalışma kapsamında yapılmak istenen algoritmaların geliştirilmesine elverişli mevcut Java tabanlı yapay sinir ağı yazılımları incelenmiş ve bunların bu tez kapsamında tekrar kullanılıp kullanılamayacağı değerlendirilmiştir.

Java programlama dilinin seçilmesinde

- Açık kaynak kodlu olması,
- Birçok hazır modülün açık kaynak kodlu olarak sunulması
- Programların her platformda çalışabilmesi,
- Açık kaynak kodu sayesinde herkesin gelişimine katkı sağlaması
- Kurumsal desteğin varlığı (Sun-Oracle)

önemli rol oynamıştır.

5.1.1 Java NN

Bu yazılımın kodları incelendiğinde yazılımın mimarisinin çok karmaşık olduğu görülmektedir. Temel olarak yazılımda olması beklenen element (neuron) - katman (layer) - ağ (network) hiyerarşisine rastlanmamaktadır.

Kullanım lisansı bu kodun ayrı bir klasörde ve içeriğinde hiçbir değişiklik yapılmadan kullanılmasına izin vermekte, ve bu kullanımın belirtilmesini istemektedir.

5.1.2 Heaton Research

Heaton Araştırma grubunun geliştirdiği Java tabanlı yapay sinir ağı programı genetik algoritmalar ve benzetim tavlama ile yapay sinir ağına parametre optimizasyonu yapmaktadır. Kodlar incelendiğinde yapay sinir ağı katmanlarını tek tek elementlerden oluşturmak yerine tek bir transfer fonksiyonu tanımlayarak oluşturmaktadır. Bu da aynı katmanda farklı transfer fonksiyonuna sahip neuronların (element) olmasını engellemektedir. Böylece işlemler matrisler üzerinden yapılabilmekte ve daha basit bir yapıya sahip olabilmektedir.

Bu program geliştirilecek algorithma kullanılırsa farklı transfer fonksiyonuna sahip katmanlar oluşturulamayacak bu da ileride karşılaştırma yapmak istediğimiz algorithmalarda sorun yaratacaktır. Diğer bir nokta, matris işlemleri ile sonuçları elde ediyor olması programlama esnekliğini ortadan kaldırmakta (bazı ysa türlerini kodlamayı zorlaştıracak) ve önerilen yeni algorithmaların uygulanması sırasında ciddi sorunlar çıkarması kaçınılmazdır.

5.1.3 Neurograph (Neuroph)

Neurograph adlı program aslında NetBeans editörüne ek olarak yazılmış bir yapay sinir ağı editörüdür. Kaynak kodları verilmemiştir; farklı çalışmalarda tekrar kullanımı yasaktır. Genel olarak program parametre optimizasyonu yapmamakta ve daha çok basit bir yapay sinir ağının çalışmasını anlatmada kullanılabilecek bir örnek seviyesindedir.

Bu çalışmanın algoritma geliştirme aşamasında sadece programın temel sınıf diyagramı grafik olarak açıklanmıştır. Tezin uygulama aşamasının bitimine yakın, yazılımın adı “neuroph” olarak değiştirilmiş kullanım lisansı genelleştirilmiş. Daha çok tek tek sinir hücresi bazında göstergeleri güçlüdür. Bu kapsamda ysa yapısı ve temel kavramlar anlatılırken derste uygulamalarda kullanılacak bir yazılımdır. Hiyerarşik yazılım yapısı anlamayı kolaylaştırmaktadır ama genel uygulama sihirbazı ve arayüz eksiklikleri veri madenciliği uygulamalarında kullanımını zorlaştırmaktadır.

5.1.4 Mevcut yazılımların kullanımının değerlendirilmesi

Yukarıda incelenen iki yazılımın kullanımında (JavaNN ve Neurograph) lisanslama sorunu nedeniyle bu tez kapsamında kullanılmamıştır. JavaNN ve Heaton Research

yazılımlarında ise element - katman - ağ hiyerarşisine uyulmamış ve bu durum yapay sinir ağının oluşturulma şekli ile ilgili detaylı bir araştırma öneren bu tez açısından ciddi sorunlar yaratacağı değerlendirilmiştir. Bunlardan en risklisi “spagetti kod” adı verilen hangi kod ile neyin oluşturulduğunun çok zor gözlemlendiği karmaşık bir kodlamanın yapılmasının kaçınılmaz olmasıdır.

Tüm bu durumlar göz önüne alınarak bu tez çalışması için sıfırdan yeni bir yazılımın (NeuroBee) Java tabanlı olarak geliştirilmesi kararlaştırılmıştır. NeuroBee yazılımının çok katmanlı algılayıcı (MLP) ve özörgütlemeli harita (SOM) modülleri ve basit bir veri dosya seçici ile kayıt ekranı yazımında ulaşılan kodlama satırı (açıklamalar hariç) yaklaşık 2500 satırdır. Yukarıda adı geçen yazılımların kimisi bu satır sayısından çok daha kısa ya da genetik ve benzetim tavlama algoritmaları ve kullanıcı arayüzleri ile beraber ancak bu civardadır. Bu da yukarıda incelenen yazılımların kısa yollardan yazıldığı ve yapısal esnekliklerinin azlığı ve tam olarak nesne tabanlı yaklaşımla yazılmadıkları yönündeki şüpheleri arttırmaktadır.

5.1.5 Java editör seçimi

En yaygın Java editörleri Eclipse ve NetBeans’dir. İkisi de yazarken otomatik tamamlama, yazarken derleme ve hata bildirimi özelliklerine sahiptir. NetBeans’in öne çıkan özelliği grafik tasarım için görsel eklentisinin ücretsiz olması ve standart paket ile birlikte bulunmasıdır. Bu çalışma başladığında Eclipse için bu eklentiler ücretliydi. Burada dikkat edilmesi gereken nokta bu eklentiler olmadan da pencere ve görsel özellikler hazırlanabilmektedir. Bu eklentiler görsel tasarımdaki bir çok ayrıntı kodlamayı ortadan kaldırmakta ve arayüz yazımını kolaylaştırmaktadırlar.

Grafik tasarım aracı olması nedeniyle projenin yazımına NetBeans editörü ile başlanmıştır ve MLP, SOM, girdi-çıkı modüllerinin tamamı bu editör ile yazılmıştır. Ancak kodun büyüklüğünün artması ile bu editörde sıkıntılar yaşanmaya başlamıştır. Bunun en önemlilerinden birisi yazılan kodların (sınıf, metod, değişken vs) NetBeans de sınıflandırılarak gösterilmemesidir. Eclipse’te ise tüm sınıflar, o sınıflara ait metodlar sıra ile hiyerarşik bir şekilde gözükmemektedir. Bu da program yapısının okunmasını kolaylaştırmaktadır.

Eclipse editörünün ücretsiz görsel tasarım eklentisi sunmaya başlamasıyla tüm yazılım Eclipse editörüne aktarılmış ve uygulama yazılımı bu editörde geliştirilmiştir.

5.2 Yazılımın Mimari Yapısı

NeuroBee iki temel aşamada geliştirilmiştir. Bunlardan ilki bilinen yapay sinir ağlarını özellikle bu çalışmada kullanılacak olan ÇKA ve Kohonen yapısını içerecek kısımların kodlanmasıdır. Bu işlem sırasında ikinci aşamadaki algoritma yapılarını destekleyebilecek esneklik ve hiyerarşik yapı bu çalışmanın ilk kısımlarında yapılan tanımlamalar dikkate alınarak sağlanmıştır.

İkinci temel aşama ise önerilen algoritmaların kodlanması, sonuçların görülmesi ve algoritmaların güncellenmesidir. Bu iki temel aşama doğal olarak yazılımın mimari yapısına da yansımış böylece YSA kısmı ve DYSA kısmı ayrı ayrı geliştirilmiştir.

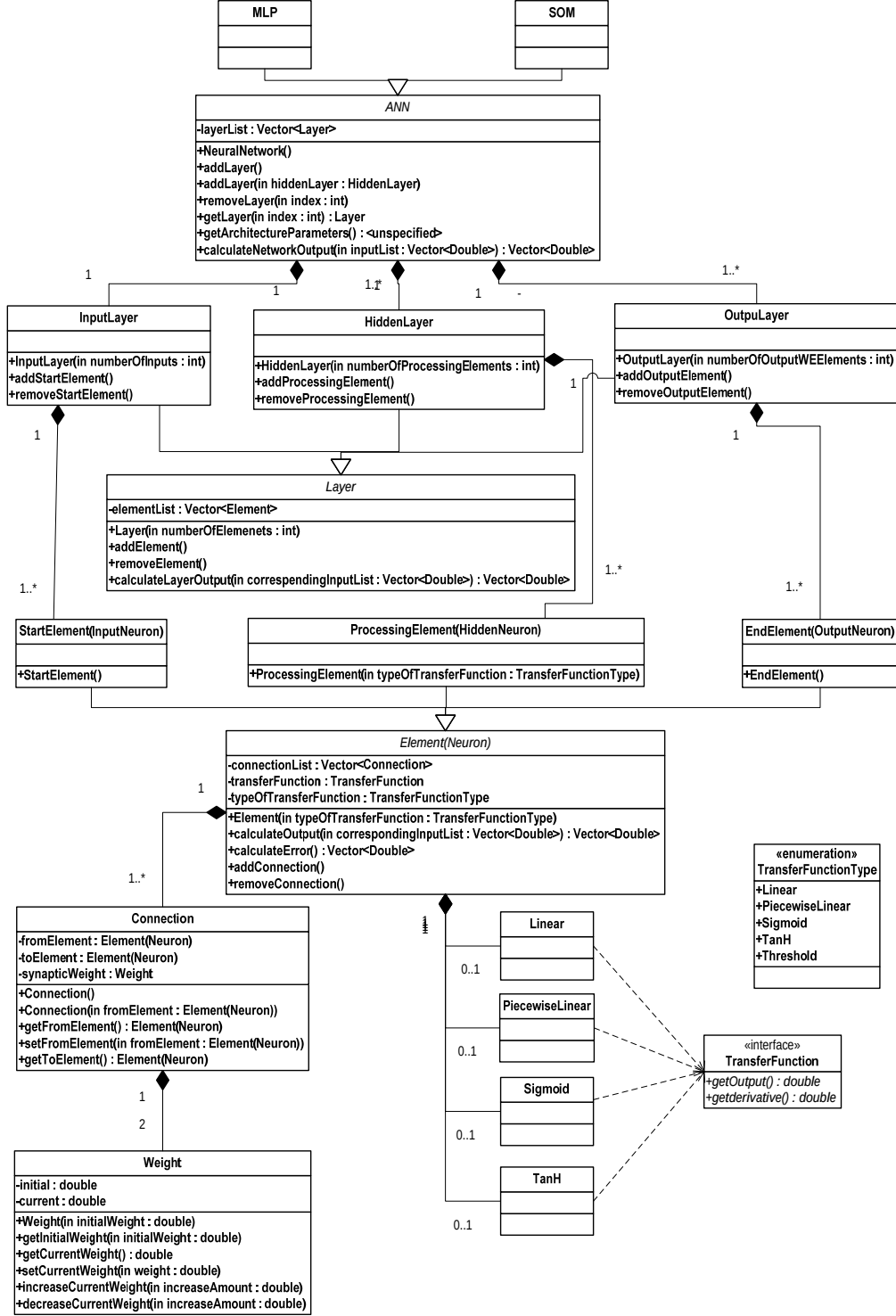
Yazılımın YSA kısmı geliştirilirken daha çok bilinen algoritmaların ve örneklerin sonuçları ile yazılımın sonuçları karşılaştırılarak yazılımın doğru çalıştığı teyit edilmiştir. İkinci kısım olan DYSA kısmı geliştirilirken yazılımın ürettiği sonuçların doğruluğu yanında önerilen algoritma ve yöntemlerin bu çalışma amacına uygunluğu da değerlendirilmiştir. Bu aşamada önerilen algoritmalar ve bu algoritmaları uygulayabilmek için gerekli yöntemler zaman içerisinde birçok değişikliğe uğramış ve son hallerini almışlardır.

5.3 YSA Yapısı

Yapay sinir ağlarının teorik yapısı incelenip, mevcut yazılımlardan neurograph'ın temel sınıf çizelgesi göz önüne alınarak yazılacak programın sınıfları oluşturulmuştur. Bu sınıflama Şekil 5.1'de görülmektedir.

Şekil 5.1'de en alttan başlayarak en yukarı doğru gidilerek, en ufak bileşenlerden yapay sinir ağına doğru hiyerarşik olarak YSA yapısının oluşturuluşu verilmiştir. oluruz. Burada görülen yapıda bir yapının (neuronun) nasıl oluşturulacağı, bünyesinde neler bulunduracağı ve her bir sınıfın hangi metodları kullanacağı görülmektedir. Şeklin en

altında yer alan ve hiçbir bağlantısı bulunmayan sınıflar ise bu yazılımda kullanılan hiyerarşik yapıda direkt yer alamayan sınıflardır.



Şekil 5.1 : Neurobee yazılımı temel sınıfları.

5.4 DYSA Yapısı

Yazılımın DYSA paketinde TA01V01, TA02V01, TA03V01 ve TA03V02 algoritmaları bulunmaktadır. Ayrıca YSA sınıfının üzerine bu çalışma kapsamında DYSA'nın aşamalarını kaydeden DYSA temel sınıfı oluşturulmuştur.

Yazılımın bu aşamasında aday havuzları, aday havuzları hazırlama yöntemleri ve K 'nın L 'li çapraz geçerliliği uygulamaları yazılmıştır.

5.5 Yazılımın Test Prosedürü

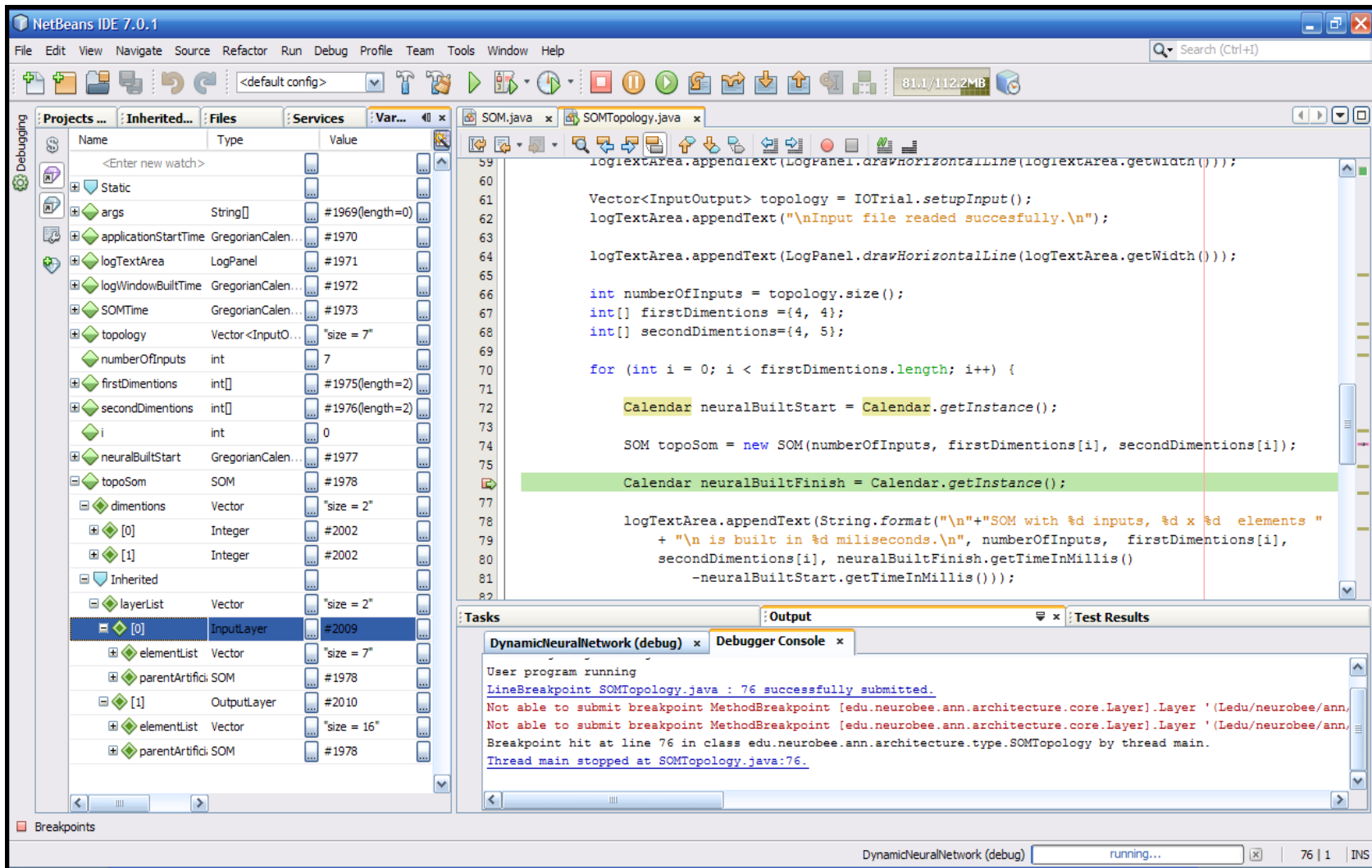
Oluşturulan programın test aşamasında aşağıdaki 4 kriter göz önünde bulundurulmuştur: hesaplama adımlarının doğruluğu, mimari yapıyı doğru kurması, genel sonuçların doğruluğu (yakınsadığı sonuçlar) ve zaman performansı.

Hesaplama adımlarının doğruluğunda ve mimari yapının doğru oluşturulması aşamasında NetBeans editörünün (sonra Eclipse editörünün) debug menüsü kullanılmıştır. Debug ekranından bir görüntü Şekil 5.2'de verilmiştir.

Ekranın sol tarafında değişkenler ve aldıkları değerler gözükmemektedir. Sağ tarafta ise çalıştırılan kodlar gözükmemektedir. Bir sonraki adımda çalıştırılacak kodlar koyu renkle üzeri çizilmiş (yeşil) olan satırdır. Bu satır çalıştırıldıktan sonra sol tarafta yer alan değişkenlerden ilgili olanların değerleri güncellenecektir. Değişkenlerin adları, türleri ve hafızanın hangi bit'inde saklandığı ekranda açıkça görülmektedir. Temel değişken cinsinden değerleri (int, double, string, char vb.) görmek için ise ilgili satırın en sağındaki düğmeye tıklamak yeterlidir.

Debug ekranları kullanılarak verileri doğru okuyup okumadığı, aktarım hataları olup olmadığı ve yapay sinir ağının doğru hiyerarşik yapıya sahip olup olmadığı kontrol edilmiştir. Farklı boyut ve kombinasyonlardaki yapay sinir ağları oluşturulmuş ve yapıları kontrol edilmiştir.

Önerilen algoritmalar adım adım işletilmiş, önerilen algoritmanın çalışma prensibi ile yazılan kodun çalışması kontrol edilerek planlanandan farklı işleyişlerin önüne geçilmiştir.



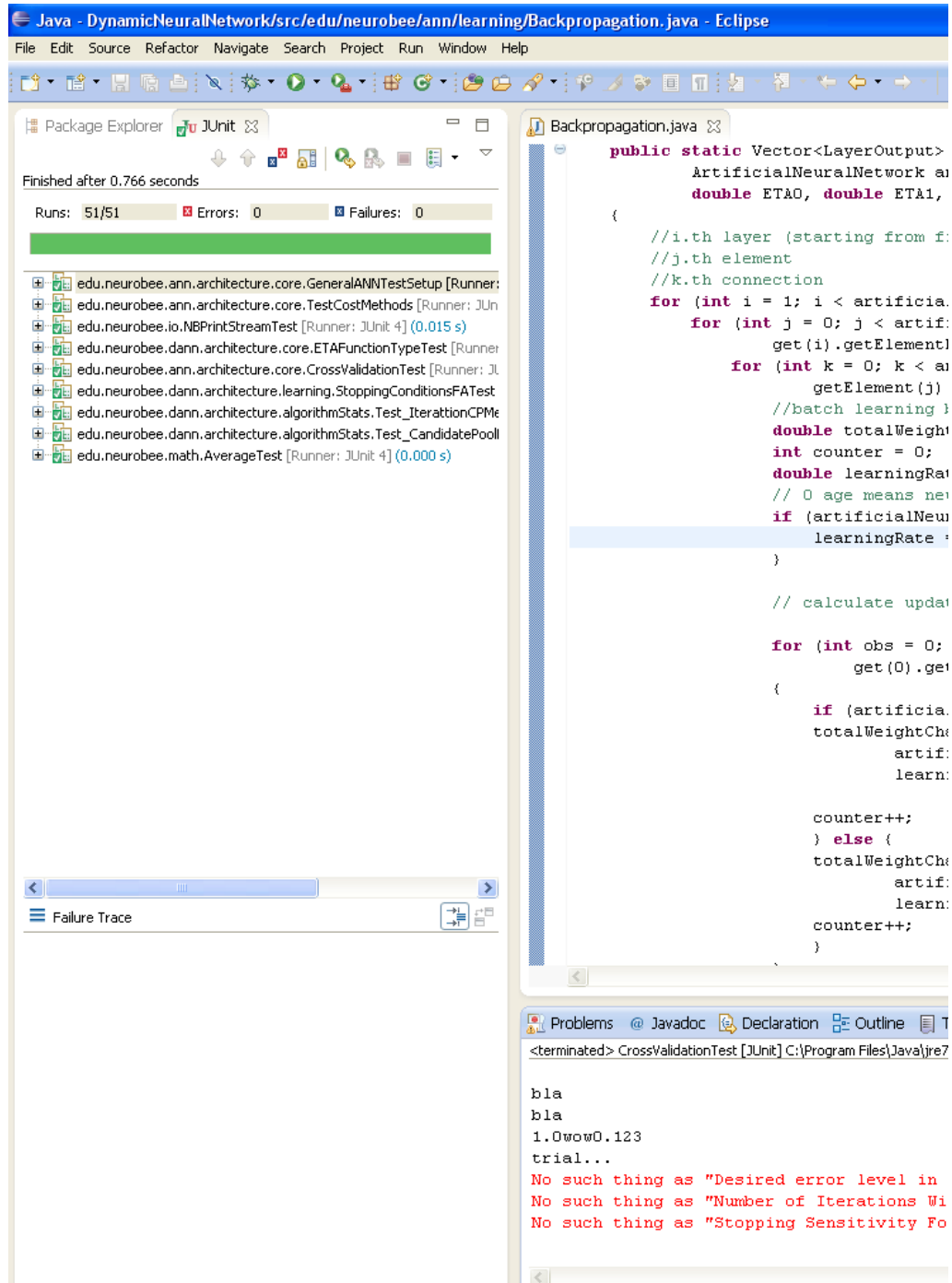
Şekil 5.2 : NetBeans debug ekranı

Ayrıca eğitim kısımları yazılırken farklı yapay sinir ağıları için işlem adımları bilinen örnekler çeşitli kaynaklardan alınmış ve birkaç epoch boyunca değişkenlerin aldığı değerler kontrol edilmiştir. Daha sonra eğitim sonucu oluşan genel sonuçlar (yakınsamalar) yine uzun koşumda sonuçları bilinen örneklerle karşılaştırılarak kontrol edilmiştir.

Zaman performansı için belli örneklerde Neurosolutions programı ile hesaplama hızları karşılaştırılmış, MLP ve SOM yapıları için hızları gözlenmiştir. Her iki yapıda da genel olarak NeuroBee programının anlık grafik çizdirmediği için çok daha hızlı çalıştığı görülmüştür. NeuroSolutions programında grafikleri kapatıldığında her ikisinin benzer hızlarda çalıştığı görülmüştür. Bu güne kadar yapılan denemelerde NeuroSolutions programının hızı hep yeterli olmuştur. Dolayısıyla NeuroBee programında çalışma hızının, kestirme yollar kullanmadan, her elemanı ve katmanı tek tek ele alıp hiyerarşik yapıda hesaplamalarını oluşturduğu yapısının hız açısından hiçbir sorun teşkil etmediği görülmüştür. Bu sebeple önerilen algoritmaların çalışma hızının mevcut algoritma ve paket programların çalışma hızıyla kıyaslanması bu çalışmanın kapsamı dışında tutulmuştur.

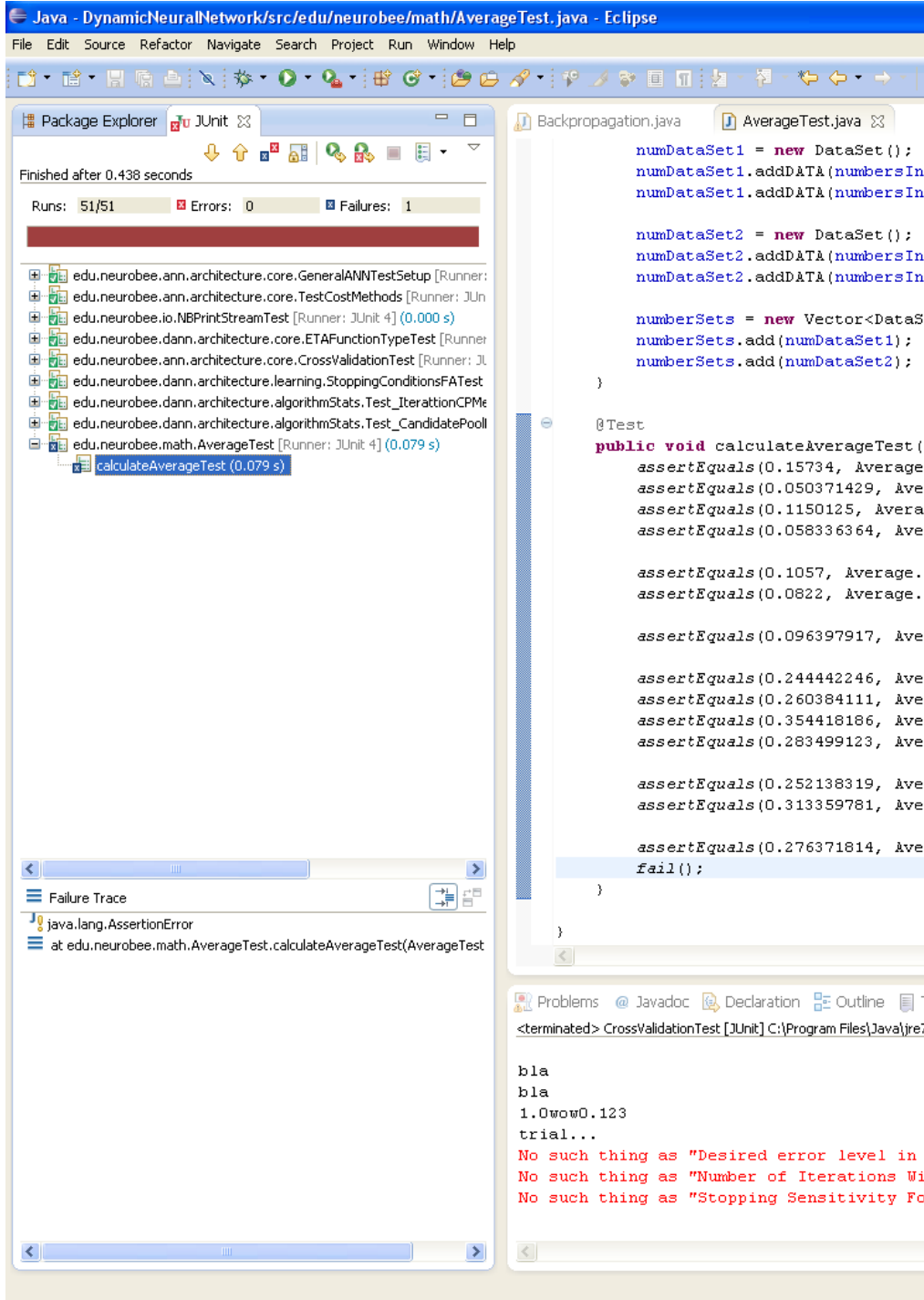
Yazılım geliştirilirken gerekli görülen karmaşık, birçok sınıfın işlemlere dahil olduğu noktalarda TDD (Test Driven Development) benimsenmiştir. Bu yaklaşımda öncelikle verilen girdiler için istenen çıktılar ortaya konulup yazılımda geliştirilecek metotlar için testleri yazılmakta sonra bu testler tamamen doğru çalışıncaya kadar yazılım geliştirmesi yapılmaktadır.

TDD yaklaşımında JUnit 4 Java test eklentisi kullanılmıştır ve testler JUnit 4'e uygun olarak yazılmıştır. Eclipse editörünün JUnit test ekranları Şekil 5.3'te ve Şekil 5.4'te görülmektedir. Şekil 5.3'te tüm testlerin başarıyla sonuçlandığı Şekil 5.4'te ise son testin istinilen girdiler için istenilen çıktıları elde edilemediği görülmektedir. Bu testler özellikle büyük yazılımlarda önceden yazılan kısımların yeni yazılan kısımlarla bozulup bozulmadığını hızlı bir şekilde tespit etmek için kullanılır.



Şekil 5.3 : Eclipse JUnit test ekranı (başarılı).

Şekil 5.4’te sol sütundaki testlerden en sonuncusu hariç hepsinin başında yeşil tik vardır. Son olan “calculateAverageTest” ise başında mavi bir çarpı taşımaktadır. Bu durum bu testin kullandığı kodların yanlış çalıştığını göstermektedir. Sol sütunun alt kısmında ise testin neden başarısız olduğu, hangi değerlere ulaşılması gerekirken hangi değerlerin yazılımla elde edildiği açıklanmaktadır.



Şekil 5.4 : Eclipse JUnit test ekranı (başarısız).

5.6 Algoritmaların Durma Koşulları

Durma koşulları temel algoritmalar ve geri yayılım algoritması için ayrı ayrı tanımlanmıştır. Geri yayılım algoritması için durma koşulları ise şöyledir:

- Maksimum iterasyon sayısına ulaşılması,
- Hatanın ardışık iterasyonlar boyunca değişmemesi,
- İstenilen hata seviyesine / sınıflama seviyesine ulaşılması.

Geri yayılım algoritmasının durmasında yukarıdaki koşulların sağlanıp sağlanmadığını kontrol için aşağıda verilen parametreler kullanılmaktadır:

- Maksimum iterasyon sayısı
- İstenen hata seviyesi / sınıflama seviyesi
- Hatanın değişmeden kalmasına müsaade edilen maksimum iterasyon sayısı
- Hata değişikliği duyarlılığı

Yukarıdaki parametrelerden hata değişiklik duyarlılığı mevcut hatanın seviyesinin değişmediği kabul edilecek değeri göstermektedir. Başka bir değişle ardışık iki iterasyon arasında belirtilen duyarlılıktan daha az bir fark olursa bu iki iterasyon arasında hatanın değişmediği kabul edilmektedir. Çizelge 5.1’de bu çalışma kapsamında kullanılan değerler verilmiştir. Bu değerlerden maksimum iterasyon sayısı, hatanın değişmeden kalmasına müsaade edilen maksimum iterasyon sayısı ve Hata değişikliği duyarlılığı NeuroSolutions programında öntanımlı kullanılan değerlerdir. İstenen hata seviyesi / sınıflama seviyesi ise SPSS Celementine programında öntanımlı kullanılan değerdir.

Çizelge 5.1 : Geri yayılım algoritması durma parametreleri öntanımlı değerleri.

Parametre	Öntanımlı Değer
Maksimum iterasyon sayısı	1000
İstenen hata seviyesi / sınıflama seviyesi	0,00001 / %100
Hatanın değişmeden kalmasına müsaade edilen maksimum iterasyon sayısı	30
Hata değişikliği duyarlılığı	0.0001

Bu çalışmada önerilen TA01V01, TA02V01, TA03V01 ve TA03V02 algoritmaları için durma koşulları aşağıdaki gibidir:

- Maksimum iterasyon sayısına ulaşılması,
- Maksimum bitiş elemanı sayısına ulaşılması ulaşılması.

Bu algoritmaların durmasında yukarıdaki koşulların sağlanıp sağlanmadığını kontrol için aşağıda verilen parametreler kullanılmaktadır.:

- Maksimum iterasyon sayısı
- Maksimum bitiş elemanı sayısı.

Yukarıdaki parametrelerden maksimum iterasyon sayısı GYA maksimum iterasyon sayısından farklı olup önerilen algoritmaların kendi içlerinde en fazla kaç iterasyon çalışacaklarını göstermektedir. Maksimum bitiş elemanı sayısı ise problemde probleme değişmekte ve ön tanımlı bir değeri bulunmamaktadır.

5.7 Aday Havuzu Oluşturma Yöntemleri

Temel algoritmaların işleyebilmesi için aday havuzlarının oluşturulup bu adaylar arasından seçim yapılabilmesi gerekmektedir. Aday havuzu için aday oluşturulurken YSA'nın yapısından bazı kopyalar almak, YSA'nın elde ettiği birikimi yeni eklenen elemanlarla aktarabilmesi *esinlenme (inspire)* açısından önemlidir.

Bazı durumlarda YSA'nın mevcut bilgisinin bir kısmını tamamen aktarıp mevcut bilgisinde de değişiklik yapmak süratiyle kendisini yerel optimumlardan kurtarabilecek bir *unutma (forget)* yapması gerekebilir. Bazı durumlarda DYSA'nın yapacağı öğrenme önceki edindiği bilgi birikiminden alakasız veya önceki bilgi birikimine tamamen ters olabilir. Yukarıda bahsedilen durumları da dikkate alarak hazırlanan aday oluşturma yöntemleri aşağıdaki başlıklar altında açıklanabilir.

5.7.1 Kopyalama ile tek EE ekleme yöntemi (#EE00)

Mevcut bir EE'nin birebir kopyalanması suretiyle yeni bir EE adayı oluşturma Bölüm 4.3.1'de açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.5'te verilmiştir.

```

Mevcut bir EE'nı seç (EE1).
Yeni Bir EE oluştur(EE2).
EE2'ye kopyalanan EE1 ile aynı tür transfer fonksiyonunu ata.
FOR kopyalanan EE1'e gelen her bir bağlantı için;
    EE2'ye yeni bir bağlantı ekle.
    Yeni bağlatının geldiği PE'nı kopyalanan EE1'in
    bağlantısının geldiği PE olarak ata.
    Yeni bağlantının ağırlığına kopyalanan EE1'nin aynı
    bağlantısının ağırlığını ata.
END FOR
EE2'ye EE1'in diğer tüm özelliklerini ata.
RETURN EE2.

```

Şekil 5.5 : Kopyalama ile tek EE ekleme yönteminin (#EE00) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağında en az bir bitiş elemanının halihazırda bulunması gerekmektedir.

5.7.2 Rassal tek EE ekleme yöntemi (#EE01)

Yapay sinir ağına rassal ağırlıklı yeni bir bitiş elemanı eklenmesi Bölüm 4.3.2'de açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.6'da verilmiştir.

```

Yeni Bir EE oluştur.
Yeni EE'ye aday havuzunda önceden tanımlanmış olan türde
transfer fonksiyonunu ata.
FOR Bir önceki katmandaki her bir düğüm için;
    Yeni EE'ye yeni bir bağlantı ekle.
    Yeni bağlatının geldiği PE'nı seçili PE olarak ata.
    Yeni bağlantının ağırlığını rassal olarak ata.
END FOR
RETURN EE'

```

Şekil 5.6 : Rassal tek EE ekleme yönteminin (#EE01) sözde programı.

Bu yöntem mevcut bir bitiş elemanına ihtiyaç duymadığından bitiş elemanı olmayan yapay sinir ağlarında da kullanılabilir. Bağlantının ağırlığının tespitinde NeuroBee'de ön tanımlı olan sezgisel yöntemle tespit edilen aralıkta düzgün dağılımla üretilen rassal sayılar kullanılmaktadır.

5.7.3 Ayırma ile tek EE ekleme yöntemi (#EE02)

Mevcut bir bitiş elemanının, birleşimleri kendisini verecek şekilde iki bitiş elemanına bölünerek yeni bir bitiş elemanı eklenmesi Bölüm 4.3.3'te açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.7'de verilmiştir.

```
Mevcut bir EE'nı seç. (EE1)
Yeni bir EE oluştur. (EE2)
EE2'ye kopyalanan EE1 ile aynı tür transfer fonksiyonunu ata.
FOR EE1'e gelen her bir bağlantı için
    EE2'ye yeni bir bağlantı ekle.
    Yeni bağlantının geldiği PE'nı EE1'in seçili bağlantısının
        geldiği PE olarak ata.
    Yeni bağlantının ağırlığını EE1'in seçili bağlantısının
        bağlantı ağırlığı olarak ata.
    Bir rassal sayı belirle.
    IF rassal sayı sabit bir eşik değerden küçükse THEN
        Mevcut EE'nin bağlantısını sıfıra yakın bir değer
         $(0+\varepsilon)$  olarak ata
    ELSE
        Yeni EE'nin bağlantısını sıfıra yakın bir değer
         $(0+\varepsilon)$  olarak ata
    END IF
END FOR
RETURN EE2.
```

Şekil 5.7 : Ayırma ile tek EE ekleme yönteminin (#EE02) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağında en az bir bitiş elemanının hâlihazırda bulunması gerekmektedir. Ayrıca dikkat edilmesi gereken nokta seçilen bitiş elemanın da değişikliğe uğramasıdır. Bu yöntemle iki yeni bitiş elemanı oluşmakta ancak oluşan bitiş elemanlarından bir tanesi seçilen bitiş elemanının yerine kullanılmaktadır. Dolayısıyla seçilen bitiş elemana gelen bağlantıların bazı ağırlıkların sıfıra yakın bir değer atanması nedeniyle bilgi kaybı olmaktadır.

5.7.4 Paylaştırma ile tek EE ekleme yöntemi (#EE03)

Mevcut bir bitiş elemanının, toplamaları kendisini verecek şekilde iki bitiş elemanına bölünerek yeni bir bitiş elemanı eklenmesi Bölüm 4.3.4'te açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.8'de verilmiştir.

```

Mevcut bir EE'nı seç. (EE1)
Yeni bir EE oluştur. (EE2)
EE2'ye kopyalanan EE1 ile aynı tür transfer fonksiyonunu ata.
FOR EE1'e gelen her bir bağlantı için
    EE2'ye yeni bir bağlantı ekle.
    Yeni bağlatının geldiği PE'nı EE1'in seçili bağlantısının
        geldiği PE olarak ata.
    Yeni bağlantının ağırlığını EE1'in seçili bağlantısının
        bağlantı ağırlığı olarak ata.
    (0,1) aralığından düzgün dağılımla rassal sayı (RS) seç.
    EE2'nin bağlantısının ağırlığını EE1'inn aynı bağlantı
        ağırlığı ile RS çarpımı olarak ata.
    EE1'in bağlatısının ağırlığını ise mevcut ağırlığı ile
        (1-RS) çarpımı olarak ata.
END FOR
RETURN EE2.

```

Şekil 5.8 : Paylaştırma ile tek EE ekleme yönteminin (#EE03) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağında en az bir bitiş elemanının hâlihazırda bulunması gerekmektedir. Ayrıca bu yöntemde dikkat edilmesi gereken nokta seçilen bitiş elemanının da değişikliğe uğramasıdır.

5.7.5 Ters işaretli ağırlıkla EE ekleme yöntemi (#EE04)

Yapay sinir ağına ağırlıkları mevcut bir bitiş elemanı ile ters işaretli yeni bir bitiş elemanı eklenmesi daha önce Bölüm 4.3.5'te açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.9'da verilmiştir.

```

Mevcut bir EE'nı seç. (EE1)
Yeni bir EE oluştur. (EE2)
EE2'ye kopyalanan EE1 ile aynı tür transfer fonksiyonunu ata.
FOR EE1'e gelen her bir bağlantı için
    EE2'e yeni bir bağlantı ekle.
    Yeni bağlatının geldiği PE'nı EE1'in seçili bağlantısının
        geldiği PE olarak ata.
    EE2'nin bağlantısının ağırlığını EE1'inn aynı bağlantı
        ağırlığı ile -1'in çarpımı olarak ata.
END FOR
RETURN EE2.

```

Şekil 5.9 : Ters işaretli ağırlıkla EE ekleme yönteminin (#EE04) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağında en az bir bitiş elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde ilk seçilen bitiş elemanı hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır.

5.7.6 Toplama ile EE ekleme yöntemi (#EE05)

Yapay sinir ağına ağırlıkları mevcut iki bitiş elemanının toplamı olacak yeni bir bitiş elemanı eklenmesi daha önce Bölüm 4.3.6'da açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.10'da verilmiştir.

```
Mevcut iki farklı EE'nı seç. (EE1,EE2)
Yeni bir EE oluştur. (EE3)
Yeni EE3'e aday havuzunda önceden tanımlanmış olan türde
transfer fonksiyonunu ata.
FOR EE1'e gelen her bir bağlantı için
    EE3'e yeni bir bağlantı ekle.
    Yeni bağlatının geldiği PE'nı EE1'in seçili bağlantısının
    geldiği PE olarak ata.
    Yeni bağlantının ağırlığını EE1'in ve EE2'nin ilgili
    bağlantılarının ağırlıkları toplamı olarak ata.
END FOR
RETURN EE3.
```

Şekil 5.10 : Toplama ile EE ekleme yönteminin (#EE05) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağında en az iki bitiş elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde seçilen iki bitiş elemanı da hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır.

5.7.7 Birleştirme ile EE ekleme yöntemi (#EE06)

Yapay sinir ağına ağırlıkları mevcut iki bitiş elemanının birleşimi olacak yeni bir bitiş elemanı eklenmesi daha önce Bölüm 4.3.7'de açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.11'de verilmiştir.

```

Mevcut iki farklı EE'nı seç. (EE1,EE2)
Yeni bir EE oluştur. (EE3)
Yeni EE3'e aday havuzunda önceden tanımlanmış olan türde
transfer fonksiyonunu ata.
FOR EE1'e gelen her bir bağlantı için
    EE3'e yeni bir bağlantı ekle.
    Yeni bağlantının geldiği PE'nı EE1'in seçili bağlantısının
    geldiği PE olarak ata.
    Bir rassal sayı belirle.
    IF rassal sayı sabit bir eşik değerden küçükse THEN
        EE3'ün bağlantısının ağırlığını EE1'in ilgili
        bağlantısının ağırlığı olarak ata
    ELSE
        EE3'ün bağlantısının ağırlığını EE2'in ilgili
        bağlantısının ağırlığı olarak ata
    END IF
END FOR
RETURN EE3.

```

Şekil 5.11 : Birleştirme ile EE ekleme yönteminin (#EE06) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağında en az iki bitiş elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde seçilen iki bitiş elemanı da hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır.

5.7.8 Ayırma ile iki aday EE ekleme yöntemi (#EE07)

Yapay sinir ağında mevcut bulunan bir bitiş elemanının birleşimi kendisini verecek şekilde iki ayrı bitiş elemanına bölünmesi daha önce Bölüm 4.3.8'de açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.12'de verilmiştir.

Bu yöntemin uygulanacağı yapay sinir ağında en az bir bitiş elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde seçilen bitiş elemanı hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır ve bu yöntemle iki ayrı bitiş elemanı adayı elde edilmektedir.

```

Mevcut bir EE'nı seç. (EE1)
Yeni iki EE oluştur. (EE2, EE3)
EE2 ve EE3'e kopyalanan EE1 ile aynı tür transfer fonksiyonunu
ata.
FOR EE1'e gelen her bir bağlantı için
    EE2'ye ve EE3'e yeni birer bağlantı ekle.
    Yeni bağlatıların geldiği PE'nı EE1'in seçili
    bağlantısının geldiği PE olarak ata.
    Bir rassal sayı belirle.
    IF rassal sayı sabit bir eşik değerden küçükse THEN
        EE2'nin bağlantısının ağırlığını EE1'in ilgili
        bağlantısının ağırlığı olarak ata
        EE3'ün bağlantısının ağırlığını sıfıra yakın bir
        değer  $(0+\epsilon)$  olarak ata
    ELSE
        EE2'nin bağlantısının ağırlığını sıfıra yakın bir
        değer  $(0+\epsilon)$  olarak ata
        EE3'ün bağlantısının ağırlığını EE1'in ilgili
        bağlantısının ağırlığı olarak ata
    END IF
END FOR
RETURN EE2 ve EE3.

```

Şekil 5.12 : Ayırma ile iki aday EE ekleme yönteminin (#EE07) sözde programı.

5.7.9 Paylaştırma ile iki aday EE ekleme yöntemi (#EE08)

Yapay sinir ağında mevcut bulunan bir bitiş elemanının toplamaları kendisini verecek şekilde iki ayrı bitiş elemanına bölünmesi yöntemi daha önce Bölüm 4.3.9'da açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.13'te verilmiştir.

Bu yöntemin uygulanacağı yapay sinir ağında en az bir bitiş elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde seçilen bitiş elemanı hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır ve bu yöntemle iki ayrı bitiş elemanı adayı elde edilmektedir.

```

Mevcut bir EE'nı seç.(EE1)
Yeni iki EE oluştur.(EE2, EE3)
EE2 ve EE3'e kopyalanan EE1 ile aynı tür transfer fonksiyonunu
    ata.
FOR EE1'e gelen her bir bağlantı için
    EE2'ye ve EE3'e yeni birer bağlantı ekle.
    Yeni bağlatıların geldiği PE'nı EE1'in seçili
    bağlantısının geldiği PE olarak ata.
    Bir rassal sayı belirle.(RS)
    EE2'nin bağlantısının ağırlığını EE1'in ilgili
    bağlantısının ağırlığı ile RS çarpımı olarak ata.
    EE3'ün bağlantısının ağırlığını EE1'in ilgili
    bağlantısının ağırlığı ile (1-RS) çarpımı olarak
    ata.
END FOR
RETURN EE2 ve EE3.

```

Şekil 5.13 : Paylaştırma ile iki aday EE ekleme yönteminin (#EE08) sözde programı.

5.7.10 Kopyalama ile tek PE ekleme yöntemi (#PE00)

```

Mevcut bir PE'nı seç (PE1) .
Yeni Bir PE oluştur(PE2) .
PE2'ye kopyalanan PE1 ile aynı tür transfer fonksiyonunu ata.
FOR kopyalanan PE1'e gelen her bir bağlantı için;
    PE2'ye yeni bir bağlantı ekle.
    Yeni bağlatının geldiği SE'nı kopyalanan PE1'in
    bağlantısının geldiği SE olarak ata.
    Yeni bağlantının ağırlığına kopyalanan PE1'nin aynı
    bağlantısın ağırlığını ata.
END FOR
FOR PE1'dan bir sonraki katmana giden her bağlantı için;
    PE2'ye yeni bir bağlantı ekle.
    Yeni bağlatının geldiği PE'nı kopyalanan olarak ata.
    Yeni bağlatının gittiği EE'nı seçili bağlantıdaki EE
    olarak ata.
    Yeni bağlantının ağırlığına kopyalanan PE1'nin aynı
    bağlantısın ağırlığını ata.
END FOR
PE2'ye PE1'in diğer tüm özelliklerini ata.
RETURN PE2.

```

Şekil 5.14 : Kopyalama ile tek PE ekleme yönteminin (#PE00) sözde programı.

Mevcut bir PE'nin birebir kopyalanması suretiyle yeni bir PE adayı oluşturma Bölüm 4.3.10'da açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.14'te verilmiştir.

Bu yöntemin uygulanacağı yapay sinir ağı gizli katmanında en az bir işlem elemanının hâlihazırda bulunması gerekmektedir.

5.7.11 Rassal tek PE ekleme yöntemi (#PE01)

Yapay sinir ağına rassal ağırlıklı yeni bir işlem elemanı eklenmesi Bölüm 4.3.11'de açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.15'te verilmiştir.

```
Yeni Bir PE oluştur.
Yeni PE'ye aday havuzunda önceden tanımlanmış olan türde
transfer fonksiyonunu ata.
FOR Bir önceki katmandaki her bir düğüm için;
    Yeni PE'ye yeni bir bağlantı ekle.
    Yeni bağlantının geldiği SE'nı seçili SE olarak ata.
    Yeni bağlantının ağırlığını rassal olarak ata.
END FOR
FOR Bir sonraki katmandaki her bir düğüm için;
    Yeni PE'nından seçili EE'ye yeni bir bağlantı ekle.
    Bu bağlantının geldiği PE'nı seçili yeni oluşturulan PE
    olarak ata.
    Yeni bağlantının ağırlığını rassal olarak ata.
END FOR
RETURN PE.
```

Şekil 5.15 : Rassal tek PE ekleme yönteminin (#PE01) sözde programı.

Bu yöntem mevcut bir işlem elemanına ihtiyaç duymadığından işlem elemanı olmayan yapay sinir ağlarında da kullanılabilir. Bağlantının ağırlığının tespitinde NeuroBee'de ön tanımlı olan sezgisel yöntemle tespit edilen aralıkta düzgün dağılımla üretilen rassal sayılar kullanılmaktadır.

5.7.12 Ayırma ile tek PE ekleme yöntemi (#PE02)

Mevcut bir işlem elemanının, birleşimleri kendisini verecek şekilde iki işlem elemanına bölünerek yeni bir işlem elemanı eklenmesi Bölüm 4.3.12'de açıklanmıştı.

Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.16’de verilmiştir.

```
Mevcut bir PE’nı seç. (PE1)
Yeni bir PE oluştur. (PE2)
PE2’ye kopyalanan PE1 ile aynı tür transfer fonksiyonunu ata.
FOR PE1’e gelen her bir bağlantı için
    PE2’ye yeni bir bağlantı ekle.
    Yeni bağlantının geldiği SE’nı PE1’in seçili bağlantısının
        geldiği SE olarak ata.
    Yeni bağlantının ağırlığını PE1’in seçili bağlantısının
        bağlantı ağırlığı olarak ata.
    Bir rassal sayı belirle.
    IF rassal sayı sabit bir eşik değerden küçükse THEN
        Mevcut PE’nin bağlantısını sıfıra yakın bir değer
         $(0+\epsilon)$  olarak ata
    ELSE
        Yeni PE’nin bağlantısını sıfıra yakın bir değer
         $(0+\epsilon)$  olarak ata
    END IF
END FOR
FOR PE1’den giden her bir bağlantı için
    PE2’den seçili bağlantının gittiği EE’ye yeni bir bağlantı
        ekle.
    Yeni bağlantının ağırlığını PE1’in seçili bağlantısının
        bağlantı ağırlığı olarak ata.
    Bir rassal sayı belirle.
    IF rassal sayı sabit bir eşik değerden küçükse THEN
        Mevcut PE’nin bağlantısını sıfıra yakın bir değer
         $(0+\epsilon)$  olarak ata
    ELSE
        Yeni PE’nin bağlantısını sıfıra yakın bir değer
         $(0+\epsilon)$  olarak ata
    END IF
END FOR
RETURN PE2.
```

Şekil 5.16 : Ayırma ile tek PE ekleme yönteminin (#PE02) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağı gizli katmanında en az bir işlem elemanının hâlihazırda bulunması gerekmektedir. Ayrıca dikkat edilmesi gereken

nokta seçilen işlem elemanın da değişikliğe uğramasıdır. Bu yöntemle iki yeni işlem elemanı oluşmakta ancak oluşan işlem elemanlarından bir tanesi seçilen işlem elemanının yerine kullanılmaktadır. Dolayısıyla seçilen işlem elemana gelen bağlantıların bazı ağırlıkların sıfıra yakın bir değer atanması nedeniyle bilgi kaybı olmaktadır.

5.7.13 Paylaştırma ile tek PE ekleme yöntemi (#PE03)

Mevcut bir işlem elemanının, toplamaları kendisini verecek şekilde iki işlem elemanına bölünerek yeni bir işlem elemanı eklenmesi Bölüm 4.3.13'te açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.17'de verilmiştir.

```
Mevcut bir PE'nı seç. (PE1)
Yeni bir PE oluştur. (PE2)
PE2'ye kopyalanan PE1 ile aynı tür transfer fonksiyonunu ata.
FOR PE1'e gelen her bir bağlantı için
    PE2'ye yeni bir bağlantı ekle.
    Yeni bağlantının geldiği SE'nı PE1'in seçili bağlantısının
        geldiği SE olarak ata.
    Yeni bağlantının ağırlığını PE1'in seçili bağlantısının
        bağlantı ağırlığı olarak ata.
    (0,1) aralığından düzgün dağılımla rassal sayı (RS) seç.
    PE2'nin bağlantısının ağırlığını PE1'inn aynı bağlantı
        ağırlığı ile RS çarpımı olarak ata.
    PE1'in bağlantısının ağırlığını ise mevcut ağırlığı ile
        (1-RS) çarpımı olarak ata.
END FOR
FOR PE1'den giden her bir bağlantı için
    PE2'den seçili bağlantının gittiği EE'ye yeni bir bağlantı
        ekle.
    Yeni bağlantının ağırlığını PE1'in seçili bağlantısının
        bağlantı ağırlığı olarak ata.
    (0,1) aralığından düzgün dağılımla rassal sayı (RS) seç.
    PE2'nin bağlantısının ağırlığını PE1'inn aynı bağlantı
        ağırlığı ile RS çarpımı olarak ata.
    PE1'in bağlantısının ağırlığını ise mevcut ağırlığı ile
        (1-RS) çarpımı olarak ata.
END FOR
RETURN PE2.
```

Şekil 5.17 : Paylaştırma ile tek PE ekleme yöntemi (#PE03) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağı gizli katmanında en az bir işlem elemanının hâlihazırda bulunması gerekmektedir. Ayrıca bu yöntemde dikkat edilmesi gereken nokta seçilen işlem elemanının da değişikliğe uğramasıdır.

5.7.14 Ters işaretli ağırlıkla PE ekleme yöntemi (#PE04)

Yapay sinir ağına ağırlıkları mevcut bir işlem elemanı ile ters işaretli yeni bir işlem elemanı ekleme yöntemi daha önce Bölüm 4.3.14'te açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.18'de verilmiştir.

```
Mevcut bir PE'nı seç. (PE1)
Yeni bir PE oluştur. (PE2)
PE2'ye kopyalanan PE1 ile aynı tür transfer fonksiyonunu ata.
FOR PE1'e gelen her bir bağlantı için
    PE2'e yeni bir bağlantı ekle.
    Yeni bağlatının geldiği SE'nı PE1'in seçili bağlantısının
        geldiği SE olarak ata.
    PE2'nin bağlantısının ağırlığını PE1'inn aynı bağlantı
        ağırlığı ile -1'in çarpımı olarak ata.
END FOR
FOR PE1'den giden her bir bağlantı için
    PE2'den seçili bağlantının gittiği EE'ye yeni bir bağlantı
        ekle.
    PE2'nin bağlantısının ağırlığını PE1'inn aynı bağlantı
        ağırlığı ile -1'in çarpımı olarak ata.
END FOR
RETURN PE2.
```

Şekil 5.18 : Ters işaretli ağırlıkla PE ekleme yönteminin (#PE04) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağı gizli katmanında en az bir işlem elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde ilk seçilen işlem elemanı hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır.

5.7.15 Toplama ile PE ekleme yöntemi (#PE05)

Yapay sinir ağına ağırlıkları mevcut iki işlem elemanının toplamı olacak yeni bir işlem elemanı ekleme yöntemi daha önce Bölüm 4.3.15'te açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.19'da verilmiştir.

```

Mevcut iki farklı PE'nı seç. (PE1, PE2)
Yeni bir PE oluştur. (PE3)
Yeni PE3'e aday havuzunda önceden tanımlanmış olan türde
transfer fonksiyonunu ata.
FOR PE1'e gelen her bir bağlantı için
    PE3'e yeni bir bağlantı ekle.
    Yeni bağlatının geldiği SE'nı PE1'in seçili bağlantısının
    geldiği SE olarak ata.
    Yeni bağlantının ağırlığını PE1'in ve PE2'nin ilgili
    bağlantılarının ağırlıkları toplamı olarak ata.
END FOR
FOR PE1'den giden her bir bağlantı için
    PE3'den seçili bağlantının gittiği EE'ye yeni bir bağlantı
    ekle.
    Yeni bağlantının ağırlığını PE1'in ve PE2'nin ilgili
    bağlantılarının ağırlıkları toplamı olarak ata.
END FOR
RETURN PE3.

```

Şekil 5.19 : Toplama ile PE ekleme yöntemi (#PE05) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağı gizli katmanında en az iki işlem elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde seçilen iki işlem elemanı da hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır.

5.7.16 Birleştirme ile PE ekleme yöntemi (#PE06)

Yapay sinir ağına ağırlıkları mevcut iki işlem elemanının birleşimi olacak yeni bir işlem elemanı ekleme yöntemi daha önce Bölüm 4.3.16'da açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.20'de verilmiştir.

Bu yöntemin uygulanacağı yapay sinir ağı gizli katmanında en az iki işlem elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde seçilen iki işlem elemanı da hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır.

```

Mevcut iki farklı PE'nı seç. (PE1, PE2)
Yeni bir PE oluştur. (PE3)
Yeni PE3'e aday havuzunda önceden tanımlanmış olan türde
transfer fonksiyonunu ata.
FOR PE1'e gelen her bir bağlantı için
    PE3'e yeni bir bağlantı ekle.
    Yeni bağlatının geldiği SE'nı PE1'in seçili bağlantısının
    geldiği SE olarak ata.
    Bir rassal sayı belirle.
    IF rassal sayı sabit bir eşik değerden küçükse THEN
        PE3'ün bağlantısının ağırlığını PE1'in ilgili
        bağlantısının ağırlığı olarak ata
    ELSE
        PE3'ün bağlantısının ağırlığını PE2'in ilgili
        bağlantısının ağırlığı olarak ata
    END IF
END FOR
FOR PE1'den giden her bir bağlantı için
    PE3'den seçili bağlantının gittiği EE'ye yeni bir bağlantı
    ekle.
    Bir rassal sayı belirle.
    IF rassal sayı sabit bir eşik değerden küçükse THEN
        PE3'ün bağlantısının ağırlığını PE1'in ilgili
        bağlantısının ağırlığı olarak ata
    ELSE
        PE3'ün bağlantısının ağırlığını PE2'in ilgili
        bağlantısının ağırlığı olarak ata
    END IF
END FOR
RETURN PE3.

```

Şekil 5.20 : Birleştirme ile PE ekleme yönteminin (#PE06) sözde programı.

5.7.17 Ayırma ile iki aday PE ekleme yöntemi (#PE07)

Yapay sinir ağında mevcut bulunan bir işlem elemanının birleşimi kendisini verecek şekilde iki ayrı işlem elemanına bölünmesi yöntemi daha önce Bölüm 4.3.17'de açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.21'de verilmiştir.

```

Mevcut bir PE'nı seç. (PE1)
Yeni iki PE oluştur. (PE2, PE3)
PE2 ve PE3'e kopyalanan PE1 ile aynı tür transfer fonksiyonunu
    ata.
FOR PE1'e gelen her bir bağlantı için
    PE2'ye ve PE3'e yeni birer bağlantı ekle.
    Yeni bağlantıların geldiği SE'nı PE1'in seçili
    bağlantısının geldiği SE olarak ata.
    Bir rassal sayı belirle.
    IF rassal sayı sabit bir eşik değerden küçükse THEN
        PE2'nin bağlantısının ağırlığını PE1'in ilgili
        bağlantısının ağırlığı olarak ata
        PE3'ün bağlantısının ağırlığını sıfıra yakın bir
        değer  $(0+\epsilon)$  olarak ata
    ELSE
        PE2'nin bağlantısının ağırlığını sıfıra yakın bir
        değer  $(0+\epsilon)$  olarak ata
        PE3'ün bağlantısının ağırlığını PE1'in ilgili
        bağlantısının ağırlığı olarak ata
    END IF
END FOR
FOR PE1'den giden her bir bağlantı için
    PE2'den ve PE3'den seçili bağlantının gittiği EE'ye yeni
    bir bağlantı ekle.
    Bir rassal sayı belirle.
    IF rassal sayı sabit bir eşik değerden küçükse THEN
        PE2'nin bağlantısının ağırlığını PE1'in ilgili
        bağlantısının ağırlığı olarak ata
        PE3'ün bağlantısının ağırlığını sıfıra yakın bir
        değer  $(0+\epsilon)$  olarak ata
    ELSE
        PE2'nin bağlantısının ağırlığını sıfıra yakın bir
        değer  $(0+\epsilon)$  olarak ata
        PE3'ün bağlantısının ağırlığını PE1'in ilgili
        bağlantısının ağırlığı olarak ata
    END IF
END FOR
DEFINITION PE2 ve PE3

```

Şekil 5.21 : Ayırma ile iki aday PE ekleme yönteminin (#PE07) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağı gizli katmanında en az bir işlem elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde seçilen işlem

elemanı hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır ve bu yöntemle iki ayrı işlem elemanı adayı elde edilmektedir.

5.7.18 Paylaştırma ile iki aday PE ekleme yöntemi (#PE08)

Yapay sinir ağında mevcut bulunan bir işlem elemanının toplamaları kendisini verecek şekilde iki ayrı işlem elemanına bölünmesi yöntemi daha önce Bölüm 4.3.18’de açıklanmıştı. Bu yöntem için NeuroBee yazılımında kullanılan sözde program ise Şekil 5.22’de verilmiştir.

```
Mevcut bir PE'nı seç.(PE1)
Yeni iki PE oluştur.(PE2, PE3)
PE2 ve PE3'e kopyalanan PE1 ile aynı tür transfer fonksiyonunu
    ata.
FOR PE1'e gelen her bir bağlantı için
    PE2'ye ve PE3'e yeni birer bağlantı ekle.
    Yeni bağlatıların geldiği SE'nı PE1'in seçili
    bağlantısının geldiği SE olarak ata.
    Bir rassal sayı belirle.(RS)
    PE2'nin bağlantısının ağırlığını PE1'in ilgili
    bağlantısının ağırlığı ile RS çarpımı olarak ata.
    PE3'ün bağlantısının ağırlığını PE1'in ilgili
    bağlantısının ağırlığı ile (1-RS) çarpımı olarak
    ata.
END FOR
FOR PE1'den giden her bir bağlantı için
    PE2'den ve PE3'den seçili bağlantının gittiği EE'ye yeni
    bir bağlantı ekle.
    Bir rassal sayı belirle.(RS)
    PE2'nin bağlantısının ağırlığını PE1'in ilgili
    bağlantısının ağırlığı ile RS çarpımı olarak ata.
    PE3'ün bağlantısının ağırlığını PE1'in ilgili
    bağlantısının ağırlığı ile (1-RS) çarpımı olarak
    ata.
END FOR
RETURN PE2 ve PE3.
```

Şekil 5.22 : Paylaştırma ile iki aday PE ekleme yönteminin (#PE08) sözde programı.

Bu yöntemin uygulanacağı yapay sinir ağı gizli katmanında en az bir işlem elemanının hâlihazırda bulunması gerekmektedir. Bu yöntemde seçilen işlem elemanı hiçbir değişikliğe uğramadan olduğu gibi kalmaktadır ve bu yöntemle iki ayrı işlem elemanı adayı elde edilmektedir.

5.7.19 Temel algoritma 1 aday havuzu

Temel algoritma 1 aday havuzu TA01V01’de kullanılan bir aday havuzudur. Bu aday havuzunun en önemli özelliği önceki bölümlerde bahsedilen dokuz adet bitiş elemanı oluşturma yöntemleri kullanılarak oluşturulan adayların tutulduğu yer olmasıdır. Buradaki her bir aday mevcut yapıya eklenme potansiyeli olan bir bitiş elemanı ve bu bitiş elemanının bağlantılarıdır.

Aday havuzu oluşturma işlemi bittiğinde bu havuzdaki adayların uygunluğu K ’nın L ’li çapraz geçerliliği çerçevesinde değerlendirilmekte ve en uygun aday mevcut yapay sinir ağı yapısına eklenmektedir. Oluşan yeni yapı bir sonraki adımlarda kullanılmaktadır.

Temel algoritma 1 aday havuzundaki her aday yapı sadece bir adet bitiş elemanı ve bu bitiş elemanına gelen bağlantılarını içerir. Temel algoritma 1 için kullanılan aday havuzunun NeuroBee yazılımında uygulanan sözde programı Şekil 5.23’te verilmiştir.

```
Boş bir aday havuzu oluştur. (AH)
Aday havuzunun transfer fonksiyonu türünü tanımla.
Aday havuzunun büyüklüğünü tanımla.
Başlangıç yapay sinir ağını aday havuzuna ekle.
IF başlangıç yapay sinir ağı bitiş elemanı içeriyorsa THEN
    FOR başlangıç yapay sinir ağındaki her bir EE için
        #EE00 yöntemi ile aday EE oluştur.
        Oluşturulan aday EE’nı aday havuzuna ekle.
    END FOR
ELSE
END IF
WHILE mevcut aday sayısı < aday havuzu büyüklüğü
    Başlangıç yapay sinir ağı için birinci tip aday oluştur.
    Oluşturulan adayı aday havuzuna ekle.
END WHILE
RETURN AH
```

Şekil 5.23 : Temel algoritma 1 için kullanılan aday havuzunun sözde programı.

TA01V01 için kullanılan sadece bitiş elemanlarından oluşan birinci tip aday oluşturma yönteminin sözde programı Şekil 5.24’te verilmiştir. Bu çalışmada “rassal durum”lerin herbirin oluşması eşit olasılığa sahiptir.

```

CASE başlangıç yapay sinir ağı EE sayısı OF
  0 : #EE01 yöntemi ile aday EE oluştur.
  1 : Ön tanımlı olasılıklarla rassal bir case oluştur.
    CASE rassal case OF
      0 : #EE01 yöntemi ile aday EE oluştur.
      1 : #EE02 yöntemi ile aday EE oluştur.
      2 : #EE03 yöntemi ile aday EE oluştur.
      3 : #EE04 yöntemi ile aday EE oluştur.
      4 : #EE07 yöntemi ile aday EE oluştur.
      5 : #EE08 yöntemi ile aday EE oluştur.
    END CASE
OTHERS:
  Ön tanımlı olasılıklarla rassal bir case oluştur.
    CASE rassal case OF
      0 : #EE01 yöntemi ile aday EE oluştur.
      1 : #EE02 yöntemi ile aday EE oluştur.
      2 : #EE03 yöntemi ile aday EE oluştur.
      3 : #EE04 yöntemi ile aday EE oluştur.
      4 : #EE05 yöntemi ile aday EE oluştur.
      5 : #EE06 yöntemi ile aday EE oluştur.
      6 : #EE07 yöntemi ile aday EE oluştur.
      7 : #EE08 yöntemi ile aday EE oluştur.
    END CASE
END CASE
RETURN Oluşturulan aday

```

Şekil 5.24 : Birinci tip aday oluşturma yönteminin sözde programı.

5.7.20 Temel algoritma 2 aday havuzu

Temel algoritma 2 aday havuzu TA02V01 de kullanılan bir aday havuzudur. Bu aday havuzunun en önemli özelliği önceki bölümlerde bahsedilen dokuz adet bitiş elemanı oluşturma yöntemleri kullanılarak oluşturulan adayların tutulduğu yer olmasıdır. Oluşturulan havuzdaki adaylar mevcut yapıya eklenme potansiyeli olan bir bitiş elemanı ve bu bitiş elemanının bağlantılarıdır. Ayrıca bu havuzda bitiş elemanı ve ilgili bağlantılarıyla beraber işlem elemanı ve ilgili bağlantılarının olduğu adaylar da bulunmaktadır. Böylece bazı adaylar sadece bitiş elemanı içerirken bazıları bir bitiş elemanı ve farklı miktarda işlem elemanı da içerir. Bu iki yapının aynı anda değerlendirilmesi algoritmayı ne zaman bitiş elemanı ekleyeceği ne zaman ise işlem elemanı ekleyeceği probleminden kurtarır.

Aday havuzu oluşturma işlemi bittiğinde bu havuzdaki adayların uygunluğu K 'nın L 'li çapraz geçerliliği çerçevesinde değerlendirilmekte ve en uygun aday mevcut yapay sinir ağı yapısına eklenmektedir. Oluşan yeni yapı bir sonraki adımlarda kullanılmaktadır.

TA02V01 de kullanılan bu aday havuzuyla birlikte gizli katmanda eksik işlem elemanı varsa bunlarında aynı zamanda tamamlanması beklenmektedir. Temel algoritma 2 aday havuzundaki her aday yapı bir adet bitiş elemanı veya bir adet bitiş elemanı ile farklı miktarlarda işlem elemanı ile birlikte tüm ilgili bağlantılarını içerir. Bu bağlantılar bitiş elemanı için gelen bağlantılar iken işlem elemanları için hem gelen hem de giden bağlantılardır.

Temel algoritma 2 için kullanılan aday havuzunun NeuroBee yazılımında uygulanan sözde programı Şekil 5.25'te verilmiştir.

TA02V01 için kullanılan sadece bitiş elemanlarından ve bitiş elemanları ile birlikte işlem elemanlarından oluşan birinci tip aday oluşturma yönteminin sözde programı Şekil 5.26'da verilmiştir. Bu çalışmada "rassal durum"ların herbirinin oluşması eşit olasılığa sahiptir.

```

Boş bir aday havuzu oluştur. (AH)
Aday havuzunun transfer fonksiyonu türünü tanımla.
Aday havuzunun büyüklüğünü tanımla.
Başlangıç yapay sinir ağını aday havuzuna ekle.
Kopyalanabilecek max PE sayısını belirle.
IF başlangıç yapay sinir ağı bitiş elemanı içeriyorsa THEN
    FOR başlangıç yapay sinir ağındaki her bir EE için
        #EE00 yöntemi ile aday EE oluştur.
        Oluşturulan aday EE'nı aday havuzuna ekle.
    END FOR
ELSE
END IF
Aday havuzundaki mevcut aday sayısını tespit et. (AS)
FOR aday havuzundaki ilk AS kadar aday için
    FOR 1'den kopyalanabilecek max PE'ye kadar her sayı için
        #PE00 yöntemi ile seçili miktarda PE adayı üret.
        Oluşan aday PE yapısını seçili aday ile birleştir.
        Oluşan yeni yapıyı aday havuzuna ekle.
    END FOR
END FOR
WHILE mevcut aday sayısı < aday havuzu büyüklüğü
    Birinci tip aday oluştur. (BTA)
    Oluşturulan birinci tip adayı aday havuzuna ekle.
    Bu adaya eklenecek max PE sayısını hesapla.
    FOR 1'den kopyalanabilecek max PE'ye kadar her sayı için
        BTA'yı de dikkate alarak ikinci tip aday oluştur.
        Oluşan aday PE yapısını BTA ile birleştir.
        Oluşan yeni yapıyı aday havuzuna ekle.
    END FOR
END WHILE
RETURN AH

```

Şekil 5.25 : Temel algoritma 2 için kullanılan aday havuzunun sözde programı.

```

CASE başlangıç yapay sinir ağı seçili katman PE sayısı OF
  0 : #PE01 yöntemi ile aday PE oluştur.
  1 : Ön tanımlı olasılıklarla rassal bir case oluştur.
    CASE rassal case OF
      0 : #PE01 yöntemi ile aday PE oluştur.
      1 : #PE02 yöntemi ile aday PE oluştur.
      2 : #PE03 yöntemi ile aday PE oluştur.
      3 : #PE04 yöntemi ile aday PE oluştur.
      4 : #PE07 yöntemi ile aday PE oluştur.
      5 : #PE08 yöntemi ile aday PE oluştur.
    END CASE
OTHERS:
  Ön tanımlı olasılıklarla rassal bir case oluştur.
    CASE rassal case OF
      0 : #PE01 yöntemi ile aday PE oluştur.
      1 : #PE02 yöntemi ile aday PE oluştur.
      2 : #PE03 yöntemi ile aday PE oluştur.
      3 : #PE04 yöntemi ile aday PE oluştur.
      4 : #PE05 yöntemi ile aday PE oluştur.
      5 : #PE06 yöntemi ile aday PE oluştur.
      6 : #PE07 yöntemi ile aday PE oluştur.
      7 : #PE08 yöntemi ile aday PE oluştur.
    END CASE
END CASE
RETURN Oluşturulan aday

```

Şekil 5.26 : İkinci tip aday oluşturma yöntemi sözde programı.

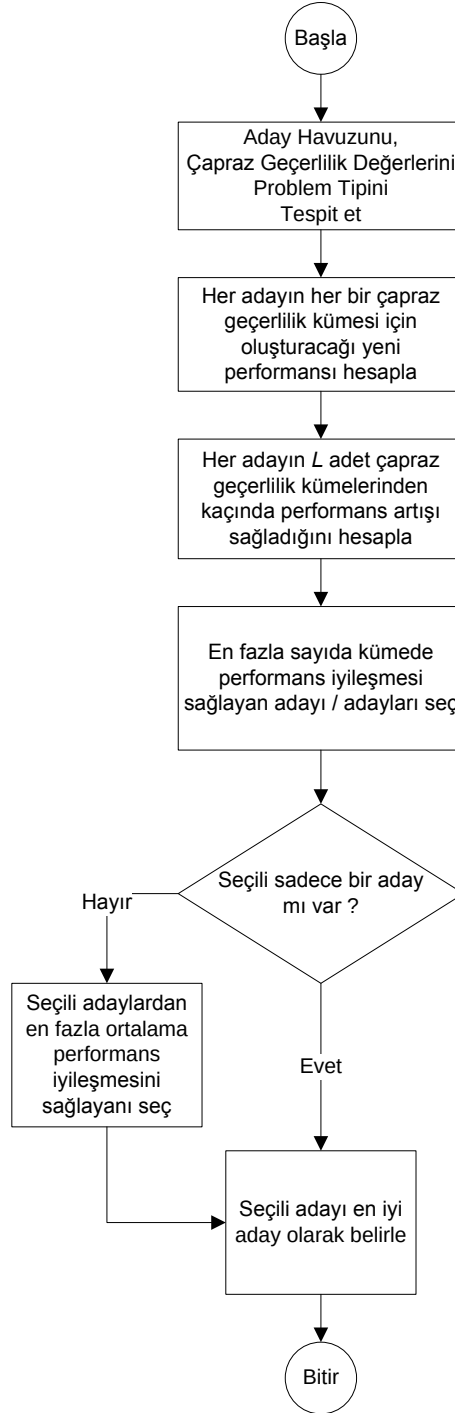
5.7.21 Temel algoritma 3 aday havuzu

İlk iki temel algoritma aday havuzlarından farklı olarak bu aday havuzlarını oluşturmak için özel yöntemler yoktur. TA03V01 ve TA03V02 algoritması çalışırken oluşan öz örgütlemeli haritalar, bunların çok katmanlı algılayıcıya çevrilmiş halleri ve son olarak da TA01V01 ve TA02V01 algoritmaları sonucunda oluşan yapay sinir ağlarını tutmak için kullanılırlar.

Bu aday havuzlarındaki her bir aday bir yapay sinir ağıdır. TA03V01 ve TA03V02 kapsamındaki aday havuzları liste olarak NeuroBee yazılımında saklanmaktadır.

5.8 Aday Seçimi

Aday havuzundan en uygun adayın seçimi sırasında kullanılan prosedürün akış şeması Şekil 5.27’de verilmiştir.



Şekil 5.27 : Aday seçim prosedürü.

5.9 Parametreler

Bu çalışma kapsamında adaptif kavrama oranı (Denklem 2.29) ve momentum (Denklem 2.28) geri yayılım algoritmasında kullanılmıştır.

Önerilen algoritmalar için geliştirilen yazılımda (NeuroBee) aksi belirtilmediği sürece Çizelge 5.2’de belirtilen ön tanımlı değerler kullanılmıştır.

Çizelge 5.2 : NeuroBee parametreleri öntanımlı değerleri.

Parametre	Öntanımlı Değer
Adaptif Kavrama Oranı (η) Başlangıç Değeri	0,7
Adaptif Kavrama Oranı (η) Sabit Artış Değeri (a)	0,1
Adaptif Kavrama Oranı (η) Geometrik Azalış Katsayısı (b)	0,2
Adaptif Kavrama Oranı (η) Kısa Vadeli Hata Süresi (t)	Son 3 İterasyon
Adaptif Kavrama Oranı (η) Uzun Vadeli Hata Süresi ($t+c$)	Son 5 İterasyon
Momentum (α) Oranı	0,5
Aday Havuzu Büyüklüğü	100
Transfer Fonksiyonu ($\phi(.)$)	$\tanh(.)$
Aday Havuzu Transfer Fonksiyonu ($\phi(.)$)	$\tanh(.)$
Rassal Snaptik Ağırlık Aralığı	(-1,1)

6. ALGORİTMALARIN PERFORMANS ÖLÇÜMÜ

Bu bölümde önerilen algoritmaların performanslarının nasıl ölçüldüğü açıklanmıştır. Performans ölçümünde temel kriter olarak her istenilen çıktının istenen değere ne kadar yaklaştığını ölçen göstergeler alınmaktadır. Bu çalışmada bu tip göstergelerden biri olan ortalama hata kareleri (OHK) kullanılmıştır.

Ortalama hata kareleri problemin türüne göre istenilen başarıyı ifade etmekte yetersiz kalabilmektedir. Bu genellikle ikili fonksiyon ve sınıflama problemlerinde rastlanan bir durumdur. Bu amaçla bu çalışma kapsamında ikili fonksiyon doğru sonuç yüzdesi ve doğru sınıflama yüzdesi de kullanılmıştır.

6.1 Ortalama Hata Kareleri (OHK)

Ortalama hata kareleri (Mean square errors) istenilen değerden ne kadar farklı sonuçlar elde edildiğini işaretler. İşaretsiz olarak aşağıdaki formülle hesaplar:

$$e_k^2 = \sum_{i=1}^n (o_{ki}^{des.} - o_{ki})^2 \quad (6.1)$$

$$OHK = \frac{1}{mn} \sum_{k=1}^m e_k^2 \quad (6.2)$$

Burada n adet çıktı özelliği ve m adet gözlem için; o_{ki} , i 'nci çıktı özelliğinin k 'ıncı hesaplanan değeri, $o_{ki}^{des.}$ ise i 'nci çıktı özelliğinin k 'ıncı istenen değeridir.

Yukarıda verilen formüller de dikkate alındığında OHK değerlerinin düşük olması istenir. Daha düşük OHK istenilen çıktıdan o kadar daha az sapmayı ifade eder.

6.2 İkili Fonksiyon Doğru Sonuç Yüzdesi (İFDSY)

İkili fonksiyonların sonuçları ya “0” değerini ya da “1” değerini almaktadırlar. Bu çalışma kapsamında kullanılan YSA transfer fonksiyonları $(-1,1)$ aralığında değer

üretebildikleri için EE'lerinin sonuçları da bu aralıkta olmaktadır. Bu aralıktan ikili sisteme dönüştürmek için aşağıdaki formül kullanılmıştır:

$$o'_{ki} = \begin{cases} 1 & o_{ki} \geq 0,5 \\ 0 & \text{diğ.} \end{cases} \quad (6.3)$$

Burada o_{ki} , i 'nci çıktı özelliği için k 'nci gözlemde YSA'nın hesapladığı değer, o'_{ki} ise i 'nci çıktı özelliği için k 'nci gözlemde hesaplanan ikili fonksiyon değeridir.

$$\varsigma_{ki} = \begin{cases} 1 & o'_{ki} = o_{ki} \\ 0 & \text{diğ.} \end{cases} \quad (6.4)$$

$$IFDSY = \frac{100}{mn} \sum_{k=1}^m \sum_{i=1}^n \varsigma_{ki} \quad (6.5)$$

Burada n adet çıktı özelliği ve m adet gözlem için; ς_{ki} , i 'nci çıktı özelliğinin k 'nci gözlemde YSA'da doğru ikili (binary) değeri verip vermediğini gösteren ikili değişken, IFDSY ise ikili fonksiyon doğru sonuç yüzdesidir.

İkili fonksiyon doğru sonuç yüzdesinin daha yüksek olması daha iyi performansı göstermektedir.

6.3 Doğru Sınıflama Yüzdesi (DSY)

Sınıflama probleminde iki sınıf olması durumunda tek EE'lı bir YSA ile sonuçları ya "0" değerini ya da "1" değerini alacak şekilde iki sınıfı birbirinden ayırmak mümkündür. Ancak ikiden fazla sınıf olduğunda sınıf sayısı kadar EE kullanmak gerekmektedir. Bu durumda her bir EE temsil ettikleri küme ile ilgili bir sonuç üreteceklerdir. Üretilen bu sonuçları aşağıdaki formülleri doğru sınıflama yüzdesi (DSY) hesaplamasında kullanabiliriz:

$$\zeta_{ki} = o_{ki} \quad (6.6)$$

$$\beta_k = i | \zeta_{ki} = \max_j \zeta_{kj} \quad (6.7)$$

Burada 0_{ki} , i 'nci çıktı özelliği için k 'ıncı gözlemde YSA'nın hesapladığı değer, β_k , k 'ıncı gözlemde YSA'nın hesapladığı sınıf değeri, ζ_{kj} ise j 'nci sınıf için k 'ıncı gözlemde hesaplanan YSA sonuç değeridir.

$$\mathfrak{S}_k = \begin{cases} 1 & \beta_k^{des.} = \beta_k \\ 0 & \text{diğ.} \end{cases} \quad (6.8)$$

$$DSY = \frac{100}{m} \sum_{k=1}^m \mathfrak{S}_k \quad (6.9)$$

Burada $\mathfrak{S}_k$, k 'ıncı gözlemde YSA'nın doğru sınıflama yapıp yapmadığını gösteren ikili değişken, $\beta_k^{des.}$, k 'ıncı gözlemde istenilen sınıf değeri, DSY doğru sınıflama yüzdesi, m ise toplam sayısıdır.

Doğru sınıflama yüzdesinin daha yüksek olması daha iyi performansı göstermektedir.

6.4 Geri Yayılım Algoritması İterasyon Sayısı

Yapay sinir ağlarının öğrenme süresi model kurmanın ciddi bir bölümünü oluşturmaktadır. Bu sebeple aynı hata performansı için GYA iterasyon sayısının düşük olması istenir.

Bu çalışma kapsamında önerilen algoritmalarda her bir EE için geçen GYA iterasyon sayısı ayrı ayrı hesaplanmış, kıyaslama için ise toplam GYA iterasyon sayısı kullanılmıştır.

6.5 En İyi YSA Yapılarında Kullanılan Yöntemlerin İncelenmesi

Bu çalışma kapsamında önerilen algoritmalar seçim yaparken bir aday havuzu oluşturup bu havuz içerisinden seçilen en iyi yapıyı kullanmaktadırlar. Bu havuzdaki adaylar oluşturulurken mevcut yapıya rassal ağırlıklı yapılar eklendiği gibi mevcut yapının bir kısmını kopyalayan veya değiştiren yöntemlerle de adaylar eklenmektedir. Bu durum hem bitiş elemanları hem de işlem elemanları için geçerlidir.

Aday havuzundan seçilen en iyi adayların hangi yöntemler kullanılarak oluşturulduğu yöntem tekrarlarının grafikleri ile tespit edildi. Bu çalışmada öne çıkan yöntemler ileride yapılacak çalışmalarda daha sık kullanılabilirler.

6.6 Performansların Hesaplanması

Önerilen algoritmaların değerlendirilmesinde Öde görülen her hesaplama 150 defa çalıştırılmış ve algoritmaların elde ettikleri sonuçlar kaydedilmiştir. Daha sonra algoritmaların en iyi parametreleri tespit edilip ve algoritmalar en iyi parametreleri için birbiriyle kıyaslanmıştır. Bu kıyaslamalarda Kayri'nin (2009) önerdiği post hoc testleri doğrultusunda izlenen yöntem Şekil 6.1'de verilen akış şemasında gösterilmiştir.

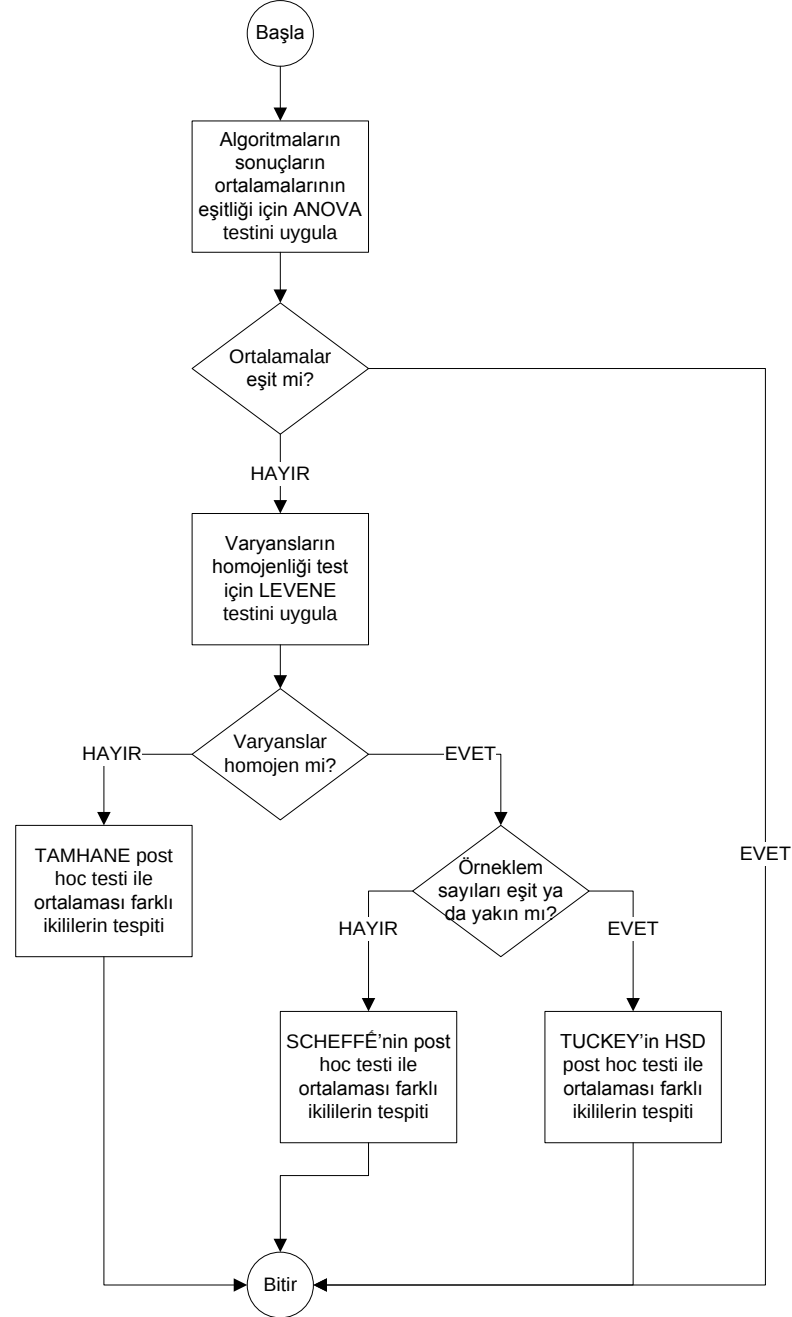
Bu çalışma kapsamında ANOVA ile ortalamaların eşitliği, LEVENE ile varyansların homojenliği, TAMHANE, TUCKEY ve SCHEFFE testleri ile ise ikili olarak ortalamaların eşitliği test edilir. Tüm bu testlerde 0,05 anlamlılık düzeyi kullanılmıştır. Çizelge 6.1'de kullanılan H0 hipotezleri ve anlamlılık değerlerine göre H0 kabul veya red durumları sunulmuştur.

Çizelge 6.1 : İstatistiksel testlerin H0 hipotezleri ve H0 kabul şartları.

Test	H0 Hipotezi	Anlamlılık $\geq 0,05$	Anlamlılık $< 0,05$
ANOVA	Ortalamalar arasında anlamlı bir fark yoktur.	H0 Kabul	H0 Red
LEVENE	Varyanslar homojendir	H0 Kabul	H0 Red
TAMHANE	Ortalamalar arasında anlamlı bir fark yoktur.	H0 Kabul	H0 Red
TUCKEY	Ortalamalar arasında anlamlı bir fark yoktur.	H0 Kabul	H0 Red
SCHEFFE	Ortalamalar arasında anlamlı bir fark yoktur.	H0 Kabul	H0 Red

Tamhane, Tuckey ve Scheffe testlerinde yapılan ikili karşılaştırmalar sonucunda 1. grup ortalaması (I) ile 2. grup ortalaması (J) farkının 2. grup ortalamasına oranı $((I - J) / J)$ karşılaştırma tablolarında verilmiştir. Birinci grup ortalaması ikinci grup ortalamasından büyükse bu oran pozitif, küçükse negatif çıkmaktadır. Ortalama hata

kareleri (OHK) ve toplam iterasyon sayısının (TİS) küçük olması istenirken, doğru sınıflama yüzdesinin (DSY) ve ikili fonksiyon doğru tahmin yüzdesinin (İFDTY) ise yüksek olması istenmektedir. Bu sebeble OHK ve TİS açısından testlerde verilen negatif oran, 1. grubun daha iyi sonuçlar ürettiğini, DSY ve İFDTY açısından ise testlerde pozitif oran 1. grubun daha iyi sonuçlar ürettiğini göstermektedir.



Şekil 6.1 : Algoritmaların kıyaslanmasında kullanılan prosedür.

Çizelge 6.2’de ise bu çalışma kapsamında her bir hesaplama aşamasında yapılan 150’şer tekrarın gerçekleştirildiği koşullar sunulmuştur.

Çizelge 6.2 : Algoritmaların çalıştırılma planı.

Algoritma Adı	Ve/ Veya/ Özel Veya Problemi (İkili Fonksiyon, OHK, İterasyon Sayısı)	Aslanağzı (Iris) Problemi (% Doğru Sınıflama, OHK, İterasyon Sayısı)		Wolf'un Sunspot Problemi (MSE, İterasyon Sayısı)	
TA01V01	Hesaplama 5 $K = 1; L = 1$ PE 2'den 15'e	Hesaplama 1 $K = 10; L = 1, 2, 3, 4, 5$ PE 2'den 15'e	Hesaplama 2 $K = 5; L = 1, 2$ PE 2'den 15'e	Hesaplama 3 $K = 10; L = 1, 2, 3, 4, 5$ PE 2'den 15'e	Hesaplama 4 $K = 5; L = 1, 2$ PE 2'den 15'e
TA02V01	Hesaplama 6 $K = 1; L = 1$	Hesaplama 9 $K = 10; L = 1, 2, 3, 4, 5$	Hesaplama 10 $K = 5; L = 1, 2$	Hesaplama 11 $K = 10; L = 1, 2, 3, 4, 5$	Hesaplama 12 $K = 5; L = 1, 2$
TA03V01	Hesaplama 7 $K = 1; L = 1$	Hesaplama 13 $K = 10; L = 1, 2, 3, 4, 5$	Hesaplama 14 $K = 5; L = 1, 2$	Hesaplama 15 $K = 10; L = 1, 2, 3, 4, 5$	Hesaplama 16 $K = 5; L = 1, 2$
TA03V02	Hesaplama 8 $K = 1; L = 1$	Hesaplama 17 $K = 10; L = 1, 2, 3, 4, 5$	Hesaplama 18 $K = 5; L = 1, 2$	Hesaplama 19 $K = 10; L = 1, 2, 3, 4, 5$	Hesaplama 20 $K = 5; L = 1, 2$

7. UYGULAMA ÖRNEKLERİ

Bu bölümde, önerilen algoritmaların üç temel problem üzerinde uygulamaları yapılmıştır. İlk önce üç temel mantık probleminin birleşiminden oluşan Ve, Veya, Özel Veya birleşik mantık problemi ele alınacaktır. Daha sonra ise Wolf'un Sunspot zaman serisi problemi irdelenecektir. Son olarak da önerilen algoritmalar yazında sık kullanılan aslanağzı (Iris) sınıflama problemine uygulanacaktır. Bu üç problemde önerilen algoritmaların, NeuroSolutions yazılımının ve SPSS Clementine yazılımının elde ettiği sonuçları verilecektir.

Bu çalışmada Wolf'un Sunspot zaman serisi problemi ve Iris problemi verileri aşağıdaki formül ile normalize edilerek kullanılmıştır:

$$t'_{ki} = \frac{t_{ki} - \min_l(t_{li})}{\max_l(t_{li}) - \min_l(t_{li})}, \forall i \quad (7.1)$$

Burada t_{ki} k 'inci gözlemin i 'inci özelliğini, t'_{ki} ise bu değer $[0,1]$ aralığına normalize edilmiş halini göstermektedir.

7.1 Ve, Veya, Özel Veya (And, Or, Xor) Birleşik Mantık Problemi

Bu çalışmada “ve” mantık operatörünün, “veya” mantık operatörünün ve “özel veya” mantık operatörlerinin birleşiminden oluşturulmuş bir problem kullanılmıştır. Bu problemlerden her biri aslında ikili (binary) fonksiyondur; sadece $\{0,1\}$ değerlerini almaktadır. Bu değerler doğruluk tablolarında “0” yanlış, “1” doğruya karşılık gelecek şekilde kullanılmaktadır. Bu tabloların yapay sinir ağlarında kullanılabilmesi için “doğru” ve “yanlış” kavramlarının $\{0,1\}$ değerlerine dönüştürülmesi gerekmektedir.

7.1.1 Problem tanımı

Doğruluk tablosu aşağıda verilen “ve” mantık operatörü her iki koşulun da sağlandığı durumları ifade etmede kullanılır. Örneğin iki kümenin kesişim kümesinde yer alan

elemanları her iki kümeye birden ait olması gerekir, sadece birine üye olması veya ikisine birden üye olmaması durumunda kesişim kümesinde yer almaz.

Çizelge 7.1 : Ve mantık problemi.

p	q	$p \wedge q$
Yanlış	Yanlış	Yanlış
Yanlış	Doğru	Yanlış
Doğru	Yanlış	Yanlış
Doğru	Doğru	Doğru

Doğruluk tablosu aşağıda verilen “veya” mantık operatörü ise herhangi bir koşulun sağlandığı durumları ifade etmede kullanılır. Örnek olarak iki kümenin birleşimi verilebilir. Bir kümeye ait olan veya her iki kümeye birden ait olan elemanlar bileşke kümeye de aittir. Ancak hiçbir kümeye ait olmayan nesneler ise bileşke kümeye de ait değildir.

Çizelge 7.2 : Veya mantık problemi.

p	q	$p \vee q$
Yanlış	Yanlış	Yanlış
Yanlış	Doğru	Yanlış
Doğru	Yanlış	Yanlış
Doğru	Doğru	Doğru

Günlük hayatta “özel veya” mantık operatörü geniş bir kullanım alanı bulmaktadır. Buna örnek olarak restoranların standart menüleri gösterilebilir. Patates kızartması, tavuk kanadı ve ayran veya koladan (içecekten) oluşan menüde içecek olarak ya ayran ya da kola tercih edebilirsiniz. Hem ayran hem de kola tercih etmeniz veya içecek almamanız menüye aykırıdır.

Başka bir örnek olarak misafirlerimize “çay, kahve ne alırsınız?” diye sorarız. Aslında bu soru ile misafirlerimizden çay ve kahveyi aynı anda içemeyecekleri için birini tercih etmelerini isteriz. Ayrıca hiçbirini tercih etmemelerinin ise aslında istemediğimiz ve alınganlık gösterebileceğimiz bir durum olduğunu ima ederiz.

Bilgisayar teknolojisinin çıkması ile özel veya problemi farklı bir boyut kazanmıştır. İkili sistemde sayılar toplanırken oluşan yeni sayının ilgili basamak değerini ifade edecek bir operatörün yokluğu hissedilmiştir. Örneğin işlem yapılan bir basamak için toplananlardan sadece birisi “1” değerini alır diğeri “0” değerini alırsa oluşacak yeni sayının ilgili basamak değeri “1” olur. Eğer işlem yapılan basamak için toplanan

sayıların ilgili basamak değerleri “0” ise sonuç “0” olur. Ancak işlem yapılan basamak için toplanan sayıların her ikisinin de ilgili basamak değeri “1” olursa bu durumda toplamın ilgili basmağı yine “0” değerini alır ve bir üst basamağa taşımak üzere elde oluşur (Çizelge 7.3).

Çizelge 7.3 : Toplama bit işlemi.

Birinci Bit	İkinci Bit	Üçüncü Bit (Toplam)	Elde
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

Yukarıda verilen toplama işleminin (Çizelge 7.3) yapılması için özel veya operatörü kullanılabilir veya bu operatör yerine “ve” operatörü ($\wedge$), “veya” operatörü ($\vee$) v “değil” operatöründen ($\neg$) oluşan aşağıdaki ifade de kullanılabilir;

$$p \oplus q = \neg((p \wedge q) \vee (\neg p \wedge \neg q)) \quad (7.2)$$

Özel veya’nın yapılması gereken ayrık olayları ifade ettiğini düşünebiliriz. Örneğin yapılması gereken iki farklı iş çizelgeleme yapılırken aynı anda aynı makinada yapılamaz. İşlerin yapılması gerektiğini de dikkate alırsak bu durumu ifade eden mantık operatörü “özel veya” mantık operatörüdür.

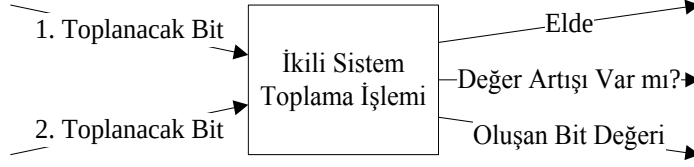
Doğruluk tablosu aşağıda verilen “özel veya” mantık operatörü “ $\oplus$ ” sembolü ile gösterilir ve sadece bir koşulun sağlandığı durumları ifade etmede kullanılır. Örnek olarak iki kümeden sadece birine ait olan nesneleri ifade etmede kullanılabilir.

Çizelge 7.4 : Özel Veya mantık problemi.

p	q	$p \oplus q$
Yanlış	Yanlış	Yanlış
Yanlış	Doğru	Doğru
Doğru	Yanlış	Doğru
Doğru	Doğru	Yanlış

Bu çalışmada yukarıda açıklaması verilen mantık problemlerinin birleşimiyle yeni bir problem oluşturulmuştur. Örnek olarak ikili sistemde toplama işlemi ele alınmıştır. İkili sistemde sayıların tüm basamakları ya “0” ya da “1” değerini almaktadır. Bu durumda ikili sistemde, iki sayı toplanırken herhangi bir basamak için

toplama işlemi sonucunda artık değer olup olmadığı “ve” operatörü ile, basamakta değer artışı olup olmadığı “veya” operatörü ile sonuç bit değeri, “özel veya” operatörü ile gösterir. Oluşturulan sistemin karakutu gösterimi aşağıda verilmiştir.



Şekil 7.1 : Bit toplama işlemi gösterimi.

Açıklanan bu işlem bütün sayısal işlemlerin temelinde yer alması nedeniyle özellikle elektronik uygulamalarında önemlidir.

Çizelge 7.5 : Bit toplama işlemi.

G1: 1. Toplananın İlgili Basamak Bit Değeri	G2: 2. Toplananın İlgili Basamak Bit Değeri	Ç1: Toplama işlemi sırasında elde var mı? (Ve)	Ç2: Değer artışı var mı? (Veya)	Ç3: Sonuç Bit Değeri (Özel Veya)
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

7.1.2 Uygulanan işlemler

Ve, veya, özel veya ikili fonksiyon probleminde elde edilen sonuçların karşılaştırılması için yapılan ANOVA testi Çizelge 7.6’da sunulmuştur. Çizelge incelendiğinde ikili fonksiyon doğru sinyal yüzdesi, ortalama hata karesi ve toplam iterasyon sayısı bakımından algoritmalar arasında istatistiksel olarak anlamlı farklar olduğu görülmektedir.

Çizelge 7.6 : ANOVA testi sonuçları (ve, veya, özel veya).

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
İkili fonksiyon doğru sinyal yüzdesi	Gruplar arası	153319,8	33	4646,1	231,85	0,00000
	Grup içi	101359,2	5058	20,0		
	Toplam	254679,1	5091			
Ortalama Hata Karesi	Gruplar arası	9,0	33	0,3	312,14	0,00000
	Grup içi	4,4	5059	0,0		
	Toplam	13,5	5092			
Toplam iterasyon sayısı	Gruplar arası	18765991,4	33	568666,4	1247,61	0,00000
	Grup içi	2305916,6	5059	455,8		
	Toplam	21071908,0	5092			

Post hoc testlerinden hangilerini yapılması gerektiğine karar verebilmek için öncelikle Levene testi yapılması gerekmektedir. Levene testi ile varyansların homojenliği testi sonuçları Çizelge 7.7’de verilmiştir. Çizelge incelendiğinde ikili fonksiyon doğru sinyal yüzdesi, ortalama hata karesi ve toplam iterasyon sayısı bakımından varyansların homojen olmadığı gözlenmiştir.

Çizelge 7.7 : Varyansların homojenliği (Levene) testi sonuçları (ve/veya/özel veya).

Değerlendirme Türü	Levene İstatistiği	sd1	sd2	Anlamlılık
İkili fonksiyon doğru sinyal yüzdesi	25,29	33	5058	0,00000
Ortalama Hata Karesi	123,93	33	5059	0,00000
Toplam iterasyon sayısı	39,19	33	5059	0,00000

Her üç performans kriteri için de varyanslar homojen olmadığından Tamhane Post Hoc testleri yapılmıştır.

7.1.3 TA01V01 sonuçları

TA01V01 algoritması ikili fonksiyon doğru yüzdesinin Tamhane Post Hoc testi sonuçları 2 PE, 5 PE ve 10 PE için sırasıyla Çizelge 7.8’de, Çizelge 7.9’da ve Çizelge 7.10’da verilmiştir.

Çizelge 7.8 : İkili fonksiyon doğru yüzdesinin TA01V01 2 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA01V01_2PE	TA01V01_5PE	-0,07	0,60	0,00000
TA01V01_2PE	TA01V01_10PE	-0,06	0,61	0,00000
TA01V01_2PE	TA03V02	-0,04	0,63	0,00000

Çizelge 7.8’de verilen sonuçlar incelendiğinde 2 PE’lı TA01V01 algoritmasının ikili fonksiyon doğru sinyal yüzdesinin 5 PE’lı TA01V01, 10 PE’lı TA01V01 ve TA03V02 algoritmalarının ikili fonksiyon doğru sinyal yüzdelerinden daha düşük olduğu görülmektedir.

Çizelge 7.9 : İkili fonksiyon doğru yüzdesinin TA01V01 5 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA01V01_5PE	TA01V01_2PE	0,07	0,60	0,00000
TA01V01_5PE	TA02V01	0,05	0,41	0,00000
TA01V01_5PE	TA03V01	0,07	0,32	0,00000
TA01V01_5PE	TA03V02	0,02	0,45	0,00000

Çizelge 7.9’da verilen sonuçlar incelendiğinde 5 PE’lı TA01V01 algoritmasının ikili fonksiyon doğru sinyal yüzdesinin 2 PE’lı TA01V01, TA02V01, TA03V01 ve TA03V02 algoritmalarının ikili fonksiyon doğru sinyal yüzdelerinden daha yüksek olduğu görülmektedir.

Çizelge 7.10 : İkili fonksiyon doğru yüzdesinin TA01V01 10 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA01V01_10PE	TA01V01_2PE	0,07	0,61	0,00000
TA01V01_10PE	TA02V01	0,04	0,42	0,00000
TA01V01_10PE	TA03V01	0,07	0,34	0,00000
TA01V01_10PE	TA03V02	0,02	0,46	0,02900

Çizelge 7.10’da verilen sonuçlar incelendiğinde 10 PE’lı TA01V01 algoritmasının ikili fonksiyon doğru sinyal yüzdesinin 2 PE’lı TA01V01, TA02V01, TA03V01 ve TA03V02 algoritmalarının ikili fonksiyon doğru sinyal yüzdelerinden daha yüksek olduğu görülmektedir.

TA01V01 algoritması ortalama hata karelerinin Tamhane Post Hoc testi sonuçları 2 PE, 5 PE ve 10 PE için sırasıyla Çizelge 7.11’de, Çizelge 7.12’de ve Çizelge 7.13’te verilmiştir.

Çizelge 7.11 : OHK’nin TA01V01 2 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA01V01_2PE	TA01V01_5PE	1,91	0,00	0,00000
TA01V01_2PE	TA01V01_10PE	1,39	0,00	0,00000
TA01V01_2PE	TA03V01	-0,46	0,00	0,00000

Çizelge 7.11’de verilen sonuçlar incelendiğinde 2 PE’lı TA01V01 algoritmasının ortalama hata karesinin TA03V01 algoritmasının ortalama hata karelerinden daha düşük olduğu görülmektedir.

Çizelge 7.12 : OHK’nin TA01V01 5 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA01V01_5PE	TA01V01_2PE	-0,66	0,00	0,00000
TA01V01_5PE	TA02V01	-0,69	0,00	0,00000
TA01V01_5PE	TA03V01	-0,81	0,00	0,00000
TA01V01_5PE	TA03V02	-0,53	0,00	0,00000

Çizelge 7.12’de verilen sonuçlar incelendiğinde 5 PE’lı TA01V01 algoritmasının ortalama hata karesinin 2 PE’lı TA01V01, TA02V01, TA03V01 ve TA03V02 algoritmalarının ortalama hata karelerinden daha düşük olduğu görülmektedir.

Çizelge 7.13 : OHK’nin TA01V01 10 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA01V01_10PE	TA01V01_2PE	-0,58	0,00	0,00000
TA01V01_10PE	TA02V01	-0,62	0,00	0,00000
TA01V01_10PE	TA03V01	-0,77	0,00	0,00000
TA01V01_10PE	TA03V02	-0,43	0,00	0,02300

Çizelge 7.13’te verilen sonuçlar incelendiğinde 10 PE’lı TA01V01 algoritmasının ortalama hata karesinin 2 PE’lı TA01V01, TA02V01, TA03V01 ve TA03V02 algoritmalarının ortalama hata karelerinden daha düşük olduğu görülmektedir.

TA01V01 algoritması toplam iterasyon sayılarının Tamhane Post Hoc testi sonuçları 2 PE, 5 PE ve 10 PE için sırasıyla Çizelge 7.14'te, Çizelge 7.15'te ve Çizelge 7.16'da verilmiştir.

Çizelge 7.14 : Toplam iterasyon sayısının TA01V01 2 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA01V01_2PE	TA01V01_10PE	-0,06	2,97	0,04500
TA01V01_2PE	TA03V01	0,51	2,44	0,00000
TA01V01_2PE	TA03V02	-0,17	3,79	0,00000

Çizelge 7.14'te verilen sonuçlar incelendiğinde 2 PE'lı TA01V01 algoritmasının GYA iterasyon sayısının 10 PE'lı TA01V01 ve TA03V02 algoritmalarının GYA iterasyon sayılarının daha düşük olduğu görülmektedir.

Çizelge 7.15 : Toplam iterasyon sayısının TA01V01 5 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA01V01_5PE	TA03V01	0,56	2,80	0,00000
TA01V01_5PE	TA03V02	-0,14	4,03	0,00000

Çizelge 7.15'te verilen sonuçlar incelendiğinde 5 PE'lı TA01V01 algoritmasının GYA iterasyon sayısının TA03V02 algoritmasının GYA iterasyon sayılarının daha düşük olduğu görülmektedir.

Çizelge 7.16 : Toplam iterasyon sayısının TA01V01 10 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA01V01_10PE	TA01V01_2PE	0,07	2,97	0,04500
TA01V01_10PE	TA02V01	0,12	3,51	0,00000
TA01V01_10PE	TA03V01	0,61	2,34	0,00000
TA01V01_10PE	TA03V02	-0,11	3,73	0,00000

Çizelge 7.16'da verilen sonuçlar incelendiğinde 10 PE'lı TA01V01 algoritmasının GYA iterasyon sayısının TA03V02 algoritmasının GYA iterasyon sayılarının daha düşük olduğu görülmektedir.

7.1.4 TA02V01 sonuçları

TA02V01 algoritması toplam ikili fonksiyon doğru yüzdesinin, ortalama hata karesinin ve iterasyon sayılarının Tamhane Post Hoc testi sonuçları sırasıyla Çizelge 7.17’de, Çizelge 7.18’de ve Çizelge 7.19’da verilmiştir.

Çizelge 7.17 : İkili fonksiyon doğru yüzdesinin TA02V01 için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA02V01	TA01V01_5PE	-0,05	0,41	0,00000
TA02V01	TA01V01_10PE	-0,04	0,42	0,00000
TA02V01	TA03V01	0,02	0,32	0,00000
TA02V01	TA03V02	-0,02	0,45	0,00100

Çizelge 7.17’de verilen sonuçlar incelendiğinde TA02V01 algoritmasının ikili fonksiyon doğru sinyal yüzdesinin TA03V01 algoritmasının ikili fonksiyon doğru sinyal yüzdelерinden daha yüksek olduğu görülmektedir.

Çizelge 7.18 : OHK’nin TA02V01 için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA02V01	TA01V01_5PE	2,21	0,00	0,00000
TA02V01	TA01V01_10PE	1,63	0,00	0,00000
TA02V01	TA03V01	-0,40	0,00	0,00000
TA02V01	TA03V02	0,50	0,00	0,00100

Çizelge 7.18’de verilen sonuçlar incelendiğinde TA02V01 algoritmasının ortalama hata karesinin TA03V01 algoritmasının ortalama hata karelerinden daha düşük olduğu görülmektedir.

Çizelge 7.19 : Toplam iterasyon sayısının TA02V01 için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA02V01	TA01V01_2PE	-0,04	3,57	1,00000
TA02V01	TA01V01_5PE	-0,07	3,83	0,29800
TA02V01	TA01V01_10PE	-0,10	3,51	0,00000
TA02V01	TA03V01	0,45	3,07	0,00000
TA02V01	TA03V02	-0,20	4,22	0,00000

Çizelge 7.19’da verilen sonuçlar incelendiğinde TA02V01 algoritmasının GYA iterasyon sayısının 10 PE’lı TA01V01 ve TA03V02 algoritmalarının GYA iterasyon sayılarının daha düşük olduğu görülmektedir.

7.1.5 TA03V01 sonuçları

TA03V01 algoritması toplam ikili fonksiyon doğru yüzdesinin, ortalama hata karesinin ve iterasyon sayılarının Tamhane Post Hoc testi sonuçları sırasıyla Çizelge 7.20’de, Çizelge 7.21’de ve Çizelge 7.22’de verilmiştir.

Çizelge 7.20 : İkili fonksiyon doğru yüzdesinin TA03V01 için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA03V01	TA01V01_5PE	-0,07	0,32	0,00000
TA03V01	TA01V01_10PE	-0,06	0,34	0,00000
TA03V01	TA02V01	-0,02	0,32	0,00000
TA03V01	TA03V02	-0,04	0,37	0,00000

Çizelge 7.20’de verilen sonuçlar incelendiğinde TA03V01 algoritmasının ikili fonksiyon doğru sinyal yüzdesinin 5 PE’lı TA01V01, 10 PE’lı TA01V01, TA02V01 ve TA03V02 algoritmalarının ikili fonksiyon doğru sinyal yüzdelерinden daha düşük olduğu görülmektedir.

Çizelge 7.21 : OHK’nin TA03V01 için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA03V01	TA01V01_2PE	0,84	0,00	0,00000
TA03V01	TA01V01_5PE	4,36	0,00	0,00000
TA03V01	TA01V01_10PE	3,40	0,00	0,00000
TA03V01	TA02V01	0,67	0,00	0,00000
TA03V01	TA03V02	1,50	0,00	0,00000

Çizelge 7.21’de verilen sonuçlar incelendiğinde TA03V01 algoritmasının ortalama hata karesinin 2 PE’lı TA01V01, 5 PE’lı TA01V01, 10 PE’lı TA01V01, TA02V01 ve TA03V02 algoritmalarının ortalama hata karelerinden daha yüksek olduğu görülmektedir.

Çizelge 7.22 : Toplam iterasyon sayısının TA03V01 için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA03V01	TA01V01_2PE	-0,34	2,44	0,00000
TA03V01	TA01V01_5PE	-0,36	2,80	0,00000
TA03V01	TA01V01_10PE	-0,38	2,34	0,00000
TA03V01	TA02V01	-0,31	3,07	0,00000
TA03V01	TA03V02	-0,45	3,32	0,00000

Çizelge 7.22’de verilen sonuçlar incelendiğinde TA03V01 algoritmasının GYA iterasyon sayısının 2 PE’lı TA01V01, 5 PE’lı TA01V01, 10 PE’lı TA01V01, TA02V01 ve TA03V02 algoritmalarının GYA iterasyon sayılarının daha düşük olduğu görülmektedir.

7.1.6 TA03V02 sonuçları

TA03V02 algoritması toplam ikili fonksiyon doğru yüzdesinin, ortalama hata karesinin ve iterasyon sayılarının Tamhane Post Hoc testi sonuçları sırasıyla Çizelge 7.23’te, Çizelge 7.24’te ve Çizelge 7.25’te verilmiştir.

Çizelge 7.23 : İkili fonksiyon doğru yüzdesinin TA03V02 için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA03V02	TA01V01_2PE	0,05	0,63	0,00000
TA03V02	TA01V01_5PE	-0,02	0,45	0,00000
TA03V02	TA01V01_10PE	-0,02	0,46	0,02900
TA03V02	TA02V01	0,02	0,45	0,00100
TA03V02	TA03V01	0,05	0,37	0,00000

Çizelge 7.23’te verilen sonuçlar incelendiğinde TA03V02 algoritmasının ikili fonksiyon doğru sinyal yüzdesinin 2 PE’lı TA01V01, TA02V01 ve TA03V01 algoritmalarının ikili fonksiyon doğru sinyal yüzdelерinden daha yüksek olduğu görülmektedir.

Çizelge 7.24 : OHK'nin TA03V02 için Tamhane Post Hoc testi ile analizi.

(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA03V02	TA01V01_5PE	1,14	0,00	0,00000
TA03V02	TA01V01_10PE	0,76	0,00	0,02300
TA03V02	TA02V01	-0,33	0,00	0,00100
TA03V02	TA03V01	-0,60	0,00	0,00000

Çizelge 7.24'te verilen sonuçlar incelendiğinde TA03V02 algoritmasının ortalama hata karesinin TA02V01 ve TA03V01 algoritmalarının ortalama hata karelerinden daha düşük olduğu görülmektedir.

Çizelge 7.25 : Toplam iterasyon sayısının TA03V02 için Tamhane Post Hoc testi ile analizi.

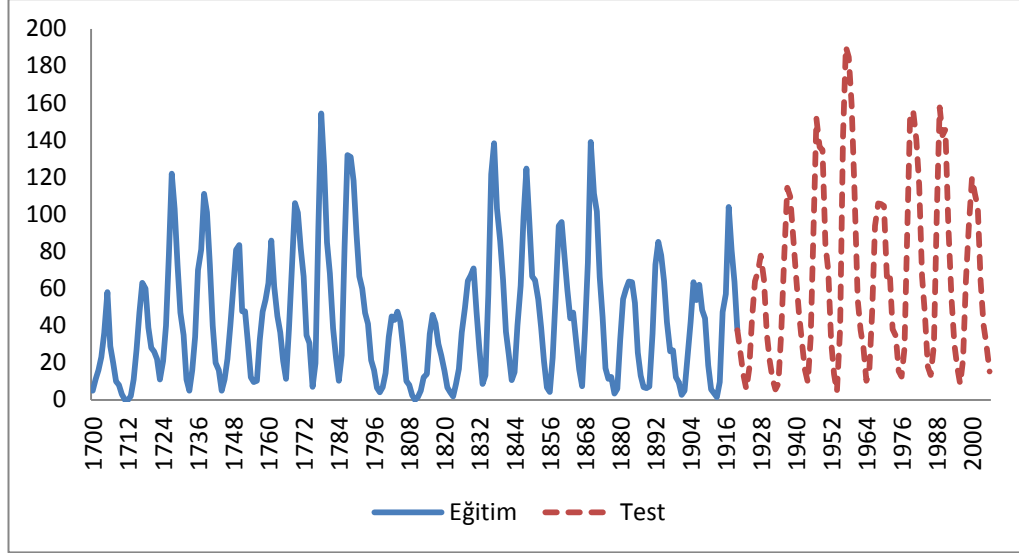
(I) Deneme	(J) Deneme	Ortalama Fark Oranı (I-J)/J	Standart Hata	Anlamlılık
TA03V02	TA01V01_2PE	0,20	3,79	0,00000
TA03V02	TA01V01_5PE	0,16	4,03	0,00000
TA03V02	TA01V01_10PE	0,12	3,73	0,00000
TA03V02	TA02V01	0,25	4,22	0,00000
TA03V02	TA03V01	0,81	3,32	0,00000

Çizelge 7.25'te verilen sonuçlar incelendiğinde TA03V02 algoritmasının GYA iterasyon sayısının 2 PE'lı TA01V01, 5 PE'lı TA01V01, 10 PE'lı TA01V01, TA02V01 ve TA03V01 algoritmalarının GYA iterasyon sayılarının çok daha fazla olduğu görülmektedir.

7.2 Wolf'un SUNSPOT Fonksiyon Tahmini Problemi

7.2.1 Problem tanımı

Wolf'un SunSpot zaman serisi 1700 yılından günümüze kadar Güneş'in yüzünde oluşan gözle görülür lekelerin yıllık kayıtlarıdır. Veri kümesinin doğrusal olmaması ve Gauss olmaması (non-linear, non-Gaussian) sebebiyle bir çok araştırmacı tarafından kıyaslama verisi olarak kullanılmasına neden olmuştur. Bu çalışmada kullanılan veriler eğitim ve test için kullanılan kısımlarıyla Şekil 7.2'de sunulmuştur.



Şekil 7.2 : SunSpot veri kümesi.

Genellikle 1700-1920 (221 gözlem) yılları arası eğitim için, 1921-1955 (35 gözlem) yılları arası ise tahmin için kullanılır (Cottrell ve diğ., 1995; DeGroot ve Wurtz, 1991; Zhang, 2003). SunSpot Zaman Serisi kullanılırken t yılını tahmin etmek için önceki dört yılın ($t-1$, $t-2$, $t-3$, $t-4$) verileri kullanılır. SunSpot veri kümesinin tanımlayıcı istatistikleri Çizelge 7.26’da sunulmuştur.

Çizelge 7.26 : SunSpot veri kümesi tanımlayıcı istatistikleri.

Aralık	Minimum	Maksimum	Ortalama	Varyans
190.2	0	190.2	50.04235	1634.015717

7.2.2 Uygulanan işlemler

Wolf’un Sunspot fonksiyon tahmin probleminde elde edilen sonuçların karşılaştırılması için öncelikle tüm algoritmaların en uygun parametrelerinin tespit edilmesi gerekmektedir. Bu amaçla öncelikle her algoritmanın Çizelge 7.27’de verilen yedi farklı çapraz geçerlilik değeri kıyaslanmıştır.

Çizelge 7.27 : Wolf’un Sunspot problemi için önerilen algoritmalarda kullanılan farklı çapraz geçerlilik değerleri.

Sıra No	K	L
1	10	1
2	10	2
3	10	3
4	10	4
5	10	5
6	5	1
7	5	2

7.2.3 TA01V01 sonuçları

Çizelge 7.28 : TA01V01'in 2 PE'li farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	3,89	0,00075
	Grup içi	0,01	1042	0,00		
	Toplam	0,01	1048			
Toplam iterasyon sayısı	Gruplar arası	3847507,06	6	641251,18	13,81	0,00000
	Grup içi	48377849,77	1042	46427,88		
	Toplam	52225356,83	1048			

Çizelge 7.28, TA01V01'in farklı çarpaz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde OHK bakımından ve toplam iterasyon sayısı bakımından istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.29'da Levene testi yapılmış ve varyansların homojen olmadığı görülmüştür. Bunun üzerine Çizelge 7.30'da ve Çizelge 7.31'de görülen Tamhane testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.29 : TA01V01'in 2 PE'li farklı ÇG değerleri için Levene testi sonuçları..

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Ortalama Hata Karesi	2,71	6	1042	0,01297
Toplam İterasyon Sayısı	60,96	6	1042	0,00000

Çizelge 7.30 : OHK bakımından TA01V01'in farklı ÇG değerlerinin 2 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama Fark Oranı (I-J)/J	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG			
TA01V01	10'un 1'li	TA01V01	5'in 2'li	-0,09	0,00	0,00200
TA01V01	10'un 2'li	TA01V01	5'in 2'li	-0,09	0,00	0,00300
TA01V01	5'in 2'li	TA01V01	10'un 1'li	0,10	0,00	0,00200
TA01V01	5'in 2'li	TA01V01	10'un 2'li	0,10	0,00	0,00300

Çizelge 7.30'da görülen TA01V01 2 PE için Tamhane testi sonuçları incelendiğinde TA01V01 algoritması için 10'un 1'li ve 10'un 2'li çarpaz geçerliliği 5'in 2'li çarpaz geçerliliğinden daha düşük OHK değerlerine sahip sonuçlar üretmektedirler.

Çizelge 7.31 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 2 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA01V01	10'un 1'li	TA01V01	10'un 4'lü	0,10	21,04	0,00100
TA01V01	10'un 1'li	TA01V01	10'un 5'li	0,07	16,76	0,00300
TA01V01	10'un 1'li	TA01V01	5'in 1'li	0,10	20,56	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 2'li	0,24	29,03	0,00000
TA01V01	10'un 2'li	TA01V01	10'un 4'lü	0,09	22,31	0,01300
TA01V01	10'un 2'li	TA01V01	5'in 1'li	0,09	21,86	0,00900
TA01V01	10'un 2'li	TA01V01	5'in 2'li	0,22	29,97	0,00000
TA01V01	10'un 3'lü	TA01V01	5'in 2'li	0,20	31,13	0,00000
TA01V01	10'un 4'lü	TA01V01	10'un 1'li	-0,09	21,04	0,00100
TA01V01	10'un 4'lü	TA01V01	10'un 2'li	-0,08	22,31	0,01300
TA01V01	10'un 5'li	TA01V01	10'un 1'li	-0,07	16,76	0,00300
TA01V01	10'un 5'li	TA01V01	5'in 2'li	0,16	33,52	0,00500
TA01V01	5'in 1'li	TA01V01	10'un 1'li	-0,09	20,56	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 2'li	-0,08	21,86	0,00900
TA01V01	5'in 2'li	TA01V01	10'un 1'li	-0,19	29,03	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 2'li	-0,18	29,97	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 3'lü	-0,17	31,13	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 5'li	-0,13	33,52	0,00500

Çizelge 7.31'de görülen TA01V01 2 PE için Tamhane testi sonuçları incelendiğinde 10'un 4'lü çapraz geçerliliğinin 10'un 1'li ve 10'un 2'li çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir. 10'un 5'li çapraz geçerliliğinin de 10'un 1'li geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

5'in 1'li ve 5'in 2'li çapraz geçerliliği, 10'un 1'li ve 10'un 2'li çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir. 5'in 2'li çapraz geçerliliğinin ayrıca 10'un 3'lü ve 10'un 5'li çapraz geçerliliğinden de daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

Çizelge 7.32 : TA01V01'in 5 PE'li farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	7,87	0,00000
	Grup içi	0,00	1040	0,00		
	Toplam	0,00	1046			
Toplam İterasyon Sayısı	Gruplar arası	13246185,20	6	2207697,53	32,51	0,00000
	Grup içi	70629976,54	1040	67913,44		
	Toplam	83876161,74	1046			

Çizelge 7.32 TA01V01'in farklı çarpaz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde ortalama hata karesi ve toplam iterasyon sayısı bakımından istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.33'te Levene testi yapılmış ve varyansların homojen olmadığı görülmüştür. Bunun üzerine Çizelge 7.34'te ve Çizelge 7.35'te görülen Tamhane testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.33 : TA01V01'in 5 PE'li farklı ÇG değerleri için Levene testi sonuçları.

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Ortalama Hata Karesi	9,38	6	1040	0,00000
Toplam İterasyon Sayısı	112,07	6	1040	0,00000

Çizelge 7.34 : OHK bakımından TA01V01'in farklı ÇG değerlerinin 5 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama Fark Oranı (I-J)/J	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG			
TA01V01	10'un 1'li	TA01V01	10'un 4'lü	-0,06	0,00	0,00500
TA01V01	10'un 1'li	TA01V01	10'un 5'li	-0,05	0,00	0,05000
TA01V01	10'un 1'li	TA01V01	5'in 2'li	-0,11	0,00	0,00000
TA01V01	10'un 2'li	TA01V01	5'in 2'li	-0,09	0,00	0,00100
TA01V01	10'un 3'lü	TA01V01	5'in 2'li	-0,08	0,00	0,00200
TA01V01	10'un 4'lü	TA01V01	10'un 1'li	0,07	0,00	0,00500
TA01V01	10'un 5'li	TA01V01	10'un 1'li	0,06	0,00	0,05000
TA01V01	5'in 1'li	TA01V01	5'in 2'li	-0,07	0,00	0,02200
TA01V01	5'in 2'li	TA01V01	10'un 1'li	0,13	0,00	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 2'li	0,09	0,00	0,00100
TA01V01	5'in 2'li	TA01V01	10'un 3'lü	0,09	0,00	0,00200
TA01V01	5'in 2'li	TA01V01	5'in 1'li	0,07	0,00	0,02200

Çizelge 7.34'te görülen TA01V01 5 PE için Tamhane testi sonuçları incelendiğinde 10'un 1'li çapraz geçerliliğinin, 10'un 4'lü ve 10'un 5'li çapraz geçerliliğinden daha az ortalama hata karesine sahip olduğunu görülmektedir. Ayrıca 10'un 1'li ve 10'un 2'li, 10'un 3'lü ve 5'in 1'li çapraz geçerliliğinin de 5'in 2'li çapraz geçerliliğinden daha az ortalama hata karesine sahip olduğunu görülmektedir.

Çizelge 7.35 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 5 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama Fark Oranı (I-J)/J	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG			
TA01V01	10'un 1'li	TA01V01	10'un 3'lü	0,05	15,12	0,03400
TA01V01	10'un 1'li	TA01V01	10'un 4'lü	0,14	23,39	0,00000
TA01V01	10'un 1'li	TA01V01	10'un 5'li	0,16	24,76	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 1'li	0,10	20,82	0,00100
TA01V01	10'un 1'li	TA01V01	5'in 2'li	0,57	35,42	0,00000
TA01V01	10'un 2'li	TA01V01	10'un 4'lü	0,12	25,56	0,00200
TA01V01	10'un 2'li	TA01V01	10'un 5'li	0,14	26,82	0,00000
TA01V01	10'un 2'li	TA01V01	5'in 2'li	0,54	36,89	0,00000
TA01V01	10'un 3'lü	TA01V01	10'un 1'li	-0,05	15,12	0,03400
TA01V01	10'un 3'lü	TA01V01	10'un 5'li	0,11	29,01	0,03100
TA01V01	10'un 3'lü	TA01V01	5'in 2'li	0,49	38,51	0,00000
TA01V01	10'un 4'lü	TA01V01	10'un 1'li	-0,12	23,39	0,00000
TA01V01	10'un 4'lü	TA01V01	10'un 2'li	-0,10	25,56	0,00200
TA01V01	10'un 4'lü	TA01V01	5'in 2'li	0,37	42,44	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 1'li	-0,14	24,76	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 2'li	-0,12	26,82	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 3'lü	-0,10	29,01	0,03100
TA01V01	10'un 5'li	TA01V01	5'in 2'li	0,34	43,21	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 1'li	-0,09	20,82	0,00100
TA01V01	5'in 1'li	TA01V01	5'in 2'li	0,43	41,08	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 1'li	-0,36	35,42	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 2'li	-0,35	36,89	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 3'lü	-0,33	38,51	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 4'lü	-0,27	42,44	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 5'li	-0,26	43,21	0,00000
TA01V01	5'in 2'li	TA01V01	5'in 1'li	-0,30	41,08	0,00000

Çizelge 7.35'te görülen TA01V01 5 PE için Tamhane testi sonuçları incelendiğinde 10'un 3'lü ve 5'in 1'li çapraz geçerliliğinin 10'un 1'li çapraz geçerliliğinden, 10'un 4'lü çapraz geçerliliğinin ise 10'un 1'li çapraz geçerliliğinden ve 10'un 2'li çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir. 10'un 5'li çapraz geçerliliğinin de 10'un 1'li, 10'un 2'li ve 10'un 3'lü çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

5'in 2'li çapraz geçerliliğinin, 10'un 1'li, 10'un 2'li, 10'un 3'lü, 10'un 4'lü ve 5'in 1'li olmak üzere tüm diğer çapraz geçerlilik değerlerinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

Çizelge 7.36 : TA01V01'in 10 PE'li farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	7,61	0,00000
	Grup içi	0,00	1039	0,00		
	Toplam	0,00	1045			
Toplam İterasyon Sayısı	Gruplar arası	12557495	6	2092915,94	23,85	0,00000
	Grup içi	91171154	1039	87748,95		
	Toplam	103728650	1045			

Çizelge 7.36 TA01V01'in farklı çapraz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde ortalama hata karesi ve toplam iterasyon sayısı bakımından istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.37'de Levene testi yapılmış ve varyansların homojen olmadığı görülmüştür. Bunun üzerine Çizelge 7.38'de ve Çizelge 7.39'da görülen Tamhane testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.37 : TA01V01'in 10 PE'li farklı ÇG değerleri için Levene testi sonuçları.

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Ortalama Hata Karesi	10,57	6	1039	0,00000
Toplam İterasyon Sayısı	107,73	6	1039	0,00000

Çizelge 7.38'de görülen TA01V01 10 PE için Tamhane testi sonuçları incelendiğinde 10'un 1'li, 10'un 2'li ve 10'un 3'lü çapraz geçerliliğinin, 10'un 5'li ve 5'in 2'li çapraz geçerliliğinden daha az ortalama hata karesine sahip olduğunu görülmektedir. Ayrıca 10'un 1'li çapraz geçerliliğinin 5'in 1'li çapraz geçerliliğinden de daha az ortalama hata karesine sahip olduğunu görülmektedir.

Çizelge 7.38 : OHK bakımından TA01V01'in farklı ÇG değerlerinin 10 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA01V01	10'un 1'li	TA01V01	10'un 5'li	-0,07	0,00	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 1'li	-0,04	0,00	0,04800
TA01V01	10'un 1'li	TA01V01	5'in 2'li	-0,08	0,00	0,00000
TA01V01	10'un 2'li	TA01V01	10'un 5'li	-0,06	0,00	0,00200
TA01V01	10'un 2'li	TA01V01	5'in 2'li	-0,07	0,00	0,00100
TA01V01	10'un 3'lü	TA01V01	10'un 5'li	-0,06	0,00	0,01500
TA01V01	10'un 3'lü	TA01V01	5'in 2'li	-0,07	0,00	0,00400
TA01V01	10'un 5'li	TA01V01	10'un 1'li	0,08	0,00	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 2'li	0,07	0,00	0,00200
TA01V01	10'un 5'li	TA01V01	10'un 3'lü	0,06	0,00	0,01500
TA01V01	5'in 1'li	TA01V01	10'un 1'li	0,05	0,00	0,04800
TA01V01	5'in 2'li	TA01V01	10'un 1'li	0,09	0,00	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 2'li	0,08	0,00	0,00100
TA01V01	5'in 2'li	TA01V01	10'un 3'lü	0,07	0,00	0,00400

Çizelge 7.39 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 10 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA01V01	10'un 1'li	TA01V01	10'un 3'lü	0,08	19,82	0,00400
TA01V01	10'un 1'li	TA01V01	10'un 4'lü	0,23	29,44	0,00000
TA01V01	10'un 1'li	TA01V01	10'un 5'li	0,26	30,20	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 1'li	0,15	24,50	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 2'li	0,52	34,62	0,00000
TA01V01	10'un 2'li	TA01V01	10'un 4'lü	0,20	31,88	0,00000
TA01V01	10'un 2'li	TA01V01	10'un 5'li	0,23	32,59	0,00000
TA01V01	10'un 2'li	TA01V01	5'in 1'li	0,11	27,38	0,00700
TA01V01	10'un 2'li	TA01V01	5'in 2'li	0,48	36,72	0,00000
TA01V01	10'un 3'lü	TA01V01	10'un 1'li	-0,08	19,82	0,00400
TA01V01	10'un 3'lü	TA01V01	10'un 4'lü	0,14	35,49	0,03000
TA01V01	10'un 3'lü	TA01V01	10'un 5'li	0,16	36,13	0,00700
TA01V01	10'un 3'lü	TA01V01	5'in 2'li	0,40	39,89	0,00000
TA01V01	10'un 4'lü	TA01V01	10'un 1'li	-0,19	29,44	0,00000
TA01V01	10'un 4'lü	TA01V01	10'un 2'li	-0,17	31,88	0,00000
TA01V01	10'un 4'lü	TA01V01	10'un 3'lü	-0,12	35,49	0,03000
TA01V01	10'un 4'lü	TA01V01	5'in 2'li	0,23	45,44	0,02000
TA01V01	10'un 5'li	TA01V01	10'un 1'li	-0,21	30,20	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 2'li	-0,18	32,59	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 3'lü	-0,14	36,13	0,00700
TA01V01	5'in 1'li	TA01V01	10'un 1'li	-0,13	24,50	0,00000

Çizelge 7.39 (devam): Toplam iterasyon sayısı bakımından TA01V01’in farklı ÇG değerlerinin 10 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA01V01	5'in 1'li	TA01V01	10'un 2'li	-0,10	27,38	0,00700
TA01V01	5'in 1'li	TA01V01	5'in 2'li	0,33	42,41	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 1'li	-0,34	34,62	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 2'li	-0,32	36,72	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 3'lü	-0,29	39,89	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 4'lü	-0,19	45,44	0,02000
TA01V01	5'in 2'li	TA01V01	5'in 1'li	-0,25	42,41	0,00000

Çizelge 7.39’da görülen TA01V01 10 PE için Tamhane testi sonuçları incelendiğinde 10’un 3’lü çapraz geçerliliğinin 10’un 1’li çapraz geçerliliğinden, 10’un 4’lü ve 10’un 5’li çapraz geçerliliğinin ise 10’un 1’li çapraz geçerliliğinden, 10’un 2’li çapraz geçerliliğinden ve 10’un 3’lü çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

5’in 1’li çapraz geçerliliğinin, 10’un 1’li ve 10’un 2’li çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir. 5’in 2’li çapraz geçerliliğinin ise 10’un 1’li, 10’un 2’li, 10’un 3’lü, 10’un 4’lü ve 5’in 1’li olmak üzere tüm diğer çapraz geçerlilik değerlerinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

7.2.4 TA02V01 sonuçları

Çizelge 7.40 : TA02V01’in farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	3,50	0,00197
	Grup içi	0,02	1040	0,00		
	Toplam	0,03	1046			
Toplam iterasyon sayısı	Gruplar arası	7416432,62	6	1236072,10	20,54	0,00000
	Grup içi	62598097,37	1040	60190,48		
	Toplam	70014530,00	1046			

Çizelge 7.40 TA02V01'in farklı çarpaz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde OHK bakımından ve toplam iterasyon sayısı bakımından istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.41'de Levene testi yapılmış ve ortalama hata karesi bakımından varyansların homojen olmadığı, toplam iterasyon sayısı bakımından ise varyansların homojen olduğu görülmüştür. Bunun üzerine Çizelge 7.42'de görülen Tukey'in HSD testi ve Çizelge 7.43'te verilen Tamhane ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.41 : TA02V01'in farklı ÇG değerleri için Levene testi sonuçları sonuçları.

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Ortalama Hata Karesi	1,75	6	1040	0,10716
Toplam İterasyon Sayısı	97,18	6	1040	0,00000

Çizelge 7.42 : OHK bakımından TA02V01'in farklı ÇG değerlerinin Tukey'in HSD Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama Fark Oranı (I-J)/J	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG			
TA02V01	10'un 1'li	TA02V01	10'un 4'lü	-0,11	0,00	0,03100
TA02V01	10'un 1'li	TA02V01	10'un 5'li	-0,12	0,00	0,00600
TA02V01	10'un 1'li	TA02V01	5'in 2'li	-0,12	0,00	0,01200
TA02V01	10'un 4'lü	TA02V01	10'un 1'li	0,12	0,00	0,03100
TA02V01	10'un 5'li	TA02V01	10'un 1'li	0,14	0,00	0,00600
TA02V01	5'in 2'li	TA02V01	10'un 1'li	0,13	0,00	0,01200

Çizelge 7.42'de görülen TA02V01 için Tukey'in HSD testi sonuçları incelendiğinde 10'un 1'li çarpaz geçerliliğinin 10'un 4'lü, 10'un 5'li ve 5'in 2'li çarpaz geçerliliğinden daha düşük ortalamam hata karelerine sahip olduğu görülmektedir.

Çizelge 7.43 : Toplam iterasyon sayısı bakımından TA02V01'in farklı ÇG değerlerinin Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA02V01	10'un 1'li	TA02V01	10'un 3'lü	0,05	15,41	0,02800
TA02V01	10'un 1'li	TA02V01	10'un 4'lü	0,15	23,25	0,00000
TA02V01	10'un 1'li	TA02V01	10'un 5'li	0,19	25,88	0,00000
TA02V01	10'un 1'li	TA02V01	5'in 1'li	0,10	20,36	0,00000
TA02V01	10'un 1'li	TA02V01	5'in 2'li	0,34	30,72	0,00000
TA02V01	10'un 2'li	TA02V01	10'un 4'lü	0,14	23,54	0,00000
TA02V01	10'un 2'li	TA02V01	10'un 5'li	0,18	26,14	0,00000
TA02V01	10'un 2'li	TA02V01	5'in 1'li	0,09	20,69	0,00100
TA02V01	10'un 2'li	TA02V01	5'in 2'li	0,33	30,94	0,00000
TA02V01	10'un 3'lü	TA02V01	10'un 1'li	-0,05	15,41	0,02800
TA02V01	10'un 3'lü	TA02V01	10'un 5'li	0,13	30,12	0,00700
TA02V01	10'un 3'lü	TA02V01	5'in 2'li	0,27	34,37	0,00000
TA02V01	10'un 4'lü	TA02V01	10'un 1'li	-0,13	23,25	0,00000
TA02V01	10'un 4'lü	TA02V01	10'un 2'li	-0,12	23,54	0,00000
TA02V01	10'un 4'lü	TA02V01	5'in 2'li	0,17	38,52	0,02300
TA02V01	10'un 5'li	TA02V01	10'un 1'li	-0,16	25,88	0,00000
TA02V01	10'un 5'li	TA02V01	10'un 2'li	-0,15	26,14	0,00000
TA02V01	10'un 5'li	TA02V01	10'un 3'lü	-0,11	30,12	0,00700
TA02V01	5'in 1'li	TA02V01	10'un 1'li	-0,09	20,36	0,00000
TA02V01	5'in 1'li	TA02V01	10'un 2'li	-0,09	20,69	0,00100
TA02V01	5'in 1'li	TA02V01	5'in 2'li	0,22	36,85	0,00000
TA02V01	5'in 2'li	TA02V01	10'un 1'li	-0,25	30,72	0,00000
TA02V01	5'in 2'li	TA02V01	10'un 2'li	-0,25	30,94	0,00000
TA02V01	5'in 2'li	TA02V01	10'un 3'lü	-0,22	34,37	0,00000
TA02V01	5'in 2'li	TA02V01	10'un 4'lü	-0,15	38,52	0,02300
TA02V01	5'in 2'li	TA02V01	5'in 1'li	-0,18	36,85	0,00000

Çizelge 7.43'te görülen TA02V01 için Tamhane testi sonuçları incelendiğinde 10'un 3'lü çapraz geçerliliğinin 10'un 1'li çapraz geçerliliğinden, 10'un 4'lü ve 5'in 1'li çapraz geçerliliğinin ise 10'un 1'li çapraz geçerliliğinden ve 10'un 2'li çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir. 10'un 5'li çapraz geçerliliğinin, 10'un 1'li çapraz geçerliliğinden, 10'un 2'li çapraz geçerliliğinden ve 10'un 3'lü çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

5'in 2'li çapraz geçerliliğinin ise 10'un 1'li, 10'un 2'li, 10'un 3'lü, 10'un 4'lü ve 5'in 1'li çapraz geçerliliklerinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

7.2.5 TA03V01 sonuçları

Çizelge 7.44 : TA03V01'in farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	3,13	0,00476
	Grup içi	0,00	1042	0,00		
	Toplam	0,00	1048			
Toplam İterasyon Sayısı	Gruplar arası	90224,09	6	15037,35	4,25	0,00031
	Grup içi	3686012,34	1042	3537,44		
	Toplam	3776236,43	1048			

Çizelge 7.44 TA03V01'in farklı çarpaz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde ortalama hata karesi ve toplam iterasyon sayısı bakımından istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.45'te Levene testi yapılmış ve varyansların OHK için homojen olduğu, toplam iterasyon sayısı için homojen olmadığı görülmüştür. Bunun üzerine Çizelge 7.46'da görülen Tukey'in HSD testi ve Çizelge 7.47'de görülen Tamhane testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.45 : TA03V01'in farklı ÇG değerleri için Levene testi sonuçları.

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Ortalama Hata Karesi	2,09	6	1042	0,05235
Toplam İterasyon Sayısı	17,11	6	1042	0,00000

Çizelge 7.46 : OHK bakımından TA03V01'in farklı ÇG değerleri için Tukey'in HSD Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama Fark Oranı (I-J)/J	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG			
TA03V01	10'un 4'lü	TA03V01	5'in 1'li	0,02	0,00	0,02600
TA03V01	10'un 5'li	TA03V01	5'in 1'li	0,02	0,00	0,04800
TA03V01	5'in 1'li	TA03V01	10'un 4'lü	-0,02	0,00	0,02600
TA03V01	5'in 1'li	TA03V01	10'un 5'li	-0,02	0,00	0,04800

Çizelge 7.46’da görülen TA03V01 için Tukey’in HSD testi sonuçları incelendiğinde 5’in 1’li çapraz geçerliliğinin, 10’un 4’lü ve 10’un 5’li çapraz geçerliliğinden daha az ortalama hata karesi değerine sahip olduğu görülmektedir.

Çizelge 7.47 : Toplam iterasyon sayısı bakımından TA03V01’in farklı ÇG değerlerinin için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama Fark Oranı (I-J)/J	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG			
TA03V01	10’un 1’li	TA03V01	5’in 2’li	0,03	8,27	0,04800
TA03V01	10’un 2’li	TA03V01	5’in 2’li	0,03	8,27	0,04800
TA03V01	5’in 2’li	TA03V01	10’un 1’li	-0,03	8,27	0,04800
TA03V01	5’in 2’li	TA03V01	10’un 2’li	-0,03	8,27	0,04800

Çizelge 7.47’de görülen TA03V01 için Tamhane testi sonuçları incelendiğinde 5’in 2’li çapraz geçerliliğinin, 10’un 1’li ve 10’un 2’li çapraz geçerliliğinden daha az GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

7.2.6 TA03V02 sonuçları

Çizelge 7.48 : TA03V02’nin farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	0,55	0,76907
	Grup içi	0,00	1039	0,00		
	Toplam	0,00	1045			
Toplam İterasyon Sayısı	Gruplar arası	302716,09	6	50452,68	3,73	0,00113
	Grup içi	14062716,02	1039	13534,86		
	Toplam	14365432,11	1045			

Çizelge 7.48 incelendiğinde TA03V02’nin farklı çapraz geçerlilik değerleri için OHK bakımından istatistiksel olarak anlamlı bir fark oluşmadığı görülmektedir. Ancak toplam iterasyon sayısı bakımından incelendiğinde ise istatistiksel olarak anlamlı bir fark olduğu görülmektedir. Post hoc testlerini yapmadan önce Çizelge 7.49’da Levene testi yapılmış ve varyansların homojen olmadığı görülmüştür. Bunun üzerine Tamhane testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır. Tamhane testi sonuçları incelendiğinde Tamhane testinin birbirinden farklı grupları tespit edemediği görülmektedir.

Çizelge 7.49 : TA03V02'nin farklı ÇG değerleri için Levene testi sonuçları.

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Toplam İterasyon Sayısı	15,48	6	1039	0,00000

7.3 Aslanağzı (IRIS) Sınıflama Problemi

7.3.1 Problem tanımı

Aslanağzı (Iris) Veri Kümesi, Fisher (1936) tarafından ayırma (discriminant) analizine örnek vermek amacıyla kullanılmıştır. Anderson'un Aslanağzı Veri kümesi olarak da bilinen bu veri kümesi Edgar Anderson tarafından üç farklı türdeki aslanağzı bitkisinin morfolojik özelliklerini tespit etmek amacıyla toplanan verilere dayanır (Anderson, 1936). Üç farklı aslanağzı türünden ikisi Gaspé yarımadasından (Quebec, Canada) aynı gün aynı çayırdan ve ölçümleri aynı kişi aynı ölçüm aletleri ile aynı zamanda ölçülmüştür (Anderson, 1935).

R.A. Fisher tarafından kullanılan, doğrusal olarak ayrılmayan 3 sınıflı bir veri kümesidir. 150 gözlem ile 3 tür (Setosa, Virginica ve Versicolor) aslanağzı (Iris) bitkisini ayırt etmeye dayanır. Her tür için 50'şer gözlem vardır. 4 özellik ile tür tespit edilmeye çalışılır:

- Çanak yaprak (sepal) uzunluğu (cm)
- Çanak yaprak genişliği (cm)
- Taç yaprak (petal) uzunluğu (cm)
- Taç yaprak genişliği (cm)

Çiçeğin dış katmanındaki yapraklar çanak yaprak olarak adlandırılırken daha iç kısımdaki yapraklar ise taç yaprak olarak adlandırılır. Çanak yapraklar Şekil 7.3'te, Şekil 7.4'te ve Şekil 7.5'te 1 nolu ok ile gösterilmiş, taç yapraklar ise aynı şekiller üzerinde 2 nolu ok ile gösterilmiştir.

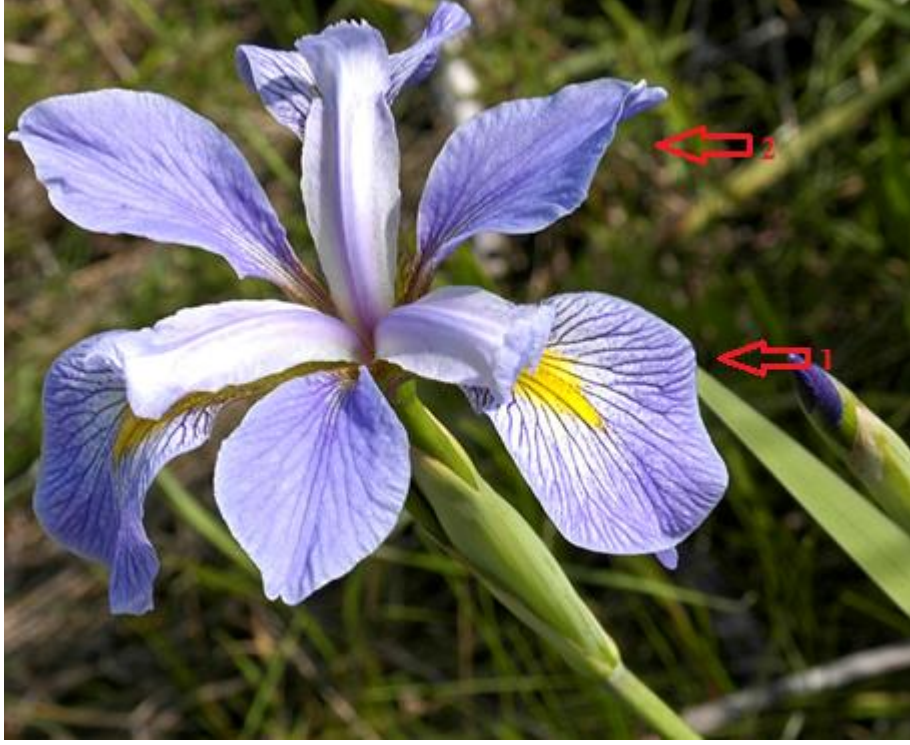
Aslanağzı veri kümesinde bulunan üç tür; Setosa, Versicolor ve Virginica sırasıyla Şekil 7.3'te, Şekil 7.4'te ve Şekil 7.5'te sunulmuştur:



Şekil 7.3 : Setosa türü aslanağzı çiçeği (Wikipedia).

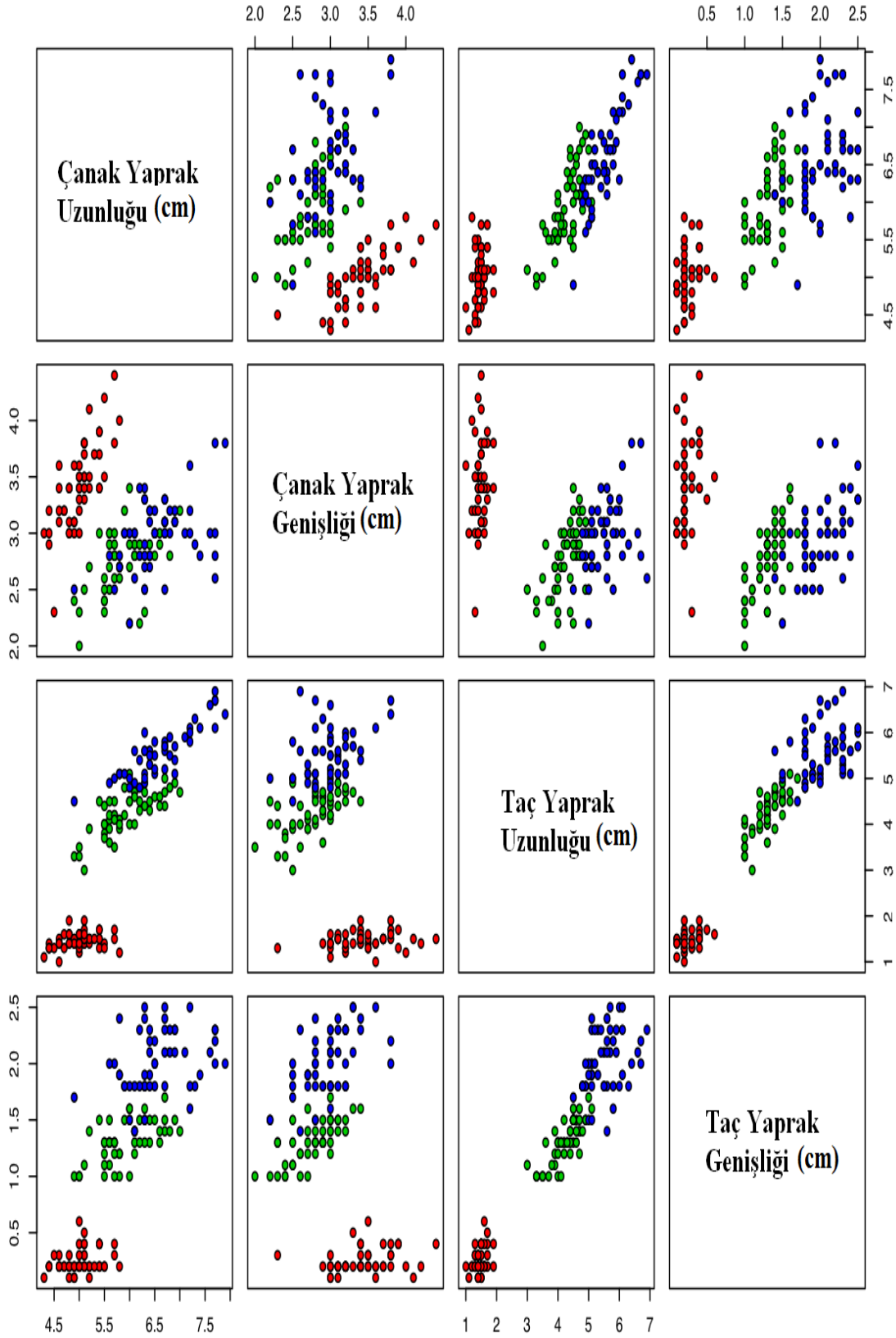


Şekil 7.4 : Versicolor türü aslanağzı çiçeği (Wikipedia).



Şekil 7.5 : Virginia türü aslanağzı çiçeği (Wikipedia).

Aslanağzı veri kümesinin sık kullanılmasını sağlayan temel özellik doğrusal olarak sınıfların birbirinden ayırlamamasıdır. Dört özelliğin, sınıf örneklemeleri üzerinde ikili kıyaslamaları Şekil 7.6’da verilmiştir. Şekil 7.6’da verilen serpilme diyagramında köşegende satır ve sütun başlıkları belirtilmiştir, böylece dört değişkenin ikili grafikleri çizilerek aslanağzı türlerinin verisinin dağılımı gösterilmeye çalışılmıştır. Şekil 7.6’da setosa türünün doğrusal olarak versicolor ve virginica türünden ayrılabilirdiği görülmektedir. Özellikle versicolor ve virginica türlerinin örnekleri birbirine çok yakın veya birbirinin içinde; aynı değerler için iki türden de örneklerin bulunduğu görülmektedir. Bu durum aslanağzı veri kümesini yazında sınıflama için sık kullanılan bir veri kümesi haline getirmektedir.



Şekil 7.6 : Aslanağzı türleri, kırmızı; setosa, yeşil; versicolor, mavi; virginica (Wikipedia).

7.3.2 Uygulanan işlemler

Aslanağzı Veri Kümesi'ndeki santimetre olarak belirtilen dört farklı ölçüm değeri Denklem 7.1 kullanılarak her bir ölçüm türü için $[0,1]$ aralığına normalize edilmiştir. Normalizasyon işleminden sonra veri kümesindeki her tür için 10'ar adet gözlem test için ayrılmıştır. Bu çalışmada test verisi olarak yazında kabul görmüş her türün son 10 gözlemi tercih edilmiştir. Aslanağzı sınıflama probleminde elde edilen sonuçların karşılaştırılması için öncelikle tüm algoritmaların en uygun parametrelerinin tespit edilmesi gerekmektedir. Bu amaçla öncelikle her algoritmanın Çizelge 7.50'de verilen yedi farklı çapraz geçerlilik değeri kıyaslanmıştır.

Çizelge 7.50 : Aslanağzı sınıflama problemi için önerilen algoritmalarda kullanılan farklı çapraz geçerlilik değerleri.

Sıra No	K	L
1	10	1
2	10	2
3	10	3
4	10	4
5	10	5
6	5	1
7	5	2

7.3.3 TA01V01 sonuçları

Çizelge 7.51 : TA01V01'in 2 PE'li farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Doğru sınıflandırma yüzdesi	Gruplar arası	67,08	6	11,18	0,69	0,66024
	Grup içi	16860,47	1036	16,27		
	Toplam	16927,54	1042			
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	0,54	0,78135
	Grup içi	0,20	1036	0,00		
	Toplam	0,20	1042			
Toplam iterasyon sayısı	Gruplar arası	15543165,06	6	2590527,51	13,47	0,00000
	Grup içi	199232453,03	1036	192309,32		
	Toplam	214775618,09	1042			

Çizelge 7.51 TA01V01'in farklı çarpaz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde doğru sınıflandırma yüzdesi bakımından ve OHK bakımından farklı çarpaz geçerlilik değerlerinin arasında anlamlı bir fark oluşmamaktadır. Toplam iterasyon sayısı bakımından ise istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.52'de Levene testi yapılmış ve varyansların homojen olmadığı görülmüştür. Bunun üzerine Çizelge 7.53'te görülen Tamhane testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.52 : TA01V01'in 2 PE'li farklı ÇG değerleri için Levene testi sonuçları.

Levene İstatistiği	sd 1	sd 2	Anlamlılık
10,75	6	1036	0,00000

Çizelge 7.53 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 2 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA01V01	10'un 1'li	TA01V01	5'in 1'li	0,11	43,28	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 2'li	0,33	54,88	0,00000
TA01V01	10'un 2'li	TA01V01	5'in 1'li	0,12	44,34	0,00000
TA01V01	10'un 2'li	TA01V01	5'in 2'li	0,36	55,72	0,00000
TA01V01	10'un 3'lü	TA01V01	5'in 1'li	0,10	45,59	0,00000
TA01V01	10'un 3'lü	TA01V01	5'in 2'li	0,31	56,72	0,00000
TA01V01	10'un 4'lü	TA01V01	5'in 1'li	0,08	50,03	0,00300
TA01V01	10'un 4'lü	TA01V01	5'in 2'li	0,26	60,35	0,00100
TA01V01	5'in 1'li	TA01V01	10'un 1'li	-0,10	43,28	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 2'li	-0,11	44,34	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 3'lü	-0,09	45,59	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 4'lü	-0,08	50,03	0,00300
TA01V01	5'in 2'li	TA01V01	10'un 1'li	-0,12	54,88	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 2'li	-0,13	55,72	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 3'lü	-0,11	56,72	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 4'lü	-0,10	60,35	0,00100

Çizelge 7.53'te görülen TA01V01 2 PE için Tamhane testi sonuçları incelendiğinde 5'in 1'li ve 5'in 2'li çarpaz geçerliliklerinin 10'un 1'li, 10'un 2'li, 10'un 3'lü ve 10'un 4'lü çarpaz geçerliliğinden daha az GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

Çizelge 7.54 : TA01V01’in 5 PE’li farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Doğru Sınıflandırma Yüzdesi	Gruplar arası	4,72	6	0,79	0,41	0,87076
	Grup içi	1973,45	1036	1,90		
	Toplam	1978,17	1042			
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	3,52	0,00186
	Grup içi	0,02	1036	0,00		
	Toplam	0,02	1042			
Toplam İterasyon Sayısı	Gruplar arası	32039875	6	5339979,29	33,59	0,00000
	Grup içi	164714384	1036	158990,72		
	Toplam	196754260	1042			

Çizelge 7.54 TA01V01’in farklı çarpaz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde doğru sınıflandırma yüzdesi bakımından farklı çarpaz geçerlilik değerlerinin arasında anlamlı bir fark oluşmamaktadır. Ortalama hata karesi ve toplam iterasyon sayısı bakımından ise istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.55’te Levene testi yapılmış ve varyansların homojen olmadığı görülmüştür. Bunun üzerine de görülen Tamhane testleri ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.55 : TA01V01’in 5 PE’li farklı ÇG değerleri için Levene testi sonuçları.

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Ortalama Hata Karesi	12,56	6	1036	0,00000
Toplam İterasyon Sayısı	10,78	6	1036	0,00000

TA01V01 5 PE OHK sonuçları için Tamhane testi sonuçları incelendiğinde testin birbirinden farklı grupları tespit edemediği görülmektedir. Bunun üzerine Dunnet ve Games-Howell testleri ile inceleme yapılmış ve aynı durumla karşılaşmıştır.

Çizelge 7.56 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 5 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA01V01	10'un 1'li	TA01V01	10'un 4'lü	0,06	36,73	0,00200
TA01V01	10'un 1'li	TA01V01	10'un 5'li	0,11	49,23	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 1'li	0,12	37,38	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 2'li	0,52	50,93	0,00000
TA01V01	10'un 2'li	TA01V01	10'un 4'lü	0,05	38,24	0,01000
TA01V01	10'un 2'li	TA01V01	10'un 5'li	0,11	50,36	0,00000
TA01V01	10'un 2'li	TA01V01	5'in 1'li	0,11	38,86	0,00000
TA01V01	10'un 2'li	TA01V01	5'in 2'li	0,51	52,02	0,00000
TA01V01	10'un 3'lü	TA01V01	10'un 5'li	0,10	50,49	0,00000
TA01V01	10'un 3'lü	TA01V01	5'in 1'li	0,11	39,03	0,00000
TA01V01	10'un 3'lü	TA01V01	5'in 2'li	0,49	52,15	0,00000
TA01V01	10'un 4'lü	TA01V01	10'un 1'li	-0,05	36,73	0,00200
TA01V01	10'un 4'lü	TA01V01	10'un 2'li	-0,05	38,24	0,01000
TA01V01	10'un 4'lü	TA01V01	5'in 1'li	0,06	41,00	0,00800
TA01V01	10'un 4'lü	TA01V01	5'in 2'li	0,37	53,64	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 1'li	-0,10	49,23	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 2'li	-0,10	50,36	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 3'lü	-0,09	50,49	0,00000
TA01V01	10'un 5'li	TA01V01	5'in 2'li	0,23	62,86	0,00800
TA01V01	5'in 1'li	TA01V01	10'un 1'li	-0,11	37,38	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 2'li	-0,10	38,86	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 3'lü	-0,10	39,03	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 4'lü	-0,06	41,00	0,00800
TA01V01	5'in 1'li	TA01V01	5'in 2'li	0,22	54,08	0,00200
TA01V01	5'in 2'li	TA01V01	10'un 1'li	-0,18	50,93	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 2'li	-0,18	52,02	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 3'lü	-0,18	52,15	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 4'lü	-0,14	53,64	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 5'li	-0,09	62,86	0,00800
TA01V01	5'in 2'li	TA01V01	5'in 1'li	-0,09	54,08	0,00200

Çizelge 7.56'da görülen TA01V01 5 PE için Tamhane testi sonuçları incelendiğinde 10'un 4'lü çapraz geçerliliğinin 10'un 1'li çapraz geçerliliğinden, 10'un 4'lü ve 5'in 1'li çapraz geçerliliğinin ise 10'un 1'li çapraz geçerliliğinden ve 10'un 2'li çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir. 10'un 5'li çapraz geçerliliğinin, 10'un 1'li çapraz geçerliliğinden, 10'un 2'li çapraz geçerliliğinden ve 10'un 3'lü çapraz geçerliliğinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

5'in 1'li çapraz geçerliliğinin ise 10'un 1'li, 10'un 2'li, 10'un 3'lü ve 10'un 4'lü çapraz geçerliliklerinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir. 5'in 2'li çapraz geçerliliğinin ise 10'un 1'li, 10'un 2'li, 10'un 3'lü, 10'un 4'lü ve 5'in 1'li olmak üzere diğer tüm çapraz geçerlilik değerlerinden daha az sayıda GYA iterasyonuna ihtiyaç duyduğu görülmektedir.

Çizelge 7.57 : TA01V01'in 10 PE'li farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Doğru Sınıflandırma Yüzdesi	Gruplar arası	8,14	6	1,36	1,08	0,37380
	Grup içi Toplam	1303,81 1311,95	1036 1042	1,26		
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	9,49	0,00000
	Grup içi Toplam	0,01 0,01	1036 1042	0,00		
Toplam İterasyon Sayısı	Gruplar arası	33395957	6	5565992,99	32,84	0,00000
	Grup içi Toplam	175596199 208992157	1036 1042	169494,40		

Çizelge 7.57 TA01V01'in farklı çapraz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde doğru sınıflandırma yüzdesi bakımından farklı çapraz geçerlilik değerlerinin arasında anlamlı bir fark oluşmamaktadır. Ortalama hata karesi ve toplam iterasyon sayısı bakımından ise istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.58'de Levene testi yapılmış ve varyansların homojen olmadığı görülmüştür. Bunun üzerine Çizelge 7.59'da ve Çizelge 7.60'da görülen Tamhane testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.58 : TA01V01'in 10 PE'li farklı ÇG değerleri için Levene testi sonuçları.

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Ortalama Hata Karesi	42,44	6	1036	0,00000
Toplam İterasyon Sayısı	19,66	6	1036	0,00000

Çizelge 7.59 : OHK bakımından TA01V01’in farklı ÇG değerlerinin 10 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA01V01	10’un 1’li	TA01V01	10’un 4’lü	-0,05	0,00	0,00200
TA01V01	10’un 1’li	TA01V01	10’un 5’li	-0,08	0,00	0,00000
TA01V01	10’un 1’li	TA01V01	5’in 2’li	-0,01	0,00	0,00100
TA01V01	10’un 2’li	TA01V01	10’un 5’li	-0,07	0,00	0,00000
TA01V01	10’un 2’li	TA01V01	5’in 2’li	-0,01	0,00	0,01000
TA01V01	10’un 3’lü	TA01V01	10’un 5’li	-0,06	0,00	0,00600
TA01V01	10’un 4’lü	TA01V01	10’un 1’li	0,05	0,00	0,00200
TA01V01	10’un 5’li	TA01V01	10’un 1’li	0,09	0,00	0,00000
TA01V01	10’un 5’li	TA01V01	10’un 2’li	0,08	0,00	0,00000
TA01V01	10’un 5’li	TA01V01	10’un 3’lü	0,06	0,00	0,00600
TA01V01	10’un 5’li	TA01V01	5’in 1’li	0,08	0,00	0,00000
TA01V01	5’in 1’li	TA01V01	10’un 5’li	-0,07	0,00	0,00000
TA01V01	5’in 1’li	TA01V01	5’in 2’li	-0,01	0,00	0,01000
TA01V01	5’in 2’li	TA01V01	10’un 1’li	0,07	0,00	0,00100
TA01V01	5’in 2’li	TA01V01	10’un 2’li	0,06	0,00	0,01000
TA01V01	5’in 2’li	TA01V01	5’in 1’li	0,06	0,00	0,01000

Çizelge 7.59’da görülen Tamhane testi sonuçları incelendiğinde 10’un 1’li çapraz geçerliliğinin, 10’un 4’lü, 10’un 5’li ve 5’in 2’li çapraz geçerliliklerinden daha düşük ortalama hata karesi seviyesine sahip olduğu görülmektedir.

10’un 2’li çapraz geçerliliğinin, 10’un 5’li ve 5’in 2’li çapraz geçerliliklerinden daha düşük ortalama hata karesi seviyesine sahip olduğu görülmektedir.

10’un 3’lü çapraz geçerliliğinin, 10’un 5’li çapraz geçerliliğinden daha düşük ortalama hata karesi seviyesine sahip olduğu görülmektedir.

5’in 1’li çapraz geçerliliğinin, 10’un 5’li ve 5’in 2’li çapraz geçerliliklerinden daha düşük ortalama hata karesi seviyesine sahip olduğu görülmektedir.

Çizelge 7.60 : Toplam iterasyon sayısı bakımından TA01V01'in farklı ÇG değerlerinin 10 PE için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA01V01	10'un 1'li	TA01V01	10'un 3'lü	0,05	31,62	0,00300
TA01V01	10'un 1'li	TA01V01	10'un 4'lü	0,09	44,15	0,00000
TA01V01	10'un 1'li	TA01V01	10'un 5'li	0,09	42,69	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 1'li	0,09	36,59	0,00000
TA01V01	10'un 1'li	TA01V01	5'in 2'li	0,62	53,16	0,00000
TA01V01	10'un 2'li	TA01V01	10'un 4'lü	0,07	45,04	0,00100
TA01V01	10'un 2'li	TA01V01	10'un 5'li	0,07	43,61	0,00200
TA01V01	10'un 2'li	TA01V01	5'in 1'li	0,07	37,66	0,00000
TA01V01	10'un 2'li	TA01V01	5'in 2'li	0,57	53,90	0,00000
TA01V01	10'un 3'lü	TA01V01	10'un 1'li	-0,04	31,62	0,00300
TA01V01	10'un 3'lü	TA01V01	5'in 2'li	0,49	55,89	0,00000
TA01V01	10'un 4'lü	TA01V01	10'un 1'li	-0,08	44,15	0,00000
TA01V01	10'un 4'lü	TA01V01	10'un 2'li	-0,07	45,04	0,00100
TA01V01	10'un 4'lü	TA01V01	5'in 2'li	0,38	63,82	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 1'li	-0,08	42,69	0,00000
TA01V01	10'un 5'li	TA01V01	10'un 2'li	-0,06	43,61	0,00200
TA01V01	10'un 5'li	TA01V01	5'in 2'li	0,39	62,82	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 1'li	-0,09	36,59	0,00000
TA01V01	5'in 1'li	TA01V01	10'un 2'li	-0,07	37,66	0,00000
TA01V01	5'in 1'li	TA01V01	5'in 2'li	0,37	58,85	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 1'li	-0,21	53,16	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 2'li	-0,20	53,90	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 3'lü	-0,17	55,89	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 4'lü	-0,14	63,82	0,00000
TA01V01	5'in 2'li	TA01V01	10'un 5'li	-0,14	62,82	0,00000
TA01V01	5'in 2'li	TA01V01	5'in 1'li	-0,14	58,85	0,00000

Çizelge 7.60'da görülen Tamhane testi sonuçları incelendiğinde 10'un 3'lü çapraz geçerliliğinin, 10'un 1'li çapraz geçerliliğinden daha düşük GYA iterasyon sayısına ihtiyaç duyduğu görülmektedir.

10'un 4'lü çapraz geçerliliğinin, 10'un 1'li ve 10'un 2'li çapraz geçerliliklerinden daha düşük GYA iterasyon sayısına ihtiyaç duyduğu görülmektedir.

10'un 5'li çapraz geçerliliğinin, 10'un 1'li ve 10'un çapraz geçerliliklerinden daha düşük GYA iterasyon sayısına ihtiyaç duyduğu görülmektedir.

5'in 1'li çapraz geçerliliğinin, 10'un 1'li ve 10'un 2'li çapraz geçerliliklerinden daha düşük GYA iterasyon sayısına ihtiyaç duyduğu görülmektedir.

5'in 2'li çapraz geçerliliğinin, 10'un 1'li, 10'un 2'li, 10'un 3'lü, 10'un 4'lü, 10'un 5'li, 5'in 1'li çapraz geçerliliklerinden daha düşük GYA iterasyon sayısına ihtiyaç duyduğu görülmektedir.

7.3.4 TA02V01 sonuçları

Çizelge 7.61 : TA02V01'in farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Doğru sınıflandırma yüzdesi	Gruplar arası	3170,84	6	528,47	1,80	0,09496
	Grup içi	303346,37	1036	292,81		
	Toplam	306517,21	1042			
Ortalama Hata Karesi	Gruplar arası	0,03	6	0,01	1,70	0,11813
	Grup içi	3,40	1036	0,00		
	Toplam	3,44	1042			
Toplam iterasyon sayısı	Gruplar arası	15926646	6	2654441,00	20,24	0,00000
	Grup içi	135890843	1036	131168,77		
	Toplam	151817489	1042			

Çizelge 7.61 TA02V01'in farklı çapraz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde doğru sınıflandırma yüzdesi bakımından ve OHK bakımından farklı çapraz geçerlilik değerlerinin arasında anlamlı bir fark oluşmamaktadır. Toplam iterasyon sayısı bakımından ise istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.62'de Levene testi yapılmış ve varyansların homojen olduğu görülmüştür. Bunun üzerine Çizelge 7.63'te görülen Tukey'in HSD testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.62 : TA02V01'in farklı ÇG değerleri için Levene testi sonuçları.

Levene İstatistiği	sd 1	sd 2	Anlamlılık
1,83	6	1036	0,08953

Çizelge 7.63 : Toplam iterasyon sayısı bakımından TA02V01'in farklı ÇG değerlerinin Tukey'in HSD Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA02V01	10'un 1'li	TA02V01	5'in 1'li	0,07	41,96	0,00200
TA02V01	10'un 1'li	TA02V01	5'in 2'li	0,13	41,96	0,00000
TA02V01	10'un 2'li	TA02V01	5'in 1'li	0,10	41,96	0,00000
TA02V01	10'un 2'li	TA02V01	5'in 2'li	0,17	41,96	0,00000
TA02V01	10'un 3'lü	TA02V01	5'in 1'li	0,07	41,96	0,00300
TA02V01	10'un 3'lü	TA02V01	5'in 2'li	0,13	41,96	0,00000
TA02V01	10'un 4'lü	TA02V01	5'in 1'li	0,07	41,96	0,00100
TA02V01	10'un 4'lü	TA02V01	5'in 2'li	0,13	41,96	0,00000
TA02V01	10'un 5'li	TA02V01	5'in 1'li	0,08	41,96	0,00000
TA02V01	10'un 5'li	TA02V01	5'in 2'li	0,14	41,96	0,00000
TA02V01	5'in 1'li	TA02V01	10'un 1'li	-0,06	41,96	0,00200
TA02V01	5'in 1'li	TA02V01	10'un 2'li	-0,09	41,96	0,00000
TA02V01	5'in 1'li	TA02V01	10'un 3'lü	-0,06	41,96	0,00300
TA02V01	5'in 1'li	TA02V01	10'un 4'lü	-0,07	41,96	0,00100
TA02V01	5'in 1'li	TA02V01	10'un 5'li	-0,07	41,96	0,00000
TA02V01	5'in 1'li	TA02V01	5'in 2'li	0,06	41,96	0,02300
TA02V01	5'in 2'li	TA02V01	10'un 1'li	-0,12	41,96	0,00000
TA02V01	5'in 2'li	TA02V01	10'un 2'li	-0,14	41,96	0,00000
TA02V01	5'in 2'li	TA02V01	10'un 3'lü	-0,11	41,96	0,00000
TA02V01	5'in 2'li	TA02V01	10'un 4'lü	-0,12	41,96	0,00000
TA02V01	5'in 2'li	TA02V01	10'un 5'li	-0,12	41,96	0,00000
TA02V01	5'in 2'li	TA02V01	5'in 1'li	-0,06	41,96	0,02300

Çizelge 7.63'te görülen Tukey'in HSD testi sonuçları incelendiğinde 5'in 1'li çapraz geçerliliğinin, 10'un 1'li, 10'un 2'li, 10'un 3'lü, 10'un 4'lü ve 10'un 5'li çapraz geçerliliklerinden daha düşük GYA iterasyon sayısına ihtiyaç duyduğu görülmektedir.

5'in 2'li çapraz geçerliliğinin, 10'un 1'li, 10'un 2'li, 10'un 3'lü, 10'un 4'lü, 10'un 5'li ve 5'in 1'li çapraz geçerliliklerinden daha düşük GYA iterasyon sayısına ihtiyaç duyduğu görülmektedir.

7.3.5 TA03V01 sonuçları

Çizelge 7.64 : TA03V01'in farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Doğru Sınıflandırma Yüzdesi	Gruplar arası	197,33	6	32,89	6,50	0,00000
	Grup içi	5243,13	1036	5,06		
	Toplam	5440,45	1042			
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	4,43	0,00020
	Grup içi	0,00	1036	0,00		
	Toplam	0,00	1042			
Toplam İterasyon Sayısı	Gruplar arası	1509276,52	6	251546,09	6,56	0,00000
	Grup içi	39719591,33	1036	38339,37		
	Toplam	41228867,85	1042			

Çizelge 7.64 TA03V01'in farklı çarpaz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde doğru sınıflandırma yüzdesi, ortalama hata karesi ve toplam iterasyon sayısı bakımından ise istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.65'te Levene testi yapılmış ve varyansların homojen olmadığı görülmüştür. Bunun üzerine Çizelge 7.66'da, Çizelge 7.67'de ve Çizelge 7.68'de görülen Tamhane testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.65 : TA03V01'in farklı ÇG değerleri için Levene testi sonuçları.

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Doğru Sınıflama Yüzdesi	2,20	6	1036	0,04086
Ortalama Hata Karesi	17,78	6	1036	0,00000
Toplam İterasyon Sayısı	26,02	6	1036	0,00000

Çizelge 7.66 : Doğru sınıflama yüzdesi bakımından TA03V01'in farklı ÇG değerlerinin Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA03V01	10'un 1'li	TA03V01	10'un 2'li	-0,01	0,26	0,00100
TA03V01	10'un 1'li	TA03V01	10'un 3'lü	-0,02	0,26	0,00000
TA03V01	10'un 1'li	TA03V01	10'un 4'lü	-0,02	0,27	0,00000
TA03V01	10'un 1'li	TA03V01	10'un 5'li	-0,02	0,27	0,00000
TA03V01	10'un 1'li	TA03V01	5'in 2'li	-0,01	0,25	0,00100
TA03V01	10'un 2'li	TA03V01	10'un 1'li	0,02	0,26	0,00100
TA03V01	10'un 3'lü	TA03V01	10'un 1'li	0,02	0,26	0,00000
TA03V01	10'un 4'lü	TA03V01	10'un 1'li	0,02	0,27	0,00000
TA03V01	10'un 5'li	TA03V01	10'un 1'li	0,02	0,27	0,00000
TA03V01	5'in 2'li	TA03V01	10'un 1'li	0,01	0,25	0,00100

Çizelge 7.66'da görülen Tamhane testi sonuçları incelendiğinde 10'un 2'li, 10'un 3'lü, 10'un 4'lü, 10'un 5'li, ve 5'in 2'li çapraz geçerliliğinin, 10'un 1'li çapraz geçerliliğinden daha yüksek doğru sınıflama yüzdesine sahip olduğu görülmektedir.

Çizelge 7.67 : OHK bakımından TA03V01'in farklı ÇG değerleri için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA03V01	10'un 3'lü	TA03V01	5'in 2'li	0,01	0,00	0,01400
TA03V01	10'un 4'lü	TA03V01	5'in 2'li	0,01	0,00	0,01400
TA03V01	10'un 5'li	TA03V01	5'in 2'li	0,01	0,00	0,04200
TA03V01	5'in 2'li	TA03V01	10'un 3'lü	-0,01	0,00	0,01400
TA03V01	5'in 2'li	TA03V01	10'un 4'lü	-0,01	0,00	0,01400
TA03V01	5'in 2'li	TA03V01	10'un 5'li	-0,01	0,00	0,04200

Çizelge 7.67'de görülen Tamhane testi sonuçları incelendiğinde 5'in 2'li çapraz geçerliliğinin, 10'un 3'lü, 10'un 4'lü ve 10'un 5'li çapraz geçerliliklerinden daha düşük ortalama hata karesi seviyesine sahip olduğu görülmektedir.

Çizelge 7.68 : Toplam iterasyon sayısı bakımından TA03V01'in farklı ÇG değerlerinin için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG	Fark Oranı (I-J)/J		
TA03V01	10'un 1'li	TA03V01	5'in 2'li	0,04	29,88	0,00300
TA03V01	10'un 2'li	TA03V01	5'in 2'li	0,04	29,88	0,00300
TA03V01	10'un 3'lü	TA03V01	5'in 2'li	0,04	30,26	0,00600
TA03V01	5'in 2'li	TA03V01	10'un 1'li	-0,04	29,88	0,00300
TA03V01	5'in 2'li	TA03V01	10'un 2'li	-0,04	29,88	0,00300
TA03V01	5'in 2'li	TA03V01	10'un 3'lü	-0,04	30,26	0,00600

Çizelge 7.68'de görülen Tamhane testi sonuçları incelendiğinde 5'in 2'li çapraz geçerliliğinin, 10'un 1'li, 10'un 2'li ve 10'un 3'lü çapraz geçerliliklerinden daha düşük GYA iterasyon sayısına ihtiyaç duyduğu görülmektedir.

7.3.6 TA03V02 sonuçları

Çizelge 7.69 : TA03V02'nin farklı ÇG değerleri için ANOVA testi sonuçları.

Değerlendirme Türü	Kareler Toplamı Türü	Kareler Toplamı	sd	Ortalama Kare	F	Anl.
Doğru Sınıflandırma Yüzdesi	Gruplar arası	21,31	6	3,55	1,34	0,23419
	Grup içi	2735,44	1036	2,64		
	Toplam	2756,75	1042			
Ortalama Hata Karesi	Gruplar arası	0,00	6	0,00	3,77	0,00102
	Grup içi	0,01	1036	0,00		
	Toplam	0,01	1042			
Toplam İterasyon Sayısı	Gruplar arası	6116105,68	6	1019350,95	13,17	0,00000
	Grup içi	80166773,17	1036	77381,06		
	Toplam	86282878,85	1042			

Çizelge 7.69 TA03V02'nin farklı çapraz geçerlilik değerleri için sonuçlar arasında istatistiksel olarak anlamlı farklar olup olmadığını göstermektedir. Çizelge incelendiğinde doğru sınıflandırma yüzdesi bakımından istatistiksel olarak anlamlı bir fark oluşmadığı görülmektedir. Ancak çizelge ortalama hata karesi ve toplam iterasyon sayısı bakımından incelendiğinde ise istatistiksel olarak anlamlı bir fark oluşmaktadır. Post hoc testlerini yapmadan önce Çizelge 7.70'de Levene testi yapılmış ve varyansların homojen olmadığı görülmüştür. Bunun üzerine

Çizelge 7.71’de görülen Tukey testi ve Çizelge 7.72’de görülen Tamhane testi ile birbirinden farklı ÇG değerleri tespit edilmeye çalışılmıştır.

Çizelge 7.70 : TA03V02’nin farklı ÇG değerleri için Levene testi sonuçları.

Performans Ölçütü	Levene İstatistiği	sd 1	sd 2	Anlamlılık
Ortalama Hata Karesi	12,76	6	1036	0,00000
Toplam İterasyon Sayısı	0,65	6	1036	0,68856

Çizelge 7.71 : OHK bakımından TA03V02’nin farklı ÇG değerleri için Tukey’in HSD Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama Fark Oranı (I-J)/J	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG			
TA03V02	10’un 3’lü	TA03V02	10’un 5’li	-0,05	0,00	0,03000
TA03V02	10’un 5’li	TA03V02	10’un 3’lü	0,05	0,00	0,03000
TA03V02	10’un 5’li	TA03V02	5’in 1’li	0,05	0,00	0,03000
TA03V02	5’in 1’li	TA03V02	10’un 5’li	-0,05	0,00	0,03000

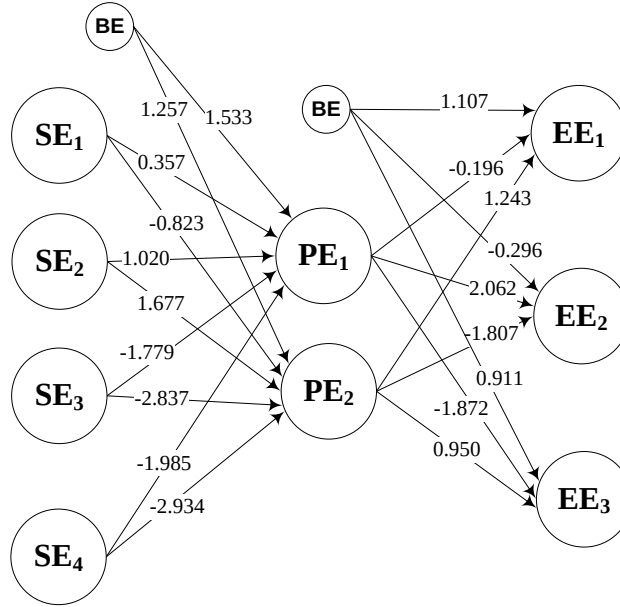
Çizelge 7.71’de görülen Tukey’in HSD testi sonuçları incelendiğinde 10’un 3’lü ve 5’in 1’li çapraz geçerliliklerinin, 10’un 5’li çapraz geçerliliğinden daha düşük ortalama hata karesi seviyesine sahip olduğu görülmektedir.

Çizelge 7.72 : Toplam iterasyon sayısı bakımından TA03V02’nin farklı ÇG değerleri için Tamhane Post Hoc testi ile analizi.

(I) Deneme		(J) Deneme		Ortalama Fark Oranı (I-J)/J	Standart Hata	Anl.
Algoritma	ÇG	Algoritma	ÇG			
TA03V02	10’un 1’li	TA03V02	5’in 2’li	0,08	39,32	0,00000
TA03V02	10’un 2’li	TA03V02	5’in 2’li	0,09	39,57	0,00000
TA03V02	10’un 3’lü	TA03V02	5’in 2’li	0,09	39,97	0,00000
TA03V02	10’un 4’lü	TA03V02	5’in 2’li	0,08	40,99	0,00000
TA03V02	10’un 5’li	TA03V02	5’in 2’li	0,06	42,69	0,00200
TA03V02	5’in 1’li	TA03V02	5’in 2’li	0,06	42,71	0,00100
TA03V02	5’in 2’li	TA03V02	10’un 1’li	-0,08	39,32	0,00000
TA03V02	5’in 2’li	TA03V02	10’un 2’li	-0,08	39,57	0,00000
TA03V02	5’in 2’li	TA03V02	10’un 3’lü	-0,08	39,97	0,00000
TA03V02	5’in 2’li	TA03V02	10’un 4’lü	-0,08	40,99	0,00000
TA03V02	5’in 2’li	TA03V02	10’un 5’li	-0,06	42,69	0,00200
TA03V02	5’in 2’li	TA03V02	5’in 1’li	-0,06	42,71	0,00100

Çizelge 7.72’de görülen Tamhane testi sonuçları incelendiğinde 5’in 2’li çapraz geçerliliğinin, 10’un 1’li, 10’un 2’li, 10’un 3’lü, 10’un 4’lü, 10’un 5’li, 5’in 1’li çapraz

geçerliliklerinden daha düşük GYA iterasyon sayısına ihtiyaç duyduğu görülmektedir.



Şekil 7.7 : Iris Probleminde %98.67 (148/150) oranında doğru sınıflandıran TA01V01 ile elde edilen 2 PE'li YSA.

Şekil 7.7'de TA01V01 ile elde edilen Test verisinde % 100 (30/30) oranında ve eğitim verisinde % 98,33 (118/120) doğru sınıflama yapan iki işlem elemanlı yapay sinir ağı yapısı verilmiştir. Bu şekilde başlangıç elemanları “SE”, eşik değerlerini temsil eden eşik elemanları “BE”, gizli katmandaki işlem elemanları “PE” ve çıktıları üreten bitiş elemanları ise “EE” ile gösterilmiştir. Katmanların birbiriyle tam olarak bağlı olduğu bu yapıda her bir bağlantının ağırlığı (sinaptik ağırlık) ilgili okun üzerine yazılmıştır.

Şekil 7.7'de görülen yapı toplamda % 98,67 (148/150) doğru sınıflama yüzdesi ile literatürdeki mevcut algoritmaların oluşturduğu en yüksek doğru sınıflama yüzdesi olan % 98,67 (148/150) seviyesinde sadece iki işlem elemanlı yapısı ile ulaşmıştır. Bu yapıda PE'leri ve EE'leri $\tanh(.)$ transfer fonksiyonu kullanmaktadır BE'ler sabit çıktı (1) üreten eşik elemanlarını göstermektedir.

7.4 Uygulama Sonuçları

7.4.1 En uygun parametreler

Algoritmaların farklı problemlerdeki performanslarını tespit etmeden önce her bir algoritmanın her bir performans kriteri için en uygun olan parametrelerini tespit etmek daha doğru olacaktır.

Çizelge 7.73 : Algoritmaların öne çıkan çapraz geçerlilik değerleri.

Algoritma	Performans Kriteri		
	Doğru Sınıflama Yüzdesi	Ortalama Hata Karesi	GYA İterasyon Sayısı
TA01V01 & 2 PE	Anlamlı Fark Yoktur	10'un 1'li 10'un 2'li	5'in 1'li 5'in 2'li
TA01V01 & 5 PE	Anlamlı Fark Yoktur	10'un 1'li	5'in 2'li
TA01V01 & 10 PE	Anlamlı Fark Yoktur	10'un 1'li 10'un 2'li 5'in 1'li	5'in 2'li
TA02V01	Anlamlı Fark Yoktur	10'un 1'li	5'in 2'li
TA03V01	10'un 1'li hariç hepsi	5'in 1'li 5'in 2'li	5'in 2'li
TA03V02	Anlamlı Fark Yoktur	10'un 3'lü 5'in 1'li	5'in 2'li

Çizelge 7.73 incelendiğinde doğru sınıflama yüzdesinin farklı K ve L değerlerinden genelde etkilenmediği görülmektedir. TA03V01 algoritması için ise 10'un 1'li çapraz geçerliliği diğer tüm değerlerden kötü sonuç vermektedir.

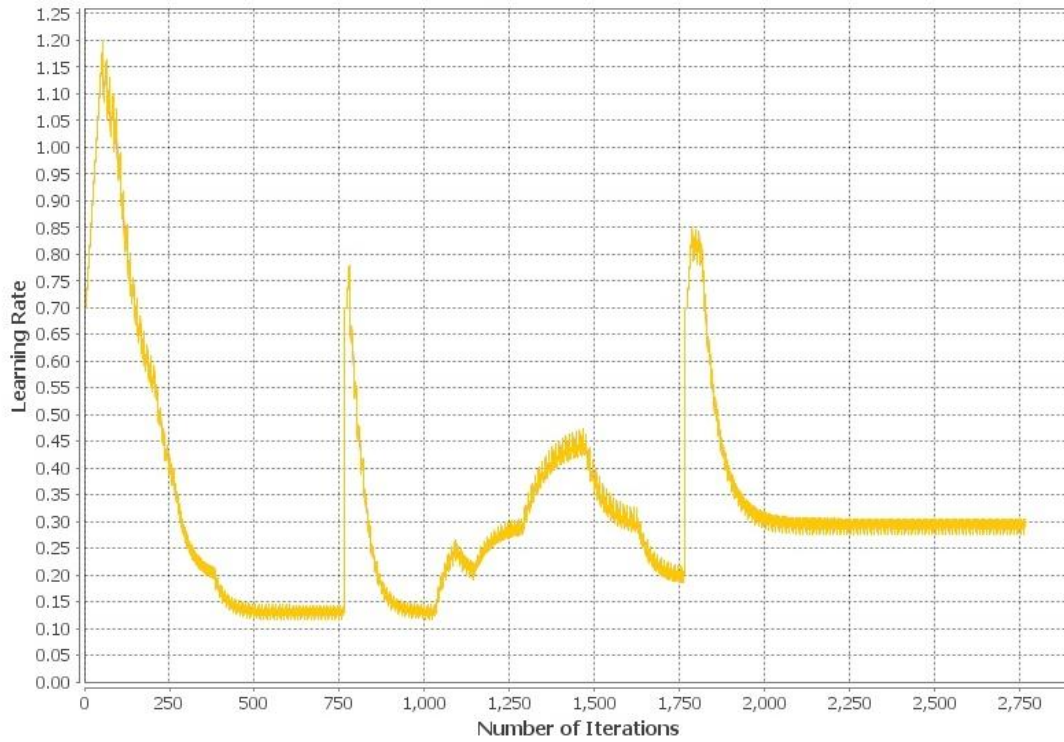
Çizelge 7.73'te net olarak görülen bir noktada 5'in 2'li çapraz geçerliliğinin yeni önerilen dört algortmada da (TA01V01, TA02V01, TA03V01 ve TA03V02) GYA toplam iterasyon ihtiyacını çok net bir şekilde düşürdüğüdür. Ayrıca Çizelge 7.73'ten yola çıkarak 10'un 1'li çapraz geçerliliğinin OHK açısından öne çıktığını da söylenebilir.

Çizelge 7.73'te özetlenen değerler ışığında algoritmaların kullanımı sırasında Çizelge 7.74'te verilen çapraz geçerlilik değerlerinin kullanılması önerilmektedir.

Çizelge 7.74 : Algoritmaların önerilen çapraz geçerlilik değerleri.

Algoritma	Performans Kriteri		
	Doğru Sınıflama Yüzdesi	Ortalama Hata Karesi	GYA İterasyon Sayısı
TA01V01	10'un 1'li	10'un 1'li	5'in 2'li
TA02V01	10'un 1'li	10'un 1'li	5'in 2'li
TA03V01	5'in 2'li	5'in 2'li	5'in 2'li
TA03V02	10'un 1'li	5'in 1'li	5'in 2'li

7.4.2 Adaptif öğrenme oranı



Şekil 7.8 : Adaptif kavrama oranı değişim örneği.

Bu çalışmada adaptif kavrama oranı kullanılmış ve bu oranın değişimine bir örnek Şekil 7.8'de verilmiştir. Bu örnek üç çıktı sinir hücresi gerektiren bir problemde DYSA algoritmalarının geri yayılım algoritmasında kullandıkları adaptif kavrama oranının değişimini göstermektedir. DYSA algoritması üç çıktı sinir hücresi için üç defa bir çıktı sinir hücresi eklemesi yapmış ve üç defa geri yayılım algoritması ile öğrenme yapmıştır. Her seferinde geri yayılım algoritması ön tanımlı değer olan 0.7 ile başlamıştır. Sonrasında kavrama arttıkça bu değer artmış, kavrama azaldıkça bu değer düşürülmüştür.

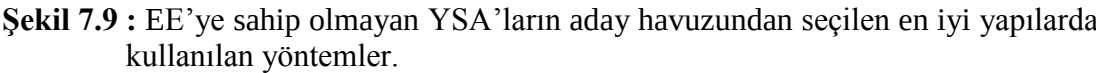
Şekil 7.8’de özellikle 750’inci ile 1750’inci iterasyonlar arasında bu oranın adaptif olarak nasıl değiştiği görülmektedir. Kavrama oranı ön tanımlı olan 0,7 ile başlamış ve biraz yükseldikten sonra 0,13-0,15 seviyelerine kadar düşmüştür. Kavramanın artmasıyla bu seviyeden tekrar 0,25 seviyesine kadar çıkmış ve kavramadaki azalmayla birlikte 0,2 seviyesine gerilemiş ve sonrasında tekrar artış göstermiştir.

7.4.3 Aday havuzu oluşturma yöntemleri

Bu çalışma kapsamında önerilen aday havuzu oluşturma yöntemleri Bölüm 4.3’te açıklanmıştı. Bu bölümde ise aday havuzlarından seçilen, en iyi performansı sağlayan adayların özellikleri incelenmiş ve bu seçilen adayların yapılarında yer alan PE’lerinin ve EE’lerinin yapısının hangi yöntemlerle oluşturulduğu araştırılmıştır.

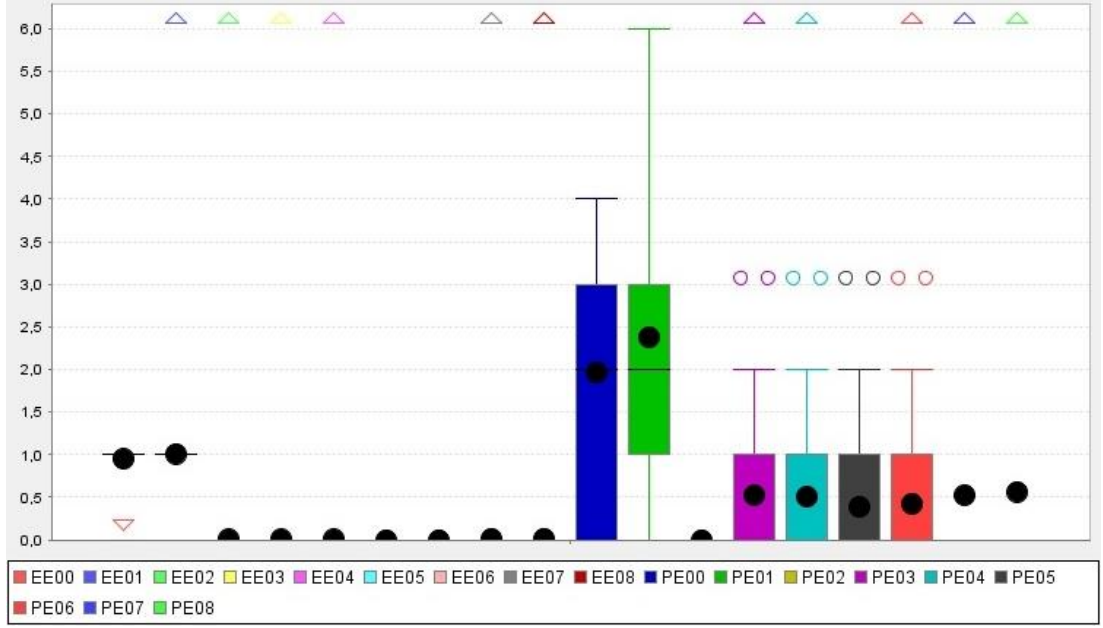
Bu kapsamda, mevcut yapay sinir ağında hiç bitiş elemanı bulunmadığı durumda aday havuzu oluşturma yöntemlerinin örnek istatistiki Şekil 7.9’da, tek bitiş elemanı bulunduğu durumda aday havuzu oluşturma yöntemlerinin örnek istatistiki Şekil 7.10’da, birden fazla bitiş elemanı bulunduğu durumda aday havuzu oluşturma yöntemlerinin örnek istatistiki Şekil 7.11’de verilmiştir. Bu örnekler 17’nci hesaplamada elde edilen (Çizelge 6.2) örneklerdir. Yapılan incelemede tüm hesaplamalarda benzer sonuçların elde edildiği görülmüştür.

Verilen kutu diyagramlarında (box plot) üst ve alt çizgi sırasıyla en büyük ve en küçük değeri gösterirken bunların dışında kalan noktalar istisnai değer (outlier) olarak gösterilmektedir. Oluşan dikdörtgenin üst sınırı ve alt sınırı sırasıyla birinci ve üçüncü dördlülüğü (quartile), dikdörtgenin orta çizgisi medyanı içi dolu yuvarlak ortalamayı göstermektedir. Kutu grafiklerindeki tüm kutular ile serilerin adları aynı sırada verilmiştir.



Önerilen DYSA algoritmalarının, mevcut yapay sinir ağında hiç bitiş elemanı bulunmadığı durumda, oluşturdukları aday yapılardan seçtikleri en iyi adaylarda kullandıkları yöntemlere ait boxplot grafiği Şekil 7.9’da verilmiştir. Bu kutu grafikleri incelendiğinde sadece rassal tek EE ekleme yöntemi (#EE01) ile bitiş elemanı eklendiği görülmektedir. Bu durum çok doğaldır, çünkü bitiş elemanı olmayan ağa yeni bir bitiş elemanı eklemek için önerilen tek yöntem rassal tek EE ekleme yöntemidir.

İşlem elemanı eklemede ise öne çıkan yöntemler ise rassal tek PE ekleme yöntemi (#PE01) ve paylaşırma ile tek PE ekleme yöntemidir (#PE03). En iyi adayların yapısında hiç rastlanmayan yöntem ise ayırma ile tek PE ekleme yöntemidir (#PE02).



Şekil 7.10 : Bir EE'ye sahip YSA'ların aday havuzundan seçilen en iyi yapılarda kullanılan yöntemler.

Önerilen DYSA algoritmalarının, mevcut yapay sinir ağında bir bitiş elemanı bulunduğu durumda, oluşturdukları aday yapılardan seçtikleri en iyi adaylarda kullandıkları yöntemlere ait boxplot grafiği Şekil 7.10'da verilmiştir. Bu kutu grafikleri incelendiğinde bitiş elemanı eklemede öne çıkan yöntemler kopyalama ile tek EE ekleme yöntemi (#EE00) ve rassal tek EE ekleme yöntemidir (#EE01).

İşlem elemanı eklemede ise öne çıkan yöntemler ise kopyalama ile tek PE ekleme yöntemi (#PE00) ve rassal tek PE ekleme yöntemidir (#PE01). Seçilen en iyi aday yapılarda kullanılan diğer işlem elemanı ekleme yöntemleri ise paylaşırma ile tek PE ekleme yöntemi (#PE03) ters işaretli ağırlıkla PE ekleme yöntemi (#PE04), toplama ile PE ekleme yöntemi (#PE05) ve birleştirme ile PE ekleme yöntemidir (#PE06). En iyi adayların yapısında hiç rastlanmayan yöntem ise ayırma ile tek PE ekleme yöntemidir (#PE02).



Şekil 7.11 : Birden fazla EE'ye sahip YSA'ların aday havuzundan seçilen en iyi yapılarda kullanılan yöntemler.

Önerilen DYSA algoritmalarının, mevcut yapay sinir ağında birden fazla bitiş elemanı bulunduğu durumda, oluşturdukları aday yapılardan seçtikleri en iyi adaylarda kullandıkları yöntemlere ait boxplot grafiği Şekil 7.10'da verilmiştir. Bu kutu grafikleri incelendiğinde bitiş elemanı eklemede öne çıkan yöntemler kopyalama ile tek EE ekleme yöntemi (#EE00) ve rassal tek EE ekleme yöntemidir (#EE01).

İşlem elemanı eklemede ise öne çıkan yöntemler ise kopyalama ile tek PE ekleme yöntemi (#PE00) ve rassal tek PE ekleme yöntemidir (#PE01). İşlem elemanı eklemede en iyi aday yapısında sık rastlanan diğer yöntemler ise ayırma ile iki aday PE ekleme yöntemi (#PE07) ve paylaşırma ile iki aday PE ekleme yöntemidir (#PE08). Seçilen en iyi aday yapılarda kullanılan diğer işlem elemanı ekleme yöntemleri ise paylaşırma ile tek PE ekleme yöntemi (#PE03) ters işaretli ağırlıkla PE ekleme yöntemi (#PE04), toplama ile PE ekleme yöntemi (#PE05) ve birleştirme ile PE ekleme yöntemidir (#PE06). En iyi adayların yapısında hiç rastlanmayan yöntem ise ayırma ile tek PE ekleme yöntemidir (#PE02).

7.4.4 Algoritmaların kıyaslanması

Yapılan ön incelemede SPSS Clementine yazılımında YSA modeli kurulurken dinamik (dynamic) yöntemin, budama (prune), yorucu budama (exhausting prune) gibi diğer yöntemlere göre daha iyi sonuçlar verdiği gözlenmiştir. Bu sebeble bu çalışmada SPSS Clementine yazılımında YSA yapıları “dinamik” yöntemiyle oluşturulmuştur. NeuroSolutions yazılımında ise öğrenme oranı, başlangıç ağırlıkları ve gizli katmandaki işlem elemanı sayısının en iyi değerleri genetik algoritma ile 25 nesil boyunca 25 kromozom ile aranmış ve bu aramada bulunan en iyi sonuçlar saklanmıştır.

Her bir algoritmanın ve yazılımın 20 tekrar sonucunda elde edilen en iyi ve ortalama performans göstergeleri ve bu değerlerin standart sapması Ve/Veya/Özel Veya Problemi için Çizelge 7.75, Wolf’un SunSpot zaman serisi problemi için Çizelge 7.76 ve Aslanağzı sınıflama problemi için Çizelge 7.77 sunulmuştur.

Çizelge 7.75 : Ve/Veya/Özel Veya problemi test verisinde elde edilen sonuçlar.

Algoritmanın	Performans Kriteri							
	SPSS Clementine / Dinamik YSA	Neuro Solutions / 3 Katmanlı ÇKA	TA01V01 2PE	TA01V01 5PE	TA01V01 10PE	TA02V01	TA03V01	TA03V02
En İyi Parametreleri	Dinamik	Genetik Algoritma	10'un 1'li	10'un 1'li	10'un 1'li	10'un 1'li	5'in 2'li	5'in 1'li
En İyi İkili Fonksiyon Doğru Yüzdesi	% 100	% 100	% 100	% 100	%100	% 100	% 92	% 100
En İyi İkili Fonksiyon Doğru Yüzdesine Göre Standart Sapma	% 0	% 0	% 7	% 3	% 4	% 4	% 2	% 4
En İyi OHK Sonucu	0,00004	0,00000	0,00100	0,00100	0,00100	0,00100	0,11000	0,00100
En İyi OHK Sonucuna Göre Standart Sapma	0,00009	0,00000	0,21591	0,21591	0,03928	0,03902	0,00000	0,04003
Ortalama İkili Fonksiyon Doğru Yüzdesi	% 100	% 100	% 91	% 98	% 97	% 94	% 91	% 95
Ortalama İkili Fonksiyon Doğru Yüzdesine Göre Standart Sapma	% 0	% 0	% 7	% 3	% 4	% 4	% 2	% 4
Ortalama OHK Sonucu	0,00017	0,00000	0,05650	0,01785	0,03005	0,06035	0,11000	0,05115
Ortalama OHK Sonucuna Göre Standart Sapma	0,00009	0,00000	0,03974	0,03395	0,04030	0,04004	0,00000	0,04107

Çizelge 7.76 : Wolf'un SunSpot Zaman Serisi Problemi test verisinde elde edilen sonuçlar.

Algoritmanın	Performans Kriteri							
	SPSS Clementine / Dinamik YSA	Neuro Solutions / 3 Katmanlı ÇKA	TA01V01 2PE	TA01V01 5PE	TA01V01 10PE	TA02V01	TA03V01	TA03V02
En İyi Parametreleri	Dinamik	Genetik Algoritma	10'un 1'li	10'un 1'li	10'un 1'li	10'un 1'li	5'in 2'li	5'in 1'li
En İyi OHK Sonucu	0,01382	0,01277	0,01210	0,01195	0,01125	0,01219	0,01112	0,01151
En İyi OHK Sonucuna Göre Standart Sapma	0,00110	0,00212	0,21591	0,21591	0,00102	0,00140	0,00065	0,00077
Ortalama OHK Sonucu	0,01606	0,01542	0,01582	0,01361	0,01294	0,01403	0,01258	0,01258
Ortalama OHK Sonucuna Göre Standart Sapma	0,00113	0,00217	0,00294	0,00177	0,00105	0,00143	0,00067	0,00079

Çizelge 7.77 : Aslanağzı (Iris) Sınıflama Problemi test verisinde elde edilen sonuçlar.

Algoritmanın	Performans Kriteri							
	SPSS Clementine / Dinamik YSA	Neuro Solutions / 3 Katmanlı ÇKA	TA01V01 2PE	TA01V01 5PE	TA01V01 10PE	TA02V01	TA03V01	TA03V02
En İyi Parametreleri	Dinamik	Genetik Algoritma	10'un 1'li	10'un 1'li	10'un 1'li	10'un 1'li	5'in 2'li	10'un 1'li & 5'in 1'li
En İyi Doğru Sınıflama Yüzdesi	% 100	% 100	% 96,7	% 100	% 100	% 100	% 76,7	% 100
En İyi Doğru Sınıflama Yüzdesine Göre Standart Sapma	% 3,0	% 2,8	% 0,0	% 0,7	% 1,3	% 18,9	% 1,3	% 1,4
En İyi OHK Sonucu	0,00018	0,01684	0,02111	0,01992	0,01944	0,00934	0,11492	0,01314
En İyi OHK Sonucuna Göre Standart Sapma	0,01118	0,11022	0,21591	0,21591	0,00118	0,07472	0,00083	0,00276
Ortalama Doğru Sınıflama Yüzdesi	% 97,0	% 96,7	% 96,7	% 96,8	% 97,3	% 74,5	% 73,2	% 99,0
Ortalama Doğru Sınıflama Yüzdesine Göre Standart Sapma	% 3,0	% 2,9	% 0,0	% 0,7	% 1,4	% 19,4	% 1,3	% 1,5
Ortalama OHK Sonucu	0,01410	0,05965	0,02279	0,02224	0,02220	0,10060	0,11569	0,01899
Ortalama OHK Sonucuna Göre Standart Sapma	0,01147	0,11309	0,00063	0,00111	0,00121	0,07667	0,00085	0,00283

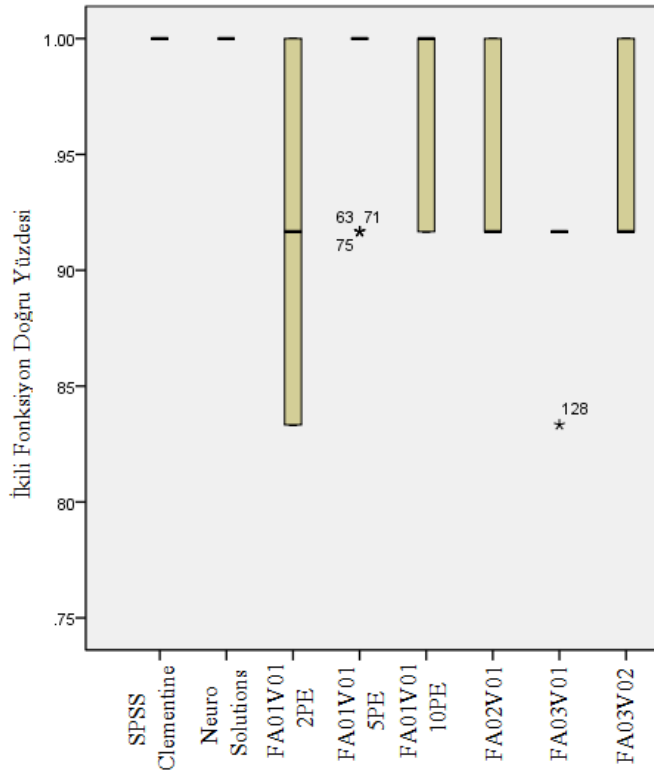
7.5 Bulguların Değerlendirilmesi

Bu çalışmada önerilen algoritmaların performansları ortalama hata karesi, doğru sınıflama yüzdesi ve doğru ikili (binary) fonksiyon tahmini açısından değerlendirilmiştir. Bu değerlendirme sırasında önerilen algoritmalar için farklı çapraz geçerlilik sonuçları ve algoritmaların hangi aday oluşturma yöntemlerini daha çok kullandıkları incelenmiştir.

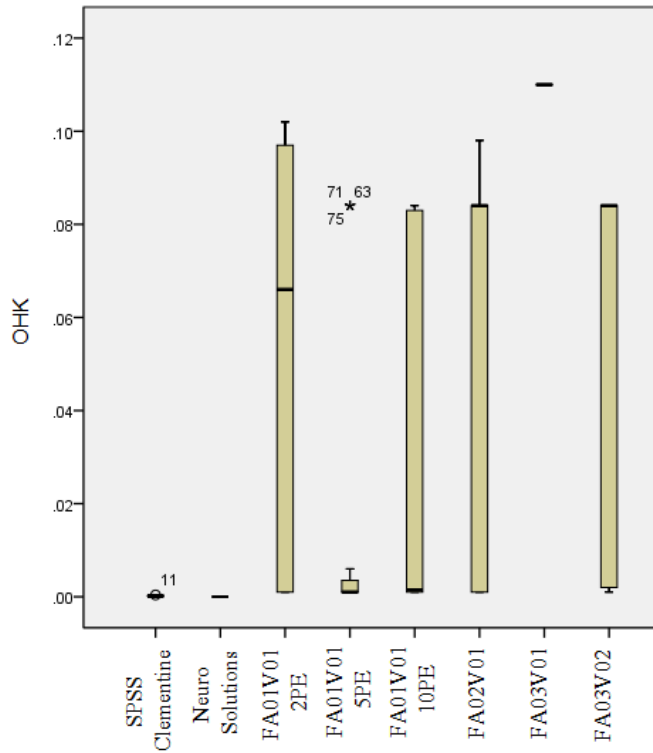
Önerilen algoritmalar, mevcut yazılımlardan SPSS Clementine'nın yapay sinir ağı modeli (dinamik) ve NeuroSolutions v.5.06 yazılımının genetik algoritma ile 25 kromozomun 25 nesil boyunca başlangıç ağırlıkları, öğrenme oranları, momentum oranlarını ve gizli hücre sayısını tarayarak oluşturduğu tek gizli katmana sahip çok katmanlı algılayıcılar model ile karşılaştırılmıştır.

Yapılan karşılaştırmalar için üç örnek problem kullanılmıştır. Bu problemler sırasıyla ikili fonksiyon tahminine dayanan ve/veya/özel veya (and/or/xor) mantık problemi, ortalama hata karesini en küçüklenmeye çalışıldığı Wolf'un SunSpot problemi ve son olarak Fisher (1936) tarafından ayırma (discriminant) analizi için kullanılan, türleri doğru sınıflamaya dayanan, Aslanağzı (Iris) Sınıflama problemidir.

İlk problem olan ve/veya/özel veya mantık problemi iki adet ikili (binary) girdiye ve üç adet ikili çıktıya sahip bir problemidir. Bu problemin ayrıntıları Bölüm 7.1.1'de açıklanmıştır. Yapılan karşılaştırmalar sonucunda (Çizelge 7.75) SPSS Clementine ve NeuroSolutions programlarının oluşturduğu modeller 20 çalıştırmanın her seferinde ikili fonksiyon sonuçlarını %100 doğrulukla veren modeller kurmuşlardır. Önerilen algoritmalarından TA01V01, TA02V01 ve TA03V02 ise ikili fonksiyon sonuçlarını %100 doğrulukla veren sonuçlar üretmenin yanında 20 tekrarın hepsinde bu sonuca ulaşamamışlardır. TA03V01 algoritması ise 20 tekrarda en fazla %92 oranında doğrulukla ikili fonksiyon değerlerini üretmiştir. Yirmi tekrar sonucunda elde edilen ikili fonksiyon doğru yüzdeleri Şekil 7.12'de sunulmuştur. Yirmi tekrar sırasında önerilen dört algoritma da "ikili fonksiyon doğru yüzdesi" performans kriterine göre en uygun yapı adaylarını seçip yapay sinir ağı yapısına katmışlardır.



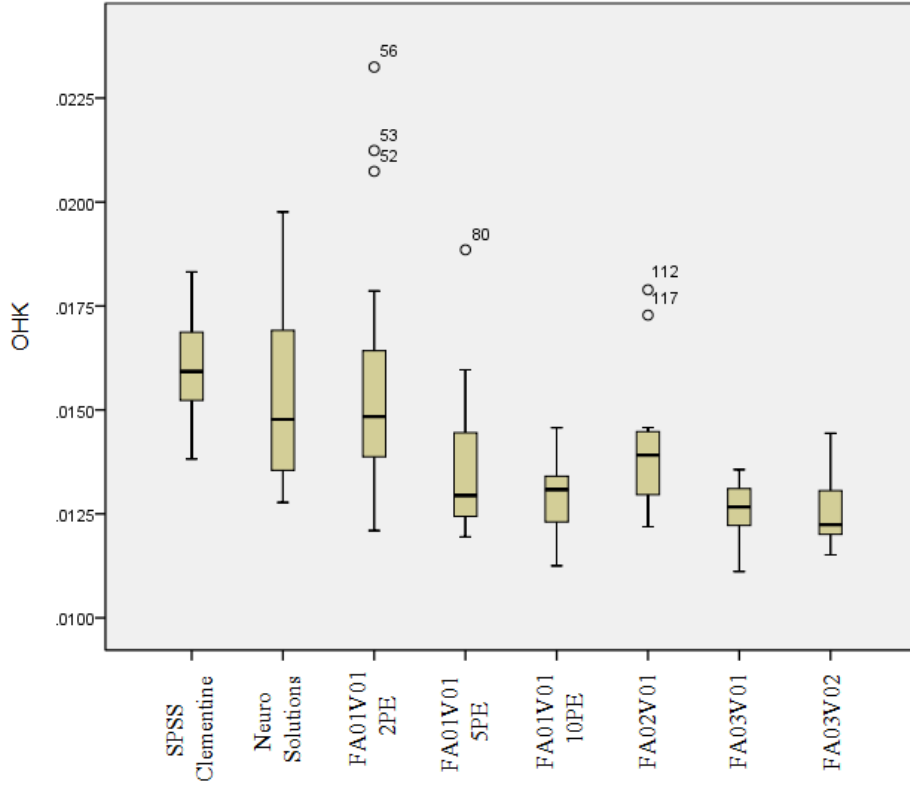
Şekil 7.12 : Ve/Veya/Özel Veya problemi ikili fonksiyon doğru yüzdesi sonuçları.



Şekil 7.13 : Ve/Veya/Özel Veya problemi OHK sonuçları.

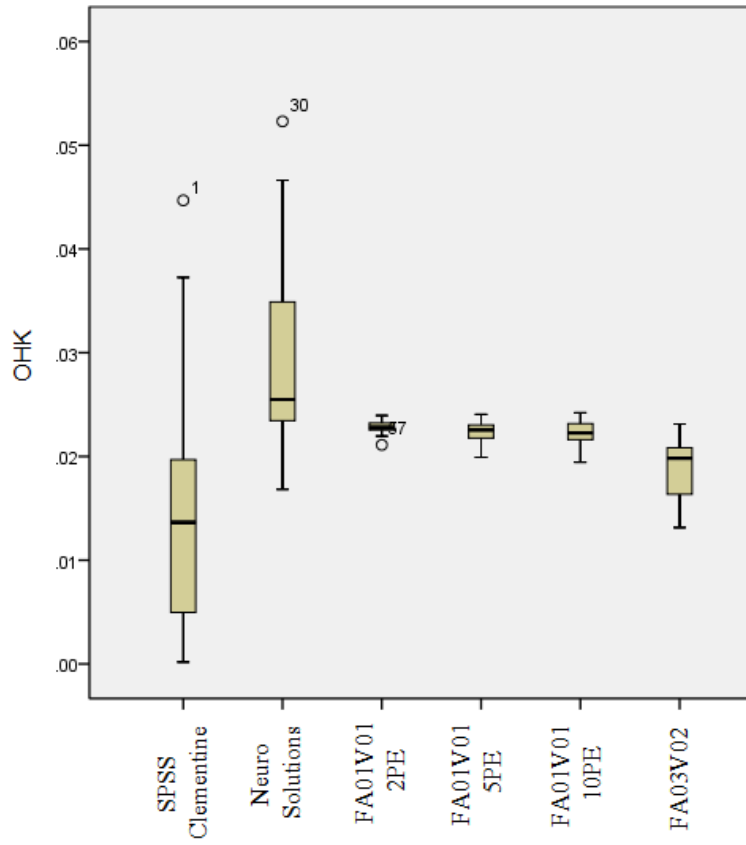
Karşılaştırmaların yapıldığı ikinci problem ise Wolf'un Güneş yüzeyinde oluşan lekelerin yıllık kayıtlarından oluşan (Bölüm 7.2.1) SunSpot veri kümesidir. Yapılan

karşılaştırmalar sonucunda (Çizelge 7.76). SPSS Clementine ve NeuroSolutions programlarının oluşturduğu modeller önerilen dört algorithmadan da daha yüksek ortalama hata karelerine sahip olmuşlardır. Yirmi tekrar sonucunda elde ortalama hata kareleri Şekil 7.14'te sunulmuştur. Yirmi tekrar sırasında önerilen dört algoritma da “ortalama hata karesi” performans kriterine göre en uygun yapı adaylarını seçip yapay sinir ağı yapısına katmışlardır.

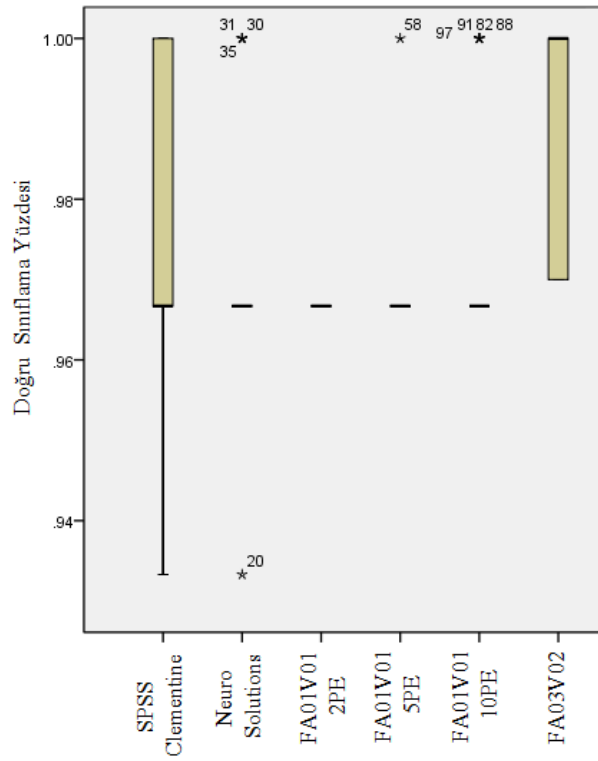


Şekil 7.14 : Wolf’un SunSpot zaman serisi problemi OHK sonuçları.

Karşılaştırmalarda kullanılan son problem ise Aslanağzı Sınıflama Problemidir (Bölüm 7.3). Yapılan karşılaştırmalar sonucunda (Çizelge 7.77) SPSS Clementine ve NeuroSolutions programlarının oluşturduğu modeller ile önerilen TA01V01, TA02V01 ve TA03V02 algoritmaları 20 çalıştırmada test verisinde %100 doğru sınıflama oranına ulaşabilmişlerdir. TA03V01 algoritması ise 20 tekrarda en fazla %76,7 oranında doğrulukla test kümesini doğru sınıflandırabilmiştir. Yirmi tekrar sonucunda test verisinde elde edilen doğru sınıflama yüzdeleri Şekil 7.16 :de sunulmuştur. Yirmi tekrar sırasında önerilen dört algoritma da “doğru sınıflama yüzdesi” performans kriterine göre en uygun yapı adaylarını seçip yapay sinir ağı yapısına katmışlardır.



Şekil 7.15 : Aslanağzı (Iris) problemi OHK sonuçları.



Şekil 7.16 : Aslanağzı (Iris) problemi % doğru sınıflama sonuçları.

Önerilen algoritmalar için performans kriterine / problem türüne göre en uygun olduğu tespit edilen ve ilerdeki çalışmalarda kullanılması önerilen K 'nın L 'li çapraz geçerliliği değerleri Çizelge 7.74'te verilmiştir. 10'un 1'li çapraz geçerliliği, 10-kat çapraz geçerlilik, TA01V01 ve TA02V01 algoritmalarında hem ortalama hata karesi hemde doğru sınıflama yüzdesi açısından uygun parametreler olarak tespit edilmiştir. 10-kat çapraz geçerlilik TA03V02 doğru sınıflama yüzdesi bakımından en uygun parametre iken, TA03V02 algoritması için 5'in 1'li çapraz geçerliliği ortalama hata karesi bakımından en uygun parametre olarak tespit edilmiştir. TA03V01 algoritması için ise ortalama hata karesi, doğru sınıflama yüzdesi ve geri yayılım algoritması toplam iterasyon sayısı bakımından uygun en uygun parametre olarak 5'in 2'li çapraz geçerliliği belirlenmiştir.

5'in 2'li çapraz geçerliliğinin önerilen dört algoritmanın herbirinde geri yayılım algoritması için gerekli toplam iterasyon sayısını önemli bir şekilde azalttığı tespit edilmiştir. Yapılan incelemede 5'in 2'li çapraz geçerliliğinin geri yayılım algoritması toplam iterasyon sayısını düşürürken ortalama hata karesi veya doğru sınıflama yüzdesi gibi diğer performans kriterlerinde performans kayıplarına sebep olduğu gözlenmiştir. Bu durum 5'in 2'li çapraz geçerliliğinin, geri yayılım algoritmasında öğrenmeden taviz verdiğini göstermektedir. Süre azlığı sorun olduğu durumlarda, 5'in 2'li çapraz geçerliliği önerilen dört algortmada alternatif bir parametre seçeneği olarak kullanılabilir.

Bu çalışma kapsamında önerilen dört algoritma da karşılaştırma problemlerinde kayda değer performans göstermiştir. Önerilen dört algortmadan her problemde ve her performans kriterinde öne çıkan algoritma TA03V02 olurken, en az performans sergileyen algoritma ise TA03V01 olmuştur.

Yapılan karşılaştırmalar sırasında önemli bir ysa yapısı elde edilmiştir. Aslanağzı probleminde TA01V01 ile elde edilen 2 işlem elemanlı bir yapı Test verisinde % 100 (30/30) oranında ve eğitim verisinde % 98,33 (118/120) doğru sınıflama yapan elde edilmiştir. Şekil 7.7 görülen bu yapı toplamda % 98,67 (148/150) doğru sınıflama yüzdesi ile literatürdeki mevcut algoritmaların oluşturduğu en yüksek doğru sınıflama yüzdesi olan % 98,67 (148/150) seviyesinde sadece iki işlem elemanlı yapısı ile ulaşmıştır. Bu yapıda 4 başlangıç elemanı, 2 işlem elemanı ve 3 bitiş elemanı vardır.

Aday havuzundan algoritmaların seçtiđi en iyi yapılar da rastlanan yöntemler Bölüm 7.4.3’te belirtilmişti. Bu kapsamda ilerideki çalışmalarda kullanılması önerilen yöntem aday havuzu oluşturma yöntem öncelikleri Çizelge 7.78’de sunulmuştur.

Çizelge 7.78 : Aday havuzu oluşturmada önerilen öncelikli yöntemler.

Öncelik Derecesi	Bitiş Elemanı Sayısı		
	0	1	1'den fazla
En Öncelikli	rassal tek EE ekleme (#EE01), rassal tek PE ekleme (#PE01), paylaştırma ile tek PE ekleme (#PE03)	kopyalama ile tek EE ekleme (#EE00), rassal tek EE ekleme (#EE01), kopyalama ile tek PE ekleme (#PE00), rassal tek PE ekleme (#PE01)	tek EE ekleme (#EE00), tek PE ekleme (#PE00), rassal tek PE ekleme (#PE01)
Orta Öncelikli	-	paylaştırma ile tek PE ekleme (#PE03), ters işaretli ağırlıkla PE ekleme (#PE04), toplama ile PE ekleme (#PE05), birleştirme ile PE ekleme (#PE06).	rassal tek EE ekleme (#EE01) ayırma ile iki aday PE ekleme (#PE07), paylaştırma ile iki aday PE ekleme (#PE08).
Düşük Öncelikli	-	-	paylaştırma ile tek PE ekleme (#PE03) ters işaretli ağırlıkla PE ekleme (#PE04), toplama ile PE ekleme (#PE05) birleştirme ile PE ekleme (#PE06)
Önceliksiz	ayırma ile tek PE ekleme (#PE02)	ayırma ile tek PE ekleme (#PE02)	ayırma ile tek PE ekleme (#PE02)

8. SONUÇLAR

Bu çalışmada öncelikle yazında “öğrenme” olarak geçen üç ayrı bilgi edinimini, yapılan tanımlarla birbirinden ayrılmıştır. Bu bilgi edinimlerinden ilki, ilk defa model oluşturmak için çevredeki gözlemlerin kullanılarak modelde belli bir bilgi birikimi sağlamayı *belleme* veya *kavrama* (*knowledge accumulation*) şeklinde tanımlanmış, ikinci tür bilgi birikimini ise çevrede meydana gelişen değişikliklerden dolayı yeni kullanılabilir hale gelen gözlemlerin mevcut bir modele dâhil edilmesini sağladığı için *uyarlama* (*adaption*) olarak tanımlanmıştır. Üçüncü tür bilgi birikimi ise önceden belli konularda bilgi birikimi elde etmiş veya yetenek kazanmış modellerin, yeni bir konuda bilgi birikimi elde edecek veya mevcut yeteneklerine yeni bir yetenek daha kazanacak şekilde genişletilmesi olarak ifade edilmiş ve *geliştirme* (*enhancement*) olarak adlandırılmıştır.

Yapay sinir ağları açısından mevcut yazın incelendiğinde kavrama ve uyarlama yapan algoritmaların mevcut olduğu ama geliştirme yapan algoritmaların varlığına veya “geliştirme” kapsamında değerlendirilebilecek bir yaklaşım sunumuna rastlanmamıştır.

Yapay sinir ağları açısından “geliştirme” yapan bir algoritmanın hangi işlemler sonucu bunu yapabileceği incelenmiştir. Yapılan incelemede bir algoritmanın yapay sinir ağı açısında geliştirme yapılabilmesi için mutlaka yapısına yeni bir işleve sahip, yeni bir amaca hizmet eden, yeni bir işleve sahip bir bitiş elemanı ekleyebilmesi gerektiği tespit edilmiştir.

Yapay sinir ağının yeni bir işleve sahip bir bitiş elemanın eklenmesi matematiksel olarak m boyutlu uzaydan n boyutlu uzaya yapılan $g: R^m \rightarrow R^n$ haritalamanın, m boyutlu uzaydan $n+1$ boyutlu uzaya $g': R^m \rightarrow R^{n+1}$ haritalamaya dönüştürülmesidir.

Modelleme ya da veri madenciliği açısından bakıldığında bu işlem yapay sinir ağının yeni bir çıktı üretmesi veya girdiler ile yeni bir çıktı özelliğini mevcut girdi/çıktı ilişkilendirmesini kullanarak geliştirmesidir.

Bu dönüşüm veya geliştirmelerin zaman içerisinde yapay sinir ağının yaptığı işi veya sağladığı faydaları yapısal olarak değiştirdiği için bu algoritmalar *dinamik yapay sinir ağı algoritmaları* olarak adlandırılmış bu işlemlerin gerçekleştiği yapay sinir ağı da *dinamik yapay sinir ağı* olarak adlandırılmıştır.

4. Bölümdeki çalışmanın sınırlarını daha iyi çizebilmek için dinamik yapay sinir ağı kavramının yanında türetilmiş yapay sinir ağı ve bütünleşik yapay sinir ağı kavramları da oluşturulmuştur. Bu çalışmanın konu kapsamı olarak ise yazındaki eksiklik nedeniyle dinamik yapay sinir ağları olarak belirlenmiştir.

Dinamik yapay sinir ağı oluşturmak için üç farklı temel algoritma; Temel Algoritma 1 Versiyon 1 (TA01V01), Temel Algoritma 2 Versiyon 1 (TA02V01), Temel Algoritma 3 Versiyon 1 (TA03V01) ve Temel Algoritma 3 Versiyon 2 (TA03V02) olmak üzere toplam dört yeni algoritma önerilmiştir.

Önerilen ilk algoritma olan TA01V01, sadece bitiş elemanı eklemektedir. Bu işlemi istenen bitiş elemanı sayısına ulaşıncaya kadar birer birer mevcut yapay sinir ağına bitiş elemanı eklemek suretiyle yapmaktadır.

Önerilen ikinci algoritma olan TA02V01 ise bitiş elemanı eklerken gizli katmana işlem elemanı eklenmesine de olanak sağlamaktadır. TA02V01 bu işlemi istenen bitiş elemanı sayısına ulaşıncaya kadar birer birer mevcut yapay sinir ağına bitiş elemanı eklemek suretiyle yapmaktadır. TA01V01 algoritmasından farklı olarak eklenecek en uygun yapıyı sadece bitiş elemanı veya bitiş elemanı ile birlikte işlem elemanı bulunan aday yapılar arasından seçerek yapmaktadır. Buradaki amaç sürekli bitiş elemanı veya işlem elemanı arasında tercih yapmak yerine yapay sinir ağının performansına olumlu katkı sağlayan yapının bir bütün olarak tespit edilmesidir.

Önerilen üçüncü temel algoritmanın iki versiyonu (TA03V01 ve TA03V02) vardır. Temel Algoritma 3 (TA03), bitiş elemanının ihtiyaç olduğu bir anda herhangi bir şekilde eklenmesini sağlamayı hedeflemektedir. Bu amaçla verilerdeki örüntülerin (pattern) tespit edilmesi başlangıç noktasıdır. Bu çalışmada TA03 kapsamında önce Kohonen'in öz-örgütlemeli haritası kullanılmıştır. Bu aşamadan sonra TA03V01 algoritmasında TA01V01 algoritması kullanılarak sadece bitiş elemanı eklenmesi, TA03V02 algoritmasında ise TA02V01 algoritması kullanılarak sadece bitiş elemanı / bitiş elemanı ile birlikte işlem elemanı eklenmesi önerilmiştir.

Mevcut yapay sinir ağına yapısına eklenebilecek yapıların oluşturulması gerekmektedir. Bu yapıları oluşturmak için ilk akla gelen çözüm tamamen rassal ağırlıklı bağlantılara sahip adayların eklenmesidir. Rassal ağırlıklı yapının yanında, mevcut yapıdaki bilgilerin kullanımına olanak sağlamak veya bazı bilgilerin unutulmasını sağlamak gibi işlevleri sağlayabilecek dokuzar farklı işlem elemanı ve bitiş elemanı oluşturma yöntemleri önerilmiştir. Bu önerilen yöntemlerle oluşturulan yapılar bir aday havuzunda toplanmakta, en uygun aday yapı önceden belirlenmiş performans kriterine bağlı olarak seçilmektedir.

Önerilen dört algoritmada da eklenecek yapının seçimi başlı başına bir problemidir. Bu amaçla yeni çapraz geçerlilik kavramları önerilmiştir. Önerilen çapraz geçerlilik yöntemlerinde, performans kriteri açısından en fazla iyileşme sağlayan aday yapı mevcut yapay sinir ağı yapısına katılması sağlanmıştır. Önerilen yeni algoritmalar ile yeni çapraz geçerlilik alternatifleri üç problem üzerinde test edilerek incelenmiştir.

Uygulamaların yapıldığı ilk problem olan ve/veya/özel veya mantık problemi iki adet ikili (binary) girdiye ve üç adet ikili çıktıya sahip bir problemidir. SPSS Clementine ve NeuroSolutions programlarının oluşturduğu modeller ve önerilen algoritmalarından TA01V01, TA02V01 ve TA03V02 ise ikili fonksiyon sonuçlarını %100 doğrulukla veren sonuçlar üretebilmişlerdir.

Karşılaştırmaların yapıldığı ikinci problem ise Wolf'un Güneş yüzeyinde oluşan lekelerin yıllık kayıtlarından oluşan SunSpot veri kümesidir. Yapılan karşılaştırmalar sonucunda SPSS Clementine ve NeuroSolutions programlarının oluşturduğu modeller önerilen dört algoritmadan da daha yüksek ortalama hata karelerine sahip olmuşlardır. Bu kapsamda önerilen algoritmalar daha yüksek performans göstermiştir.

Karşılaştırmalarda kullanılan son problem ise Aslanağzı Sınıflama Problemidir. Yapılan karşılaştırmalar sonucunda SPSS Clementine ve NeuroSolutions programlarının oluşturduğu modeller ile önerilen TA01V01, TA02V01 ve TA03V02 algoritmaları 20 çalıştırmada test verisinde %100 doğru sınıflama oranına ulaşmışlardır.

Bu çalışma kapsamında önerilen dört algoritma da karşılaştırma problemlerinde kayda değer performans göstermiştir. Önerilen dört algoritmadan her problemde ve her performans kriterinde öne çıkan algoritma **TA03V02** olmuştur.

Yapılan karşılaştırmalar sırasında önemli bir ysa yapısı elde edilmiştir. Aslanağzı probleminde TA01V01 ile elde edilen 2 işlem elemanlı bir yapı Test verisinde % 100 (30/30) oranında ve eğitim verisinde % 98,33 (118/120) doğru sınıflama elde edilmiştir. Şekil 7.7’de (S. 145) görülen bu yapı toplamda % 98,67 (148/150) doğru sınıflama yüzdesi ile literatürdeki mevcut algoritmaların oluşturduğu en yüksek doğru sınıflama yüzdesi olan % 98,67 (148/150) seviyesinde sadece iki işlem elemanlı yapısı ile ulaşmıştır. Bu yapıda 4 başlangıç elemanı, 2 işlem elemanı ve 3 bitiş elemanı vardır.

8.1 Geliştirilen Yöntemlerin Potansiyel Uygulama Alanları

Yapay sinir ağları geniş bir kullanım alanına sahiptirler. Bu çalışma kapsamında önerilen dört algoritma da, yapay sinir ağının yapısını oluşturduklarından, yapay sinir ağının mevcut kullanım alanlarından gözetimli eğitimin kullanıldığı alanlarda sorunsuz bir şekilde kullanılabilirler. Bu alanlar veya problemler, fonksiyon tahmini / haritalama veya sınıflandırma olarak özetlenebilir. Önerilen algoritmalarda hem ikili değişkenler hem de reel değişkenler kullanılabilirler.

Günümüzde verilerin otomatik toplanması nedeniyle, incelenmesi gereken veri miktarı sürekli artmaktadır. Yaşanan çevresel değişimler karşısında yeni bir çıktı ihtiyacı çıkabilir. Bu yeni çıktı yeni bir sınıf (pazara yeni giren bir müşteri türü) veya yeni bir fonksiyon/haritalama oluşturulması ihtiyacı olabilir. Bu durumda yazındaki mevcut algoritmalar tüm yapay sinir ağı modelini yeniden oluştururken, önerilen algoritmalar ise mevcut yapay sinir ağlarını kullanarak üzerine istenen çıktıyı (bitiş elemanını) eklemeye olanak sağlamaktadırlar.

Rekabet ortamının etkisiyle piyasalar sürekli değişmekte, bu da piyasalar veya sistemler için kurulan modellerin sürekli güncellenmesini gerektirmektedir. Çevrede oluşan değişiklikler eğer modele parametre değişikliği ile yansıtılabiliyorsa mevcut model kullanılmaya devam edilebilmektedir. Ancak çevrede oluşan değişiklik yeni bir çıktıyı değişkeni oluşturmayı veya yeni bir amacı sağlamayı gerektirdiğinde statik modeller kullanılamaz hale gelmektedir. Bu durumda çevredeki değişikliği de içerecek şekilde yeni bir model kurulmasını gerektirmektedir. Büyük boyutlarda datanın olması durumunda yeni model kurumu zaman almakta, bu da çevrede oluşan değişikliğe tepki verme süresini uzatmaktadır. Önerilen algoritmalar, mevcut yapay sinir ağına bitiş elemanı ekleme yapması nedeniyle çevrede meydana gelen ve yeni

bir bitiş elemanı ile temsil edilmesi gereken değişikliği mevcut modele katılmasını sağlamaktadır.

8.2 Öneriler ve Gelecekteki Çalışma Alanları

Mevcut bir yapay sinir ağının üzerine yeni bitiş elemanları eklenmesi ve yeni çıktı işlevleri kazandırılması, yapay sinir ağlarının teorik olarak sonradan sınırsız bir şekilde yeni konular öğrenmesinin önünü açmaktadır. Bu gelişme, yeni işlevler kazanması nedeniyle sürekli gelişen yapay sinir ağı modellerine fırsat tanımaktadır.

İleride yapılacak çalışmalarda, önerilen algoritmaların yüksek hacimli veri (Big Data) problemlerinde güçlü ve zayıf yönleri ortaya konulabilir, meta-sezgisel algoritmalarla da karşılaştırma yapılabilir. Ayrıca yeni aday oluşturma yöntemleri geliştirilerek önerilen algoritmaların performansları arttırılmaya çalışılabilir.

Bu çalışma kapsamında önerilen algoritmalarda kullanılan K 'nın L 'li çapraz geçerliliği ile aday seçme yöntemi genişletilerek, başlangıç elemanı seçimine uygulanabilir, böylece oluşturulacak modelde kullanılacak girdi değişkenlerinin belirlenmesine yönelik yeni bir yöntem geliştirilebilir.

Bu çalışma yapay zekâ açısından değerlendirildiğinde, yapılan çalışmanın yapay zekâ çalışmalarının insanla anlaşabilen (Turing, 1950) bir yapıya “nasıl” kavuşabileceği sorusuna yanıt aradığı görülmektedir. Bu soruya verilebilecek basit bir cevap; yapay sinir ağlarının yeni sonuçlar üretebilmeleri, yeni işlevler kazanabilmeleri yani yeni konularda bilgi birikimi elde edebilmeleri gereğidir.

Yapay zeka modellerinin insana yakın düşünebilen yapıya doğru ilerleyebilmeleri için aşmaları gereken üç temel problem vardır. Bunlardan ilki, yapay zekâ modellerinin hangi yeni işlevleri veya özellikleri kazanması gerektiğidir. İkincisi, bu yeni işlev ve özellik kazanma işlemlerinin yapılması için gerekli koşulların tespitidir. Üçüncüsü ve sonuncusu ise yeni işlev ve özellik kazanma işlemlerinin ne şekilde yapılacağıdır.

Yapay zekâ çalışmalarının önünde duran bu üç temel engeli “neyi”, “ne zaman” ve “nasıl” olarak kısaca özetlenebilir. Bu çalışma “nasıl” sorusunu sorarak, üçüncü temel engelin aşılması için bir çözüm arayışı içerisindedir. İleride yapılacak çalışmalar yapay sinir ağları için “neyi” ve “ne zaman” sorularına yanıt arayabilecekleri gibi, mevcut yapay zeka modelleri için “hangi yeni işlevi”, “ne

zaman” ve “nasıl” de sorarak yeni yöntemler geliřtirmeye, yeni yaklařımlar oluřturmaya odaklanabilirler.

KAYNAKLAR

- Alavala, C. R.** (2007). *Fuzzy Logic and Neural Networks: Basic Concepts & Applications*. India: New Age International Publishers.
- Almasri, M. & Kaluarachchi, J.** (2005). Modular neural networks to predict the nitrate distribution in ground water using the on-ground nitrogen loading and recharge data, *Environmental Modelling & Software*, 20 (7), 851-871.
- Alpaydın, E.** (2004). *Introduction to Machine Learning*. London, England: The MIT Press.
- Anderson, E.** (1935). The irises of the Gaspé Peninsula, *Bulletin of the American Iris Society*, 59, 2-5.
- Anderson, E.** (1936). The species problem in Iris, *Annals of the Missouri Botanical Garden*, 23 (3), 457-509.
- Anderson, J.A.** (1972). A simple neural model generating an interactive memory, *Mathematical Biosciences*, 14, 197-220.
- Ash, T.** (1989). Dynamic node creation in backpropagation networks, *Connection Science*, 1 (4), Sf. 365-375.
- Azam, F.** (2000). *Biologically Inspired Modular Neural Networks*. Electrical and Computer Engineering. (Doktora tezi). Polytechnic Institute and State University, Virginia.
- Azimi-Sadjadi, M. R., Sheedvash, S. & Trujillo, F. O.** (1993). Recursive dynamic node creation, *IEEE Transactions on Neural Networks*, 4 (2), 242-256.
- Back, T. & Kursawe, F.** (1995). Evolutionary algorithms for fuzzy logic: A brief overview. Bouchon-Meunier, B., Ronald, R.Y., Zadeh, L.A. (Ed.). *Fuzzy logic and soft computing* (s. 3-10). Singapore: World Scientific.
- Baş, N.** (2006). *Yapay sinir ağları ve bir uygulama*. (Yüksek lisans tezi). Mimar Sinan Güzel Sanatlar Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.
- Bose, N.K. & Liang, P.** (1996). *Neural Network Fundamentals with Graphs, Algorithms and Applications*. New York: McGraw-Hill International Editions.
- Braspenning, P. J., Thuijsman, F. & Weijters, A. J. M. M.** (1995). *Artificial Neural Networks*. Almany: Springer-Verlag Berlin Heidelberg.
- Broomhead, D. S. & Lowe, D.** (1988). Multivariable functional interpolation and adaptive networks, *Complex Systems*, 2, 321-355.
- Buntine, W. L. & Weigend, A. S.** (1991). Bayesian backpropagation, *Complex Systems*, 5, 603-643.

- Burkitt, A. N.** (1991). Optimization of the architecture of feed-forward neural networks with hidden layers by unit elimination, *Complex Systems*, 5, 371-380.
- Castellano, G., Faneli, A. M. & Pelillo, M.** (1993). An empirical comparison of node pruning methods for layered feed-forward neural networks, *International Joint Conference on Neural Networks.*, (s. 321-326) Nagoya, Japan.
- Castellano, G., Faneli, A. M. & Pelillo, M.** (1997). An iterative pruning algorithm for feedforward neural networks, *IEEE Transactions on Neural Networks*, 8 (3), 519-531.
- Castillo, P. A., Merelo, J. J., Prieto, A., Rivas, V. & Romero, G.** (2000). G-Prop: Global optimization of multilayer perceptrons using Gas, *Neurocomputing*, 35, 149-163.
- Chris Tseng, H. & Almogahed, B.** (2009). Modular neural networks with applications to pattern profiling problems, *Neurocomputing*, 72 (10), 2093-2100.
- Cottrell, M., Girard, B., Girard, Y., Mangeas, M., & Muller, C.** (1995). Neural modeling for time series: a statistical stepwise method for weight elimination, *IEEE Transactions on Neural Networks*, 6 (6), 1355–1364.
- Cybenko, G.** (1989). Approximation by superpositions of a sigmoidal function, *Math. Control Signals Systems*, 2, 303-314.
- Çınar, D.** (2007). *Hidroelektrik enerji üretiminin hibrid bir model ile tahmini.* (Yüksek lisans tezi). İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.
- DeGroot, C. & Wurtz, D.** (1991). Analysis of univariate time series with connectionist nets: A case study of two classical examples., *Neurocomputing*, 3, 177–192.
- Doganis, P., Alexandridis, A., Patrinos, P. & Sarimveis, H.** (2006). Time series sales forecasting for short shelf-life food products based on artificial neural networks and evolutionary computing, *Journal of Food Engineering*, 75, 196-204.
- Donnelly, G. M., Ojha, P. C. & Bell, D. A.** (1999). Growing smaller networks with the tiling algorithm, *IJCNN '99- International Joint Conference on Neural Networks*, Washington, DC , USA
- Eberhart, R. & Shi, Y.** (2007). *Computational Intelligence*. USA: Morgan Kaufmann.
- Elmas, Ç.** (2003). *Yapay sinir ağları kuram, mimari, eğitim, uygulama*. Ankara: Seçkin Yayıncılık.
- Emir, Ş.** (2013) *Yapay Sinir Ağları ve Destek Vektör Makineleri Yöntemlerinin Sınıflandırma Performanslarının Karşılaştırılması: Borsa Endeks Yönünün Tahmini Üzerine Bir Uygulama.* (Doktora tezi). İstanbul Üniversitesi, Sosyal Bilimler Enstitüsü, İstanbul.

- Fahlman, S. E. & Lebiere, C.** (1990). The cascade-correlation learning architecture. Touretzky, D. S. (Ed.). *Advances in Neural Information Processing Systems* (s. 524–532). San Mateo, CA: Morgan Kaufmann
- Fang, W. & Lacher, R. C.** (1994). Network complexity and learning efficiency of constructive learning algorithms. *IEEE International Conference on Neural Networks* (s. 321-326). Orlando, FL: USA.
- Fanguy, R. & Bhutta, K.** (2003). Supplier Selection with the Upstart Algorithm. *Production Operations Management Society Annual Meeting*, Savannah, GA: USA.
- Fanguy, R. & Kubat, M.** (2002). Modifying Upstart for Use in Multiclass Numerical Domains. *The Florida AI Research Society Conference - FLAIRS Conference* (s. 339-343). Pensacola Beach, Florida: USA.
- Farley, B. & Clark, W.A.** (1954). Simulation of self-organizing systems by digital computer, *IRE Transactions on Information Theory*, 4, 76-84.
- Fayyad, U. M., Shapire, G. P., Smyth, P. & Uthurusamy, R.** (1996). *Advances in knowledge discovery and data mining*. Cambridge, MA: MIT Press.
- Ferentinos, K. P.** (2005). Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms, *Neural Networks*, 18, 934-950.
- Fisher, R.A.** (1936). The use of multiple measurements in taxonomic problems, *Annals of Eugenics*, 7 (2), 179–188.
- Frattale-Mascioli, F. M. & Martinelli, G.** (1995). A constructive algorithm for binary neural networks: the oil-spot algorithm, *IEEE Transactions on Neural Networks*, 6 (3), 794 - 797
- Frean, M.** (1990). The Upstart Algorithm a method for constructing and training feedforward neural networks, *Neural Computation*, 2, 198-209.
- Friedman, J. H. & Stuetzle, W.** (1981). Projection Pursuit Regression. *Journal of the American Statistical Association*, 73 (376), 817-823.
- Fritzke, B.** (1993). Growing cell structures-A self-organizing network for unsupervised and supervised learning (Rapor No. TR-93-026). International Computer Science Institution Teknik Raporu.
- Fritzke, B.** (1994a). Growing cell structures-A self-organizing network for unsupervised and supervised learning, *Neural Networks*, 7 (9), 1441-1460.
- Fritzke, B.** (1994b). Supervised Learning with Growing Cell Structures. Cowan, J. D., Tesauro, G. ve Alspector, J. (Eds.). *Advances in Neural Information Processing Systems* (s.. 255-262). San Mateo, CA, USA: Morgan Kaufmann.
- Fritzke, B.** (1995a). Growing Grid - a self - organizing network with constant neighborhood range and adaptation strength, *Neural Processing Letters*, 2 (5), 9-13.
- Fritzke, B.** (1995b). A Growing Neural Gas Network Learns Topologies. Tesauro, G., Touretzky, D. S. ve Leen, T. K. (Ed.). *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press.

- Frohlich, J.** (1997). Neural Net Overview, Eriřim: [<http://fbim.fhregensburg.de/~saj39122/jfroehl/diplom/e-1.html>], Eriřim Tarihi: 12.12.2010.
- Fukushima, K.** (1975). Cognitron: a self-organizing multilayered neural network, *Biological Cybernetics*, 20, 121-136.
- Fukushima, K.** (1980). Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biological Cybernetics*, 36, 193-202.
- Funahashi, K.-I.** (1989). On the approximate realization of continuous mappings by neural networks, *Neural Networks*, 2 (3), 183-192.
- Funahashi, K.-I.** (1990). On the Approximate Realization of Identity Mappings by 3-Layer Neural Networks, *Trans. IEICE A*, J73-A, 139-145.
- Gallant, S. I.** (1986). Optimal Linear Discriminants. *IEEE 8th Conference on Pattern Recognition*, Paris, France.
- Gallant, S. I.** (1990). Perceptron-based learning algorithms, *IEEE Transactions on Neural Networks*, 1 (2), 179-191.
- Ghiassi, M. & Burnley, C.** (2010). Measuring effectiveness of a dynamic artificial neural network algorithm for classification problems, *Expert Systems with Applications*, 37 (4), 3118-3128.
- Ghiassi, M. & Nangoy, S.** (2009). A dynamic artificial neural network model for forecasting nonlinear processes, *Computers & Industrial Engineering*, 57 (1), 287-297.
- Ghiassi, M. & Saidane, H.** (2005). A dynamic architecture for artificial neural networks, *Neurocomputing*, 63, 397-413.
- Ghiassi, M., Saidane, H. & Zimbra, D.** (2005). A dynamic artificial neural network model for forecasting time series events, *International Journal of Forecasting*, 21 (2), 341-362.
- Ghiassi, M., Zimbra, D. & Saidane, H.** (2006). Medium term system load forecasting with a dynamic artificial neural network model, *Electric Power Systems Research*, 76 (5), 302-316.
- Ghiassi, M., Zimbra, D. & Saidane, H.** (2008). Urban water demand forecasting with a dynamic artificial neural network model, *Journal of Water Resources Planning and Management*, 134 (2), 138-146.
- Gibbons, A.** (1999). *Algorithmic Graph Theory*. USA: Cambridge University Press.
- Goldberg, D.** (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.
- Gomes, G.S.S., Maia, A.L.S., Ludermir, T.B., Carvalho, F. & Araujo, A.F.R.** (2006). Hybrid model with dynamic architecture for forecasting time series, *Proceedings of the International Joint Conference on Neural Networks. IEEE Computer Society* (s. 7133-7138). Vancouver, Kanada.
- Grossberg, S.** (1988). Nonlinear neural networks: principles. mechanisms, and architectures, *Neural Networks*, 1 (1), 17-61.

- Guresen, E. & Kayakutlu, G.** (2010). Definition of Artificial Neural Networks with Comparison to Other Networks, *World Conference on Information Technology*, Istanbul.
- Guresen, E. & Kayakutlu, G.** (2011a). Artificial Neural Networks: Definition, Properties and Misuses. Flores, J.A. (Ed.). *Focus on Artificial Neural Networks* (s. 171-189). USA: Nova Publishing.
- Guresen, E. & Kayakutlu, G.** (2011b). Definition of artificial neural networks with comparison to other networks, *Procedia Computer Science*, 3, 426-433.
- Guresen, E. & Kayakutlu, G.** (2012). Topology of Constructive Neural Network Algorithms, *3rd World Conference on Information Technology (WCIT-2012)*, Barselona, İspanya
- Guresen, E. & Kayakutlu, G.** (2013a). FA01V01 algoritması: yeni bir dinamik yapay sinir ağı algoritması, *KHO Bilim Dergisi*, 23 (1), 61-98.
- Guresen, E., & Kayakutlu, G.** (2013b). Topology of Constructive Neural Network Algorithms, *AWERProcedia Information Technology & Computer Science*, 4, 375-380.
- Guresen, E., Kayakutlu, G. & Daim, T. U.** (2011). Using artificial neural network models in stock market index prediction, *Expert Systems with Applications*, 38 (8), 10389-10397.
- Gurney, K.** (2003). *An Introduction to Neural Networks*. London, UK: CRC Press.
- Gündoğdu, M.E.** (2007). *Meteorolojik parametrelerin hava kirliliğine etkilerinin yapay sinir ağları modeli ile incelenmesi*. (Tüksek lisans tezi) Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.
- Han, H.-G. & Qiao, J.-F.** (2013). A structure optimisation algorithm for feedforward neural network construction, *Neurocomputing*, 99, 347-357.
- Haykin, S.** (1994). *Neural Networks, A comprehensive Foundation*. New York, NY: Macmillan College Publishing Company.
- Haykin, S.** (1999). *Neural Networks: A Comprehensive Foundation*. New Jersey: Prentice Hall.
- Hebb, O.** (1949). *The Organization of Behavior*. New York: Wiley.
- Hinton, G.E. & Sejnowski, T.J.** (1986). Learning and relearning in Boltzmann machines. Rumelhart DE. ve McClelland J.L. (Ed.). *Parallel Distributed Processing: explorations in microstructure of cognition*. Cambridge, MA: MIT Press.
- Hopfield, J.** (1982). Neural networks and physical Systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences*, 79, 2554–2558.
- Hopfield, J.** (1984). Neurons with graded response have collective computational properties like those of two-stage neurons, *Proceedings of the National Academy of Sciences*, 81, 3088–3092.

- Hopfield, J.** (1987). Learning algorithms and probability distributions in feed-forward and feed-back networks, *Proceedings of the National Academy of Sciences*, 84, 8429–8433.
- Hopfield, J.** (1994). Neurons, dynamics and computation, *Physics Today*, 47, 40–60.
- Hopfield, J.** (1995). Pattern recognition computation using action potential timing for stimulus representation, *Nature*, 376, 33-36.
- Hornik, K.** (1991). Approximation capabilities of multilayer feedforward networks, *Neural Networks*, 4 (2), 251-257.
- Hornik, K.** (1993). Some new results on neural network approximation, *Neural Networks*, 6 (8), 1069-1072.
- Hornik, K., Stinchcombe, M., & White, H.** (1989). Multilayer feedforward networks are universal approximators, *Neural Networks*, 2 (5), 359-366.
- Hornik, K., Stinchcombe, M., & White, H.** (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, *Neural Networks*, 3 (5), 551-560.
- Hwang, J.-N., Lay, S.-R., Maechler, M., Martin, R. D. & Schimert, J.** (1994). Regression modeling in back-propagation and projection pursuit learning, *IEEE Transactions on Neural Networks*, 5 (3), 342-353.
- Iris flower data set.** (t.y.). Wikipedia. Alındığı tarih: 21.03.2014, adres: http://en.wikipedia.org/wiki/Iris_flower_data_set
- Ivakhnenko, A. G.** (1984). Self-Organizing Methods in Modeling: GMDH Type Algorithms. Farlow, S. J. (Ed.). *Statistics: Textbooks and Monographs*. New York:Marcel Dekker Inc.
- Jain, A.K. & Mao, J.** (1996). Artificial neural networks: A tutorial, *IEEE Computer Special Issue on Neural Computing*, 29 (3), 31-44.
- Karnin, E. D.** (1990). A simple procedure for pruning back-propagation trained neural networks, *IEEE Transactions on Neural Networks*, 1 (2), 239–242.
- Kayri, M.** (2009). Araştırmalarda gruplar arası farkın belirlenmesine yönelik çoklu karşılaştırma (Post-Hoc) teknikleri. *Fırat Üniversitesi Sosyal Bilimler Dergisi*, 19 (1), 51-64.
- Kennedy, J. & Eberhart, R.** (1995). Particle swarm optimization, *IEEE International Conference on Neural Networks*, Perth, Australia.
- Khorasani, K. & Weng, W.** (1994). Structure Adaptation in Feed-Forward Neural Networks. *IEEE International Conference on Neural Networks* (3, s. 1403-1408). Orlando, Florida.
- Kim, B. & Bae, J.** (2005). Prediction of plasma processes using neural network and genetic algorithms, *Solid-State Electronics*, 49, 1576-1580.
- Kiranyaz, S., Ince, T., Yildirim, A. & Gabbouj, M.** (2009). Evolutionary artificial neural networks by multi-dimensional particle swarm optimization, *Neural Networks*, 22 (10), 1448-1462.

- Kohonen, T.** (1972). Correlation matrix memories, *IEEE Trans. Computers*, 21, 353-359.
- Kohonen, T.** (1974). An adaptive associative memory principle, *IEEE Trans. Computers*, 23, 444-445.
- Kohonen, T.** (1982a). Self-Organized formation of topologically correct feature maps, *Biological Cybernetics*, 43, 59-69.
- Kohonen, T.** (1982b). Analysis of simple self-organizing process, *Biological Cybernetics*, 44, 135-140.
- Kohonen, T.** (1984). *Self-Organization and Associative Memory*. Berlin: Springer-Verlag.
- Kohonen, T.** (1988). The 'Neural' Phonetic Typewriter, *IEEE Computer Magazine*, Mart 1988, 11-22.
- Kohonen, T.** (1989). *Self-Organization and Associative Memory*. Berlin: Springer-Verlag.
- Kohonen, T.** (1990). Improved Versions of Learning Vector Quantization. *The International Joint Conference on Neural Networks*., (1, s. 545-550). San Diego, New York.
- Kohonen, T.** (1991). Self-Organizing Maps: Optimization Approaches. *International Conference on Artificial Neural Networks* (s. 981-990), Amsterdam.
- Kohonen, T.** (1993a). Things You Haven't Heard About the Self-Organizing Map. *IEEE International Conference on Neural Networks* (3, s. 1147-1156). San Francisco, New York.
- Kohonen, T.** (1993b). Physiological Interpretation of the self-organizing map algorithm, *Neural Networks*, 6 (7), 895-905.
- Kohonen, T. & Ruohonen, M.** (1973). Representation of associated data by matrix operators, *IEEE Trans. Computers*, 22, 701-702.
- Kohonen, T., Barna, G., & Chrisley, R.** (1988). Statistical Pattern Recognition with Neural Networks: Benchmarking Studies. *IEEE International Conference on Neural Networks* (1, s. 61-68). San Diego, New York.
- Kotania, M., Katsurab, M. & Ozawab, S.** (2004). Detection of gas leakage sound using modular neural networks for unknown environments, *Neurocomputing*, 62, 427-440.
- Kourakos, G. & Mantoglou, A.** (2009). Pumping optimization of coastal aquifers based on evolutionary algorithms and surrogate modular neural network models, *Advances in Water Resources*, 32 (4), 507-521.
- Koza, J.** (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.
- Kuo, R. J.** (2001). A sales forecasting system based on fuzzy neural network with initial weights generated by genetic algorithm, *European Journal of Operational Research*, 129, 496-517.
- Kwok, T.-Y. & Yeung, D.-Y.** (1997a). Constructive algorithms for structure learning in feedforward neural networks for regression problems, *IEEE Transactions on Neural Networks*, 8 (3), 630-645.

- Kwok, T.-Y. & Yeung, D.-Y.** (1997b). Objective functions for training new hidden units in constructive neural networks, *IEEE Transactions on Neural Networks*, 8 (5), 1131-1148.
- Lashley, K.** (1950). In search of the engram. *Society of Experimental Biology Symposium* (4, s. 454–482). Oxford, İngiltere.
- Lay, S.-R., Hwang, J.-N. & You, S.-S.** (1994). Extensions To Projection Pursuit Learning Networks With Parametric Smoothers. *IEEE International Conference on Neural Networks*. Orlando, Florida
- Li, Z., Cheng, G. & Qiang, X.** (2010). Some classical constructive neural networks and their new developments. *2010 International Conference on Educational and Network Technology* (s. 174-178). Qinhuangdao, China.
- Ma, L. & Khorasani, K.** (2003). A new strategy for adaptively constructing multilayer feedforward neural networks, *Neurocomputing*, 51, 361-385.
- Ma, L. & Khorasani, K.** (2004a). Facial expression recognition using constructive feedforward neural networks, *IEEE Transactions on Systems, Man, And Cybernetics—Part B: Cybernetics*, 34 (3), 1588-1595.
- Ma, L. & Khorasani, K.** (2004b). New training strategies for constructive neural networks with application to regression problems, *Neural Networks*, 17 (4), 589-609.
- Malsburg, C.** (1973). Self-organization of orientation sensitivity and columns in the visual cortex, *Kybernetik*, 14, 85-100.
- Mattson, R.L.** (1959a). *The design and analysis of an adaptive system for statistical classification*. (Yüksek lisans tezi). MIT, Boston.
- Mattson, R.L.** (1959b). A Self-Organizing Logical System. *Eastern Joint Computer Conference Record*. New York, ABD.
- McCulloch, W. ve Pitts, W.** (1943). A logical calculus of ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 5, 115-133.
- Mezard, M. & Nadal, J.-p.** (1989). Learning in feedforward layered networks: The tiling algorithm, *Journal of Physics A: Math. Gen.*, 22 (12), 2191-2203.
- Michie, D., Spiegelhalter, D.J. & Taylor, C.C.** (1994). *Machine Learning, Neural and Statistical Classification*. London: Ellis Horwood.
- Minsky, M. & Papert, S.** (1969). *Perceptrons, An Introduction to Computational Geometry*. Cambridge, MA: MIT Press.
- Molnavilaplana, J., Feliubattle, J. & Lopezcoronado, J.** (2007). A modular neural network architecture for step-wise learning of grasping tasks, *Neural Networks*, 20 (5), 631-645.
- Moody, J. & Darken, C.** (1989). Fast learning in networks of locally-tuned processing units, *Neural Computation*, 1, 281-294.
- Montgomery, D.C.** (2009). *Statistical quality control: A modern introduction*. Asia: John Wiley & Sons.

- Mozer, M. C. & Smolensky, P.** (1989). Skeletonization: a technique for trimming the fat from a network via relevance assessment. Touretzky, D. S. (Ed.). *Advances in Neural Information Processing* San Mateo, CA: Morgan Kaufman.
- Muller, B., Reinhardt, J. & Strickland, M. T.** (1995). *Neural Networks an Introduction*. Almany: Springer-Verlag Berlin Heidelberg.
- Munakata, T.** (2008). *Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More*. Londra: Springer.
- Nagabhushana, T. N., Padma, S. K. & Raghavendra Bairy, M.** (2003). Extended tower algorithm for multicategory classification. *Joint Conference of the Fourth International Conference on Information, Communications and Signal Processing* (2, s. 1182-1186). Singapore.
- Ngom, A., Stojmenovic, I. & Milutinovic, V.** (2001). STRIP—A strip-based neural-network growth algorithm for learning multiple-valued functions, *IEEE Transactions on Neural Networks*, 12 (2), 212-227.
- Niska, H., Hiltunen, T., Karppinen, A., Ruuskanen, J. & Kolehmainen, M.** (2004). Evolving the neural network model for forecasting air pollution time series, *Engineering Applications of Artificial Intelligence*, 17, Sf. 159-167.
- Nilson, N.J.** (1965). *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. New York: McGraw-Hill.
- Olmsted, D.** (1990). The reticular formation as a multi-valued logic neural network. *International Joint Conference on Neural Networks* (1, s. 619 – 624). San Diego, CA, ABD.
- Öztemel, E.** (2012). *Yapay Sinir Ağları*. İstanbul: Papatya Yayıncılık.
- Parekh, R., Yang, J. & Honavar, V.** (2000). Constructive neural-network learning algorithms for pattern classification, *IEEE Transactions on Neural Networks*, 11 (2), 436-451.
- Parker, R. E. & Tummala, M.** (1992). Identification of Volterra systems with a polynomial neural network. *IEEE International Conference on Acoustics, Speech, and Signal Processing* (s. 561–564). San Francisco, CA, ABD.
- Pelillo, M. & Fanelli, A. M.** (1993). A method of pruning layered feed-forward neural networks. Mira, J., Cabestany, J. ve Prieto, A. (Ed.). *New Trends in Neural Computation (Lecture Notes in Computer Science)* (s. 278-283).. Berlin, Springer-Verlag.
- Phatak, D. S. & Koren, I.** (1994). Connectivity and performance tradeoffs in the cascade correlation learning architecture, *IEEE Transactions on Neural Networks*, 5 (6), 1045-9227
- Platt, J.** (1991). A resource-allocating network for function interpolation, *Neural Computation*, 3 (2), 213-225.
- Principe, J.C., Euliano, N.R. & Lefebvre, W.C.** (1999). *Neural and Adaptive Systems: Fundamentals Through Simulations*. New York: John Wiley & Sons.

- Puma-Villanueva, W. J., dos Santos, E. P. & Von Zuben, F. J.** (2012). A constructive algorithm to synthesize arbitrarily connected feedforward neural networks, *Neurocomputing*, 75 (1), 14-32.
- Qiang, X., Cheng, G. & Wang, Z.** (2010). An Overview of Some Classical Growing Neural Networks and New Developments. *2nd International Conference on Education Technology and Computer* (3, s. 351-355) Shanghai, Çin.
- Reilly, D. L., Cooper, L. N. & Elbaum, C.** (1982). A neural model for category learning, *Biological Cybernetics*, 45, 35-41.
- Rojas, R.** (1996). *Neural Networks: A Systematic Introduction*. Almanya: Springer-Verlag Berlin Heidelberg.
- Rosenblatt, F.** (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Washington D.C.: Spartan books.
- Rumelhart, D.E., Hinton, D.E. & Williams, R.J.** (1986). Learning representation by backpropagating errors, *Nature*, 323 (9), 533-536.
- Sagiroğlu S., Güney, K.** (1997). Calculation of resonant frequency for an equilateral triangular microstrip antenna using artificial neural networks, *Microwave Opt. Technol. Lett.*, 14 (2), 89-93.
- Sagiroğlu, Ş., Beşdok, E., Erler, M.** (2004). *Mühendislikte Yapay Zeka uygulamaları I :Yapay Sinir ağları* , Kayseri: Ufuk Kitap Kirtasiye.
- Saha, A., Wu, C.-L. & Tang, D.-S.** (1993). Approximation, dimension reduction, and nonconvex optimization using linear superpositions of Gaussians, *IEEE Transactions on Computers*, 42 (10), 1222-1233.
- Setiono, R.** (2001). Feedforward neural network construction using cross validation, *Neural Computation*, 13, 2865-2877.
- Sexton, R. S., Dorsey, R. E. & Sikander, N. A.** (2004). Simultaneous optimization of neural network function and architecture algorithm, *Decision Support Systems*, 36, 283-296.
- Sharkey, A. J.** (1996). On combining artificial neural networks, *Connection Science*, 8 (3), 299-313.
- Sharma, S. K. & Chandra, P.** (2010). Constructive neural networks: A review, *International Journal of Engineering Science and Technology*, 2 (12), 7847-7855.
- Shin, Y. & Ghosh, J.** (1995). Ridge polynomial networks, *IEEE Transactions on Neural Networks*, 6 (3), 610-622.
- Sietsma, J. & Dow, R. J. F.** (1988). Neural net pruning - Why and how. *International Conference on Neural Networks* (s. 325-333.). San Diego, CA, ABD.
- Sjogaard, S.** (1992). Generalization in cascade-correlation networks. *Neural Networks for Signal Processing* (s. 59 - 68). Helsingoer, Danimarka.
- Specht, D.F.** (1988). Probabilistic neural networks for classification mapping, or associative memory. *Proceedings of IEEE International Conference on Neural Networks* (1, s. 525-532). San Diego, CA, ABD.

- Specht, D.F.** (1991). A general regression neural network, *IEEE Transactions on Neural Networks*, 2, 568-576.
- Steinbuch, K.** (1961). Die lernmatrix, *Kybernetik*, 1, 36-45.
- Subirats, J.L., Franco, L. & Jerez, J.M.** (2012). C-Mantec: A novel constructive neural network algorithm incorporating competition between neurons, *Neural Networks*, 26, 130–140.
- Şen, Z.** 2004. *Yapay Sinir Ağları ve İlkeleri*. İstanbul: Su Vakfı Yayınları.
- Tenorio, M.F. & Lee, W.T.** (1990). Self-organizing network for optimum supervised learning, *IEEE Transactions on Neural Networks*, 1 (1), 100-110.
- Thodberg, H.H.** (1996). A review of Bayesian neural networks with an application to near infrared spectroscopy, *IEEE Transactions on Neural Networks*, 7, 56-72.
- Turing, A.M.** (1950). Computing machinery and intelligence, *Mind*, 49, 433-460.
- Velásquez, J.D. & Franco, C.J.** (2010). Prediction of the prices of electricity contracts using a neural network with dynamic architecture, *Innovar*, 20 (36), 7-14.
- Velásquez, J.D., Franco, C.J. & Olaya-Morales, Y.** (2012). A Review of DAN2 (Dynamic architecture for artificial neural networks) model in time series forecasting, *Ingeniería y Universidad*, 16 (1), 135-146.
- Verkooijen, W. & Daniels, H.** (1994). Connectionist projection pursuit regression, *Computational Economics*, 7, 155-161.
- Vural, B.** (2007). *Yapay Sinir Ağları ile Finansal Tahmin*. (Yüksek lisans tezi). Ankara Üniversitesi, Ankara.
- Walpole, R.E., Myers, R.H., Myers, S.L. & Ye, K.** (2007). *Probability & statistics for engineers & scientists*. New Jersey: Pearson Prentice Hall.
- Wang, D.** (2008). Fast Constructive-Covering Algorithm for neural networks and its implement in classification, *Applied Soft Computing*, 8 (1), 166-173.
- Wang, T. & Huang, C.** (2007). Applying optimized BPN to a chaotic time series problem, *Expert Systems with Applications*, 32, 193-200.
- Wang, J., Niu, D. & Li, L.** (2010). Middle-long term load forecasting based on dynamic architecture for artificial neural network, *Journal of Information and Computational Science*, 7 (8), 1711-1717.
- Werbos, P.J.** (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. (Doktora tezi), Harvard University, Cambridge, MA.
- Werbos, P.J.** (1988). Generalization of backpropagation with application to a recurrent gas market model, *Neural Networks*, 1, 140-151.
- Werbos, P.J.** (1990). Backpropagation through time: what it does and how to do it, *Proceedings of the IEEE*, 78 (10), 1550-1560.

- Specht, D.F.** (1991). A general regression neural network, *IEEE Transactions on Neural Networks*, 2, 568-576.
- Steinbuch, K.** (1961). Die lernmatrix, *Kybernetik*, 1, 36-45.
- Subirats, J.L., Franco, L. & Jerez, J.M.** (2012). C-Mantec: A novel constructive neural network algorithm incorporating competition between neurons, *Neural Networks*, 26, 130–140.
- Şen, Z.** 2004. *Yapay Sinir Ağları ve İlkeleri*. İstanbul: Su Vakfı Yayınları.
- Tenorio, M.F. & Lee, W.T.** (1990). Self-organizing network for optimum supervised learning, *IEEE Transactions on Neural Networks*, 1 (1), 100-110.
- Thodberg, H.H.** (1996). A review of Bayesian neural networks with an application to near infrared spectroscopy, *IEEE Transactions on Neural Networks*, 7, 56-72.
- Turing, A.M.** (1950). Computing machinery and intelligence, *Mind*, 49, 433-460.
- Velásquez, J.D. & Franco, C.J.** (2010). Prediction of the prices of electricity contracts using a neural network with dynamic architecture, *Innovar*, 20 (36), 7-14.
- Velásquez, J.D., Franco, C.J. & Olaya-Morales, Y.** (2012). A Review of DAN2 (Dynamic architecture for artificial neural networks) model in time series forecasting, *Ingeniería y Universidad*, 16 (1), 135-146.
- Verkooijen, W. & Daniels, H.** (1994). Connectionist projection pursuit regression, *Computational Economics*, 7, 155-161.
- Vural, B.** (2007). *Yapay Sinir Ağları ile Finansal Tahmin*. (Yüksek lisans tezi). Ankara Üniversitesi, Ankara.
- Walpole, R.E., Myers, R.H., Myers, S.L. & Ye, K.** (2007). *Probability & statistics for engineers & scientists*. New Jersey: Pearson Prentice Hall.
- Wang, D.** (2008). Fast Constructive-Covering Algorithm for neural networks and its implement in classification, *Applied Soft Computing*, 8 (1), 166-173.
- Wang, T. & Huang, C.** (2007). Applying optimized BPN to a chaotic time series problem, *Expert Systems with Applications*, 32, 193-200.
- Wang, J., Niu, D. & Li, L.** (2010). Middle-long term load forecasting based on dynamic architecture for artificial neural network, *Journal of Information and Computational Science*, 7 (8), 1711-1717.
- Werbos, P.J.** (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. (Doktora tezi), Harvard University, Cambridge, MA.
- Werbos, P.J.** (1988). Generalization of backpropagation with application to a recurrent gas market model, *Neural Networks*, 1, 140-151.
- Werbos, P.J.** (1990). Backpropagation through time: what it does and how to do it, *Proceedings of the IEEE*, 78 (10), 1550-1560.

- Werbos, P.J.** (1994). *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and political forecasting*. New York: John Wiley & Sons.
- Widrow, Bernard, and Hoff, Marcian, E.** (1960). *Adaptive Switching Circuits*. Convention Record, New York: IRE WESCON.
- Wolf number.** (t.y.). *Wikipedia*. Alındığı tarih: 21.03.2014, adres: http://en.wikipedia.org/wiki/Wolf_number
- Yakut, E.** (2012). *Veri Madenciliği Tekniklerinden C5.0 Algoritması Ve Destek Vektör Makineleri ile Yapay Sinir Ağlarının Sınıflandırma Başarılarının Karşılaştırılması: İmalat Sektöründe Bir Uygulama*. (Doktora tezi), Atatürk Üniversitesi: Erzurum.
- Yang, S.-H. & Chen, Y.-P.** (2012). An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications, *Neurocomputing*, 86, 140-149.
- Yi-Hui, L.** (2007). Evolutionary neural network modeling for forecasting the field failure data of repairable systems, *Expert Systems with Applications*, 33, 1090-1096.
- Young, S. & Downs, T.** (1998). CARVE - A constructive algorithm for real valued examples, *IEEE Transactions on Neural Networks*, 9 (6), 1180-1190.
- Zhang, B.-T.** (1994). An incremental learning algorithm that optimizes network size and sample size in one trial. *IEEE World Congress on Neural Networks* (1, s. 215-220). Orlando, Florida.
- Zhang, G.P.** (2003). Time series forecasting using a hybrid ARIMA and neuralnetwork model, *Neurocomputing*, 50, 159–175.
- Zhao, Z.-Q.** (2009). A novel modular neural network for imbalanced classification problems, *Pattern Recognition Letters*, 30 (9), 783-788.
- Zhao, Z.-Q., Gao, J., Glotin, H. & Wu, X.** (2010). A matrix modular neural network based on task decomposition with subspace division by adaptive affinity propagation clustering, *Applied Mathematical Modelling*, 34 (12), 3884-3895.
- Url-1** <http://en.wikipedia.org/wiki/Iris_flower_data_set>, alındığı tarih: 21.03.2014



ÖZGEÇMİŞ

Ad Soyad: Erkam GÜREŞEN
Doğum Yeri ve Tarihi: Balıkesir, 06.06.1984
E-Posta: erkamguresen@gmail.com
Lisans: Yıldız Teknik Üniversitesi
Yüksek Lisans: İstanbul Teknik Üniversitesi

Mesleki Deneyim ve Ödüller:

Öğretim Görevlisi; Mart 2010 - Temmuz 2011, Okan Üniversitesi, İstanbul.
Öğretim Görevlisi; Mart 2012 - ..., Kara Harp Okulu, Ankara.

Yayın Listesi:

- **Sevim, C., Oztekin, A., Bali, O., Gumus, S., & Guresen, E.** (2014). Developing an early warning system to predict currency crises exemplified in the Turkish Economy, *European Journal of Operational Research*, 237 (3) 1095-1104.
- **Bali, O., & Guresen, E.** (2013). Çoklu periyotta çevreci tedarikçi seçimi için belirsizlik etmenli bir ÇÖKV yöntemi, *The Journal of Defense Sciences*, 12 (1), 43-70.
- **Aktel, A., & Guresen, E.** (2013). Automatically forecasting magazine demands and modeling seasonality, *AWERProcedia Information Technology & Computer Science*, 4, 370-374.
- **Sevim, Cuneyt, Bali, Ozkan, Gumus, Serkan, & Guresen, E.** (2013). Developing an early warning system for currency crises in Turkey *AWERProcedia Information Technology & Computer Science*, 4, 509-515.

- **Aktel, A., & Guresen, E.** (2012). Automatically forecasting magazine demands and modeling seasonality *3rd World Conference on Information Technology (WCIT-2012)*, Barselona..
- **Sevim, Cuneyt, Bali, Ozkan, Gumus, Serkan, & Guresen, E.** (2012). Developing an early warning system for currency crises in Turkey, *3rd World Conference on Information Technology (WCIT-2012)*, Barselona: İspanya.
- **Guresen, E., Kayakutlu, G., & Daim, T.U.** (2011). Using artificial neural network models in stock market index prediction, *Expert Systems with Applications*, 38 (8). 10389-10397.
- **Onden, I. & Guresen, E.** (2011). Interior Design of Passenger Coaches Using Fuzzy Optimization, *Procedia Computer Science*, 3. 491-98.
- **Onden, I. & Guresen, E.** (2010). Interior design of passenger coaches using fuzzy optimization, *World Conference on Information Technology*, Istanbul.
- **Guresen, E. & Kayakutlu, G.** (2008). Forecasting stock exchange movements using artificial neural network models and hybrid models, *5th International Conference on Intelligent Information Processing*, Pekin: Çin.
- **Guresen, E. & Kayakutlu, G.** (2008). Forecasting Stock Exchange Movements Using Artificial Neural Network Models and Hybrid Models. Zhongzhi Shi, E. Mercier-Laurent ve D. Leake (Ed.). *Intelligent Information Processing Iv*, (s. 129-137). Boston: Springer.
- **Guresen, E. & Kayakutlu, G.** (2008). Evaluating artificial neural network approaches used for time series forecasting, *Inform's Annual Meeting*, Washington, USA.
- **Cebeci, U., Güreşen, E. & Ekizoğlu, B.** (2007). İş zekası yazılımı seçimine derecelendirme yöntemi ile analitik ağ süreci yaklaşımı, *XXVII. YA/EM Ulusal Kongresi*, İzmir: Türkiye.

TEZDEN TÜRETİLEN YAYINLAR/SUNUMLAR

- **Guresen, E. & Kayakutlu, G.** (2013). FA01V01 algoritması: yeni bir dinamik yapay sinir ağı algoritması, *KHO Bilim Dergisi*, 23 (1). 61-98.
- **Guresen, E., & Kayakutlu, G.** (2013). Topology of Constructive Neural Network Algorithms, *AWERProcedia Information Technology & Computer Science*, 4. 375-380.

- **Guresen, E. & Kayakutlu, G.** (2012). Topology of Constructive Neural Network Algorithms, *3rd World Conference on Information Technology (WCIT-2012)*, Barselona: İspanya.
- **Guresen, E. & Kayakutlu, G.** (2011). Artificial Neural Networks: Definition, Properties and Misuses. Flores J.A. (Ed.), *Focus on Artificial Neural Networks*. (s. 171-189). New York: Nova Publishing.
- **Guresen, E. & Kayakutlu, G.** (2011). Definition of artificial neural networks with comparison to other networks, *Procedia Computer Science*, 3, 426-33.
- **Guresen, E. & Kayakutlu, G.** (2010). Definition of artificial neural networks with comparison to other networks, *World Conferance on Information Technology*, Istanbul: Türkiye.

