

1. Giriş

Yapay sinir ağlarının nasıl genetik algoritmalar ile evrimleştiğini bilmek için, önce Genetik algoritmaların ve yapay sinir ağlarının ne olduğunu bilmek gereklidir. Genetik Algoritmalar altında evolution.js kütüphanesinin nasıl çalıştığını, Yapay sinir Ağlarının altında ANN.js kütüphanesinin nasıl çalıştığını anlattım. Bu kütüphaneleri yazabilecek ve anlayabilcek kadar bilgi verdikten sonra, nöroevrim hakkında bilgi verip, XOR probleminin nöroevrim ile nasıl çözülebileceğini gösterdim. Ardından gerçek zamanlı bir problem olan kartal.io'da, yapay canlıların hayatlarını simüle ettim.

2. Genetik Algoritma

2.1 Tanımlar

Genetik algoritmalar doğadaki evrimsel süreci taklit eden optimizasyon yöntemleridir. Her problemde en iyi çözümü bulması zorunlu değildir. Fakat amacı en iyi veya en iyiye yakın çözümü bulmaktır. En iyilerin hayatı kalması şartına göre çalışır ve bu "hayatta kalmak" amacı, "en iyi çözüme uygun olmak" amacıyla değiştirelerek birçok probleme uygulanabilir. Bu kavrama "Fitness" veya "Uygunluk" diyebiliriz.

Yine doğada nasıl tek canlı yani tek çözüm yoksa, genetik algoritmalar da tek çözüm yoktur. Tüm çözümlerin ortak kümesine Popülasyon denilir. Kötü çözümlerin yok olma zorunluluğu yoktur. Kötü çözümlerin ne kadar yok olma ihtimali yüksek olsada, bir çok genetik yöntemde, kötü çözümler bizi ilerde iyi bir sonuca götürebileceği için tamamen yok edilmezler.

Genetik algoritmalarda her bir çözüm Kromozomlarda tutulur ve her bir çözüm değişkenine Gen denir. Onun için "çözüm" kelimesi yerine Birey kelimesini kullanacağız. Her kromozom sahibi olan ve Popülasyonun üyesi olan şeye 'Birey(Member)' denilir. Her birey üreyip, çocuklar oluşturabilir. Bu durumda o birey, çocukların ebeveyni olur. Dolayısıyla ilk ebeveynler için bir başlangıç populasyonu olması gereklidir. Genelde başlangıç populasyonu rastgele oluşturulur.

Genetik algoritmalar 3 temel kavram üzerine kuruludur ; Mutasyon, Çaprazlama ve Seçim.

Bir bireyin genleri çocuklarına aktarılırken, mutasyonlar oluşabilir. Mutasyonlar, gen aktarımında oluşan hatalardır. Bilgisayarda, bu hatayı doğadaki gibi taklit etmek için rastgelelik fonksiyonları kullanılır. Mutasyon, doğadaki gibi çoğu sefer zararlı olabilir. Mutasyon sonucu oluşan zararlı özellik o canlıının hayatı kalma ihtimalini (uygunluğunu) düşürecektir. Fakat bazı zamanlar, mutasyonlar canlıya faydalı olabilir. Bu faydalı özellikler canlıının hayatı kalma ihtimali artıracak ve diğerlerine karşı o

canlıyı üstün kılacaktır. Onun için mutasyondan kötü etkilenen canlı zamanla yok olur, iyi etkilenen ise büyük avantajlar sağlayarak, kendini çoğaltır.

Çaprazlama, 2 canlıının eşyeli üreyerek kendi kromozomlarını tek bir kromozoma naklederek, kendilerinde farklı bir çocuk oluşturmasıdır. Çocuk, ebeveynlerinin özelliklerini taşıır. Ebeveynlerini kısmen benzer. Bunun sebebi, iki ebeveynin kromozomlarının bir veya birkaç noktadan kesilerek, iki taraftan rastgele alınan genler ile bir kromozom oluşturulmasıdır.

Doğal seçim, hayatı kalma ihtimali az olan canlıların birçoğunun ölüp, hayatı kalma ihtimali yüksek olanlarının birçoğunun hayatı kalmasıdır. Eğer doğada bir elitizim(en iyinin ölmemesi) durumu yoksa, hayatı kalma ihtimali yüksek olanlarında yine ölüm ihtimali vardır. Bilgisayarda seçim, bireyin uygunluk fonksiyonu gözetilerek rastgele yapılır. Doğanın zor koşulları ve ön görülemezliği, rastgelelik fonksiyonları ile taklit edilir.

Bu algoritmalar ilk kez 1975 yılında John Holland'in "Adaptation in Natural and Artificial Systems" [1], kitabında ortaya atılmıştır [2].

2.1.1 Genetik Teknik

1. $P(0)$ başlangıç populasyonunu oluştur. ($t=0$)
2. $P(t)$ 'yi değerlendir
3. Durdurma kriterini kontrol et.
4. $P(t)$ 'nin bireylerini seç.
5. $P(t)$ 'nin seçilmiş bireylerini değiştirerek $P(t+1)$ populasyonunu oluştur. ($t=t+1$)
6. Adım 2'ye geç [3].

2.1.2 GA (Genetik Algoritma)

Genetik algoritma hakkında bilgilerin saklandığı objedir.

Değişken	Açıklama
defaultParameters{}	Her Evolution sınıfın varsayılan parametreleridir
charSet[]	Bir String'de kullanılabilen karakterlerin listesidir
TYPE{}	Gen türlerin saklandığı ENUM BIT, INT, UNIPOLAR, BIPOLAR, STRING , CHROMOSOME
CO_TYPE{}	Çaprazlama yönteminin saklandığı ENUM PARTIALY, MULTIPARTIALY, UNIFORM
SELECTION{}	Seçim yönteminin saklandığı ENUM ROULETTE, SORN
ALGORITHMS{}	Algoritmaların saklandığı ENUM STANDART, DIEANDBORN

parameters{}	Açıklama
crossing_over	Çaprazlamanın aktif olma durumu
crossing_overRate	Çaprazlama ihtimali(UNIFORM türü CO için)
mutation_rate	Mutasyon ihtimali
population_size	Populasyon büyüğü (önerilen 100-300)
iteration	Kaç jenerasyon üretileceği
algorithm	Algoritmanın türü. ENUM
crossing_overMethod	Çaprazlama Algoritması. ENUM : Varsayılan: MULTIPARTIALLY
selectionMethod	Seçim Algoritması. ENUM : Varsayılan: ROULETTE

Fonksiyon	Amaç
random()	0-1 arası rastgele sayı üreten fonksiyondur.
randomINT(min,max)	min,max aralığında tam sayı üreten fonksiyondur
chance(rate)	rate ihtimalini gerçekleyen ve buna göre boolean:true/false döndüren fonksiyon
copyGene(oldGene)	Bir genin aynen kopyalayıp,yeni gen üreten fonksiyondur
INS(geneArray)	Kromozomdaki gen sayısını, herhangi bir geni kopyalayarak arttırır
RMV(geneArray)	Kromozomdaki herhangi bir geni siler
SWP(geneArray)	Kromozomdaki herhangi iki geni yer değiştirir
splitToParts(chromosome)	Kromozomu ağaç gibi açar. Çaprazlama için uygun ağacın en altındaki genleri, GenePart'a çevirir.
crossingOverable(partsA,partsB)	İki Gen Parçası(GenePart) dizisininin, çaprazlama için uygun olanlarını ayıklar.
crossingOver(chromA,chromoB,CO_TYPE,E,PARTNUM_OR_UNIFORMRATE)	İki kromozomu çaprazlar.
crossingOverRULED(chromA,chromoB,CO_TYPE,PARTNUM_OR_UNIFORMRATE)	İki kromozomu çaprazlar ve çaprazlama sonucu oluşan kromozomlardaki hataları onarır.(mutasyon sonrasında aynı işlem yapıldığı için crossingOver fonksiyonu tercih edilir)

2.1.3 Evolution (Evrim)

Altında populasyonları bulunduran. En üst sınıfır.

Değişken	Açıklama
populations[]	Populasyonlar
parameters {}	Evrimsel parametreler
fitnessFunction(member)	fitness fonksiyonu
createPopulation(population_size)	Rastgele başlangıç populasyonu oluşturma fonksiyonu

Fonksiyon	Amaç
setParameters(param)	Parametreleri değiştirir
start()	Evrimsel Algoritmayı çalıştırır

2.1.4 Population (Populasyon)

Member'ların dizisidir. Population'un fitness fonksiyonu ve Memberların en iyi ve kötüsü burada belirlenip, sıralanır.

Değişken	Açıklama
members[]	Population'un üyelerinin
evolution	Evolution üst sınıfı
bestMember	En yüksek fitness değerine sahip üyedir
avgFitness	Ortalama fitness değeri
minFitness	Minimum fitness değeri
maxFitness	Maximum fitness değeri
totalFitness	Toplam fitness değeri
lastMemberId	Son bireyin Id'si saklanır

Fonksiyon	Amaç
calcFitness()	Populasyonun fitness değerlerini hesaplar
select()	Seçilim Yapar (bir adet)
rate()	Her üyenin seçim ihtimalini belirler
selection()	Seçilim yapılarak yeni jenerasyon oluşturulur
crossing_over()	Bütün bireyler kendi aralarında çaprazlamaya uğratılır

Fonksiyon	Amaç
mutation()	Bütün bireyler mutasyona uğratılır

2.1.5 Member (Birey)

Population'un her bir üyesine verilen addır Chromosome'a sahiptir

Değişken	Açıklama
int generation	Bireyin kaçinci nesil olduğunu saklar
num fitness	Bireyin fitness değeri
num id	Bireyin Idsi
chromosome	Bireyin kromozomu
population	Bireyin bağlı olduğu populasyon

Fonksiyon	Amaç
kill()	Üye öldürülür
generate()	Üye ürer

2.1.6 Fitness (Uygunluk)

Bir Memberin amacına ne kadar uygun olduğunu belirleyen fonksiyondur. Mesele : Hayatta kalmak ve Üreyebilmek Yüksek olması Memberin, soyunun devam edebilme ihtimalini artttırır.

2.1.7 Chromosome (Kromozom)

Gene'lerden oluşan bir dizidir.Ancak kendisi bir gen türüdür. Yani bir kromozomun, alt kromozomları olabilir. Bu sayede, daha kolay bir şekilde çözümü genetik olarak ifade edebiliyoruz.

2.1.8 Gene (Gen)

Member'in yapısı belirleyen her bir birime denir.

Gen Türü	Açıklama
bit	0 -> 1 [0 veya 1]
int	min-max arası tam sayıdır
unipolar	0 -> 1 0.3->0.5 [0 ile 1 arası değer]
bipolar	-1 -> 1 -0.3->0.5 [-1 ile 1 arası değer]
string	"abc" -> "adef"
chromosome	başka genlerden oluşan, gen dizisi

Değişken	Açıklama
val	Genin Türüne göre aldığı değerdir.
type	Genin Türüdür.
chg	CHANGE türü mutasyon geçirebilme özellikinin açık olmasını saklanır.
ins	INSERT türü mutasyon geçirebilme özellikinin açık olmasını saklanır.
rmv	REMOVE türü mutasyon geçirebilme özellikinin açık olmasını saklanır.
swp	SWAP türü mutasyon geçirebilme özellikinin açık olmasını saklanır.
mutation_rate	Genin mutasyon geçirme ihmalidir.
chg_rate	STRING her karakterinin CHANGE türü mutasyon geçirme ihmalidir. [sadece Stringde kullanılır]
ins_rate	Genin INSERT türü mutasyon geçirme ihmalidir.
rmv_rate	Genin REMOVE türü mutasyon geçirme ihmalidir.
swp_rate	Genin SWAP türü mutasyon geçirme ihmalidir.
min	INT tipi genler için minimum alınabilecek değer saklanır
max	INT tipi genler için maximum

Değişken	Açıklama
	alınabilecek değer saklanır
Fonksiyon	Amaç
mutate()	mutasyona uğrar

2.1.9 Başlangıç Populasyonu

Populasyon büyüklüğüne bağlı olarak ilk oluşan rastgele populasyondur.

2.1.10 Fitness Fonksiyonu (Uygunluk Fonksiyonu)

Bireyin ne kadar uygun olduğunu belirleyen değeri üreten fonksiyondur. Uygun populasyon üyesi daha fazla soyunu devam ettirmeye meyllidir. Bu fonksiyonun döndürdüğü değerin yüksek olması canlıının, soyunu devam ettirme ihtimalini arttırmır.

2.1.11 Çaprazlama

2 Kromozomun genlerini kaynaştırılması ile yeni atalarına benzer bireyler üretmektedir.

- **[PARTIALLY] Partially Crossing Over:** Genlerden bir nokta seçilir. O Noktadan bölünürek parçalar değiştirilir.
- **[MULTIPARTIALLY] Multi-Partially Crossing Over:** Gen çok noktadan parçalanarak, parçalar yer değiştirir.
- **[UNIFORM] Uniform Crossing Over:** Her gen, belli bir olasılıkta diğer genle yer değiştirir.
- **Aritmetic Crossing Over :** AND, OR, XOR ile yapılan takastır. $X \text{ AND } Y = 0$ $\text{AND } 1 = 0$ (Bu yöntem projede kullanılmayacaktır)

2.1.12 Mutasyon

Her gen, alabileceği bir değer ile değiştirilir veya o miktarda arttırılıp, azaltılır.

- **[CHG] CHANGING(DEĞİŞTİRME)** : Genin değeri değişir.
- **[INS] INSERTION(EKLEME)** : Kromozoma Gen eklenir
- **[RMV] REMOVING (ÇIKARMA)**: Kromozamdan gen çıkartılır.
- **[SWP] SWAPING (YER DEĞİŞTİRME)** : İki genin yerleri değişir.

2.1.13 Seçilim

- **[ROULETTE] Rulet seçimi** : Tüm bireylerin uygunluk değerleri bir tabloya yazılır, sonra bu uygunluk değeri toplam uygunluk değerine bölünerek, olasılıklar belirlenir.
- **[SORT] Sıralama seçimi** : Rulet seçiminde, eğer çok yüksek uygunluğa sahip birey varsa , diğerlerinin seçim ihtimali imkansızlaşmaktadır. Bu da ileri vadede bir sorundur. Onun için uygunluk değerine göre değilde, sıralama yapılarak kaçinci sırada olduğuna göre bir seçim yapılır. Yani uygunluk değeri 1 ile (Birey Sayısı) arasında olur.
- **Sabit durum seçimi** : Buna göre, ebeveyn seçimi için kromozomların büyük parçaları bir sonraki nesile taşınmalıdır. Yeni döl oluşturulma üzere birkaç kromozom seçilir.(Genellikle en yüksek uygunluğa sahip olanlar seçilir.)
- **Elitizm(Seçkinlik)** : En iyi birey(ler) bozulmadan bir sonraki jenerasyona kopyalanır. (Gerçek zamanlı algoritmalarla kullanılamaz.)

2.1.14 Gerçek Zamanlılık

Eğer evrimsel süreç gerçek zamanlıysa, bireyin ölmesi ve üremesi durumları göz önüne alınır. Evrimsel algoritma, adım adım olarak değil, asenkron şekilde çalışır. Yani 1.nesil birey ile 3.nesil birey aynı populasyonda bulunabilir.

2.2 Teknik Gereksinimler

- **Probleminin Genetik Temsili** : Her bir çözümün genlere aktarılabilmesi
- **Değerlendirme** : Her bir çözümün uygunluğunu değerlendiren bir fonksiyon ; Fitness/Uygunluk fonksiyonu
- **Başlangıç populasyonu oluşturma yöntemi** : Başlangıç populasyonun belirlenmesinde yöntemler. Örneğin : Rastgele genler oluşturma.
- **Genetik kompozisyon yöntemleri** : Bir sonraki populasyonun kromozomları oluşturacak teknikler. Örneğin : Mutasyon ve Çaprazlama

2.3 Basit Algoritma

1. Rastgele başlangıç populasyonu üret
2. Populasyonu değerlendir
3. Maximum nesile ulaşıldıysa ; Adım 8'e git
4. Seçilim yap
5. Çaprazlama yap
6. Mutasyona uğrat
7. Adım 2'ye git
8. Dur

2.4 "Her ölüm bir yeniden doğumdur " Gerçek Zamanlı Algoritması

1. Rastgele başlangıç populasyonu üret
2. Bir üyenin ölmesini bekle => 3.Adıma geç, ve Sonlanmayı bekle => 10.adıma geç
3. Populasyonu değerlendir
4. 2 birey için seçim yap
5. 2 bireyi çaprazlamaya uğrat ve yeni birey oluştur.
6. Yeni bireyi mutasyona uğrat.
7. Ölen bireyi, populasyondan sil.
8. Yeni bireyi populasyona ekle.
9. 2.adıma geç
10. Dur

3. Yapay Sinir Ağları

İnsan beyninin özellikleri taklit edilerek geliştirilmiş bir bilgi işleme tekniğidir. Sinir ağları, sinirlerden ve sinirler arası bağlantılarından oluşur.

NOT: Buradaki bilgilerden yararlanılarak ANN.js kütüphanesi yazıldı.

3.1 Network (Sinir Ağları)

Katmanlardan ve sinirlerden oluşan sinir ağının yapısıdır.

Network (ANN) Sınıfı

Değişken	Açıklama
layers[]	Ağın katmanlarının dizisidir.
TYPE	Ağın mimari tipidir

Fonksiyon	Amaç
fire([inputArray])	Ağın girdilerinden(inputArray içindeki veya girdi katmanındaki çıkış değerlerinden) çıktı üretmek için ateşler
inputLayer()	Girdi katmanını döndürür
outputLayer()	Çıktı katmanını döndürür
getOutputs()	Son katmanın çıkış değerlerini döndürür.
setOutputs(outputs)	Girdi katmanının outputlarını günceller.

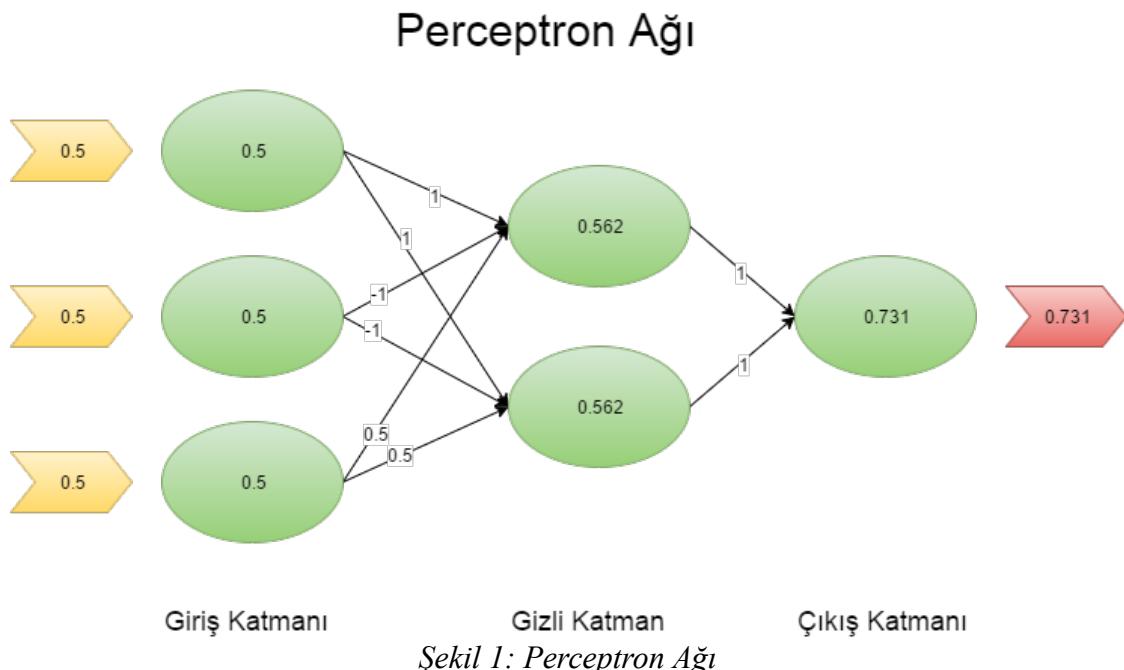
3.1.1 Ağ mimarileri

3.1.1.1 Perceptron

Sıralı katmanlardan oluşan, girdi ,gizli ve çıktı katmanı sahibi çok katmanlı sinir ağı mimarisidir.

Fonksiyon	Amaç
setWeights(wArray)	w : Katmanların ağırlık matrislerinin dizisidir. Bu fonksiyon ağırlıkları günceller.

3.1.1.1 Perceptronun Javascriptte Oluşturulması ve Ateşlenmesi



```
// @Gerekli : ANN.js
```

```
// # Giriş katmanı 3, Gizli katmanı 2, Çıkış katmanı 1 sinirden oluşan perceptron ağı oluştur ;
var YSA = new ANN().PERCEPTRON(3,[2],1);
```

```
// # 2 adet ağırlık matrisi ile ağı set et ;
```

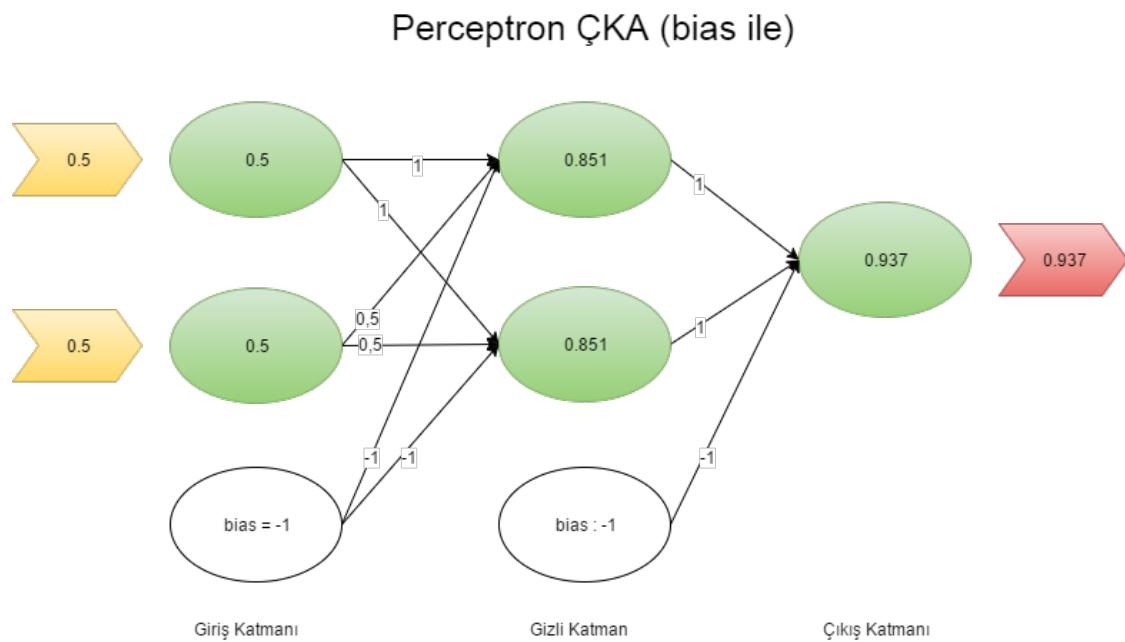
```
YSA.setWeights([
  [
    [1,1],
    [-1,-1],
    [0.5,0.5]
  ],
  [
    [1],
    [1]
  ]
]);
```

```
// # [0.5,0.5,0.5] girdi vektörü ile ileri beslemeli olarak ateşle.
```

```
YSA.fire([0.5,0.5,0.5]);
```

```
console.log(YSA.getOutputs()); // # çıktı : [ 0.7547952605810734 ]
```

Bias ile



```
var YSA = new ANN().PERCEPTRON(2,[2],1).setBias( [-1,-1] );
```

```

YSA.setWeights([
    [
        [1,1],
        [0.5,0.5],
        [-1,-1]
    ],
    [
        [1],
        [1],
        [-1]
    ]
]);
```

```

YSA.fire([0.5,0.5]);
console.log(YSA.getOutputs()); // # çıktı : [ 0.9372567117128635 ]
```

3.2 Layer (Katman)

Sinir hücrelerinden oluşan gruplardır.

Değişken	Açıklama
neurons[]	Katmana bağlı sinirlerin dizisidir.
dentriteLayers[]	Bu katmanın solundaki katmanların bağlantılarının dizisidir. Bunlar kendisinden önce ateşlenecek katmanlardır.
axonLayers[]	Bu katmanın sağındaki katmanların bağlantılarının dizisidir. Bunlar kendisinden sonra ateşlenecek katmandır

Fonksiyon	Amaç
connect(layer)	Bu katmayı diğer katmana bağlar.
connectWithNeurons(layer)	Bu katmayı diğer katmana bağlar. İki katmanın sinirlerini birbirine bağlar.
fire()	Katmandaki bütün sinirleri ateşler.
fireConsecutive()	Bu katmayı ve buna bağlı bütün katmanlı ardarda ateşler.
getOutputs()	Katmanın sinirlerin çıkış değerleri dizi olarak döndürür.
setOutputs(outputArray)	Katmanın sinirlerin çıkış değerleri günceller.

3.3 Neuron (Sinir)

Başka sinirlerden dentriteler ile bilgi alıp, toplama ve transfer fonksiyonu tâbi tutup, bu çıktıyı başka sinirlere axonlar ile veren yapının, her hücresine sinir denir.

Değişken	Açıklama
functions.sum ()	Toplama fonksiyonudur. (Varsayılan : sum(+))
functions.transfer[] ()	Transfer fonksiyonlarının dizisidir. (Varsayılan : [sigmoid(f)])
out	Sinirin çıktısıdır.
dentrite []	Sinire gelen bağlantıların dizisidir.
axon []	Sinirden çıkan bağlantıların dizisidir.

Fonksiyon	Amaç
connect(neuron)	Bu sinire başka sinire bağlar. Axon oluşturur.
fire ()	Siniri ateşler.

3.4 Connection(Bağlantı)

Sinirleri veya Katmanları birbirine bağlayan yapıdır.

Değişken	Açıklama
input	Bağlantının başladığı yer.(Sinir veya Katman)
output	Bağlantının bittiği yer.(Sinir veya Katman)
w	Bağlantının ağırlığıdır. (Sadece sinirlerde kullanılır)

4. Nöroevrim

4.1 Tanım

Yapay sinir ağlarının genetik algoritmalar ile evrimleştirme tekniğidir. Bu teknikte, Yapay sinir ağları ve Genetik algoritmalar birlikte kullanılır. Bu tekniğin zorluğu, Yapay sinir ağlarının bağlantılarının, ağırlıklarının, nöron sayılarının ve katman sayılarının evrimsel bir süreçle belirlenmesidir. Yapay sinir ağları öğrenmede sadece ağırlıklarını değiştirirken, bu teknikte sinir ağı tamamen otomatik oluşturulur. Eğer oluşturulan sinir ağı yapısı itibariyle yetersiz ise, sonraki nesillerde sinir ağıının yapısında değişebilir. Yapay sinir ağıının yapısı tamamen kromozom üzerinde saklanacağı için, sinir ağı tamamen mutasyonlara ve çaprazlamalara uğramaya açıktır.

4.2 Yapay Sinir Ağıının Genetik Olarak Modellemesi

Yapay sinir ağları , girişlerden ve çıkışlardan oluşur. Giriş ve çıkış sayısı değisemeyeceği için, bu değerlerinin genetik transformasyonlara tabi tutulmasına gerek yoktur. Bir Perceptron ağı gizli katmanlardan oluşur ve bu gizli katmanların sayısı değişebilir. Gizli katmanların sinir sayılarında değişebilir. O zaman gizli katmanlar diye bir genetik diziye ihtiyacımız vardır. Bu dizisinin içine her katmanın sinir sayısı belirleyen genler koymalıdır. Yapay sinir ağlarının en önemli parçası ağırlıklardır. Her katman arası ağırlıkları tutan ağırlık matrisleri iç içe genetik dizi olarak saklanabilir. Bu ağırlık matrislerinde dizi olacağı için, iç içe 3 dizi katmanına sahip bir yapı oluşturmalıyız. Bu yapıda sürekli, katmanlara ve sinir sayılarına göre güncellenmelidir.

4.2.1 YSA'nın genetik ifadesi

- Giriş katmanı sinir sayısı
- Çıkış katmanı sinir sayısı
- Gizli katman sinir sayısı dizisi
 - Gizli katman 1'in sinir sayısı
 - Gizli katman 2'nin sinir sayısı
 - ...
 - Gizli katman n'in sinir sayısı
- Ağırlık matrisleri dizisi
 - Ağırlık matrisi 1
 - Satır 1 : [...]
 - ...
 - Satır n : [...]
 - ...
 - Ağırlık matrisi n
 - Satır 1 : [...]
 - ...
 - Satır n : [...]

4.2.2 evulotion.js ile ifadesi

```
var chromosomeOfEve = new Chromosome().VAL({  
    /* input sayısı : sabit */  
    inputNN : new Gene().TYPE(GA.TYPE.INT).VAL(10).CHG(false),  
    /* output sayısı : sabit */  
    outputNN : new Gene().TYPE(GA.TYPE.INT).VAL(2).CHG(false),  
    /* gizli katmanları sinir sayılarının dizisi */  
    hiddenNN : new Chromosome().VAL([  
        new Gene().TYPE(GA.TYPE.INT).MIN(5).MAX(15).VAL(11),  
        new Gene().TYPE(GA.TYPE.INT).MIN(5).MAX(15).VAL(10)  
    ]).SIZE(3),  
    /* ağıın ağırlık matrislerinin dizisi */  
    W : new Chromosome().VAL([ // ağırlık matislerinin dizisi  
        new Chromosome().VAL([ // 3x2lik ağırlık matrisi  
            new Chromosome().VAL([  
                GA.copyGene(exampleWeightGene).VAL(),  
                GA.copyGene(exampleWeightGene).VAL()  
            ]),  
            new Chromosome().VAL([  
                GA.copyGene(exampleWeightGene).VAL(),  
                GA.copyGene(exampleWeightGene).VAL()  
            ]),  
            new Chromosome().VAL([  
                GA.copyGene(exampleWeightGene).VAL(),  
                GA.copyGene(exampleWeightGene).VAL()  
            ])  
        ]),  
        new Chromosome().VAL([ // 3x1lik ağırlık matrisi  
            new Chromosome().VAL([  
                GA.copyGene(exampleWeightGene).VAL()  
            ]),  
            new Chromosome().VAL([  
                GA.copyGene(exampleWeightGene).VAL()  
            ]),  
            new Chromosome().VAL([  
                GA.copyGene(exampleWeightGene).VAL()  
            ])  
        ])  
    ])  
})
```

5. XOR

Farklı iki input durumunda, 1 çıktısı döndüren mantıksal kapıdır.

input	output
0 0	0
0 1	1
1 0	1
1 1	0

5.1 XOR probleminin Nöroevrim ile çözümü

- Bu problemin çözümü için YSA kullanacağız.
- Bu YSANın yapısı kromozomda saklanacak.
- Genetik algoritmalar ile yeni kromozomlar üretilecek.
- Bu yeni kromozomlar, YSA'ya çevrilip, ileri beslemeli olarak ateşlenecek.
- Çıktının XOR'a uyumluluğuna göre bir fitness değeri atanacak.

5.1.1 Problem

- YSA'nın XOR çözebilecek şekilde nöroevrim ile eğitilmesi.

5.1.2 Fitness Fonksiyonu

1. [0,0] , [0,1] , [1,0] , [1,1] Giriş setleri için YSA ileriye doğru ateşlenir
2. YSA'dan alınan çıktılar OUTS dizisine eklenir.
3. beklenen = [0,1,1,0] dizisini oluştur.
4. hata -> her bir outs elemanı için hata = hata +| outs[i]-beklenen[i] |
5. fitness = 8 – hata

5.1.3 Başlangıç fonksiyonu oluşturma fonksiyonu

1. EVE kromozomunu kopyalayarak yeni kromozom oluştur.
2. Kromozomu mutasyona uğrat.
3. Populasyon sayısına ulaşmadıysa Adım.1'e dön.
4. DUR

5.1.3.1 EVE kromozomu

Başlangıç için , örnek YSA kromozomudur.

5.1.4 Evrimsel Parametreler

Parametre	Değer
algorithm	GA.ALGORITHMS.STANDARD
crossing_overMethod	GA.CO_TYPES.MULTIPARTIALLY
selectionMethod	GA.SELECTION.ROULETTE
crossing_overPartNum	3
population_size	100
iterations	100
elitism	true
elit_num	10

(varsayılan ; mutation_rate : 0.9 ,crossing_overRate : 0.5,)

5.1.4.1 Algoritma Hakkında

- Standart Genetik Algoritma kullanılacak
- Populasyon büyülüğu 100 olacak
- 100 iterasyon yapılacak.
- Elitizm aktif olacak, ve en iyi ilk 10 elit olacak.

5.2 Sonuçlar ve Analiz

Jenerasyon	Ortalama Fitness(0-8)	Maximum Fitness(0-8)
0	5.9853793614933855	6.937523766717593
1	6.02892588461636	6.937523766717593
2	6.0245286695214215	6.937523766717593
3	6.068766390885982	6.937523766717593
4	6.039143621117259	6.937523766717593
5	6.088661354283733	6.937523766717593
6	6.123594933367906	7.551015688546342
7	6.121492726326373	7.551015688546342
8	6.1348510323782035	7.551015688546342
9	6.142278608978312	7.551015688546342
10	6.122652097146754	7.551015688546342
11	6.104899427317041	7.551015688546342
12	6.0947191208213845	7.551015688546342
13	6.1193007619777635	7.551015688546342
14	6.12889032501849	7.551015688546342
15	6.143243942880327	7.551015688546342
16	6.133391256562615	7.551015688546342
17	6.135757265634031	7.551015688546342
18	6.107409840413275	7.551015688546342
19	6.085533291048266	7.551015688546342
20	6.122126100002131	7.551015688546342
21	6.1497327219166165	7.551015688546342
22	6.115778558311185	7.551015688546342
23	6.124639177996905	7.551015688546342
24	6.128873567446565	7.551015688546342
25	6.0979540910337455	7.551015688546342
26	6.129459550464836	7.551015688546342
27	6.155084122509484	7.551015688546342
28	6.192821588219176	7.551015688546342
29	6.101950393090954	7.551015688546342
30	6.0992035627284125	7.551015688546342
31	6.119977624867375	7.551015688546342
32	6.133814588038197	7.551015688546342
33	6.121812866296519	7.551015688546342

Jenerasyon	Ortalama Fitness(0-8)	Maximum Fitness(0-8)
34	6.144438476903181	7.551015688546342
35	6.161879431592395	7.551015688546342
36	6.147625488034674	7.551015688546342
37	6.137115143079325	7.551015688546342
38	6.124069244898761	7.551015688546342
39	6.108855858198939	7.551015688546342
40	6.136075791636646	7.551015688546342
41	6.126241827595781	7.551015688546342
42	6.186575257303394	7.551015688546342
43	6.16007016727998	7.551015688546342
44	6.145218893422915	7.551015688546342
45	6.150360583825057	7.551015688546342
46	6.139894597358867	7.551015688546342
47	6.159741922155199	7.551015688546342
48	6.173780653749059	7.551015688546342
49	6.139255328901698	7.551015688546342
50	6.142670585094293	7.551015688546342
51	6.168015011171161	7.551015688546342
52	6.142285682222023	7.551015688546342
53	6.170531912703114	7.551015688546342
54	6.212699643149327	7.551015688546342
55	6.1819702879467195	7.551015688546342
56	6.136222556941739	7.551015688546342
57	6.1747047660152745	7.551015688546342
58	6.162556446876005	7.551015688546342
59	6.137216800040178	7.551015688546342
60	6.17712086443637	7.551015688546342
61	6.171824846576377	7.551015688546342
62	6.126765154225122	7.551015688546342
63	6.206620093532654	7.551015688546342
64	6.184783755379397	7.551015688546342
65	6.131011986480643	7.551015688546342
66	6.182083925561704	7.551015688546342
67	6.176537559523223	7.551015688546342
68	6.1358227499780185	7.551015688546342

Jenerasyon	Ortalama Fitness(0-8)	Maximum Fitness(0-8)
69	6.163269250561004	7.551015688546342
70	6.155117048928087	7.551015688546342
71	6.160019864861104	7.551015688546342
72	6.170276822506386	7.551015688546342
73	6.161401449472436	7.551015688546342
74	6.143276732791595	7.551015688546342
75	6.2073081443751725	7.551015688546342
76	6.202690272537954	7.551015688546342
77	6.148574936810714	7.551015688546342
78	6.16523295831971	7.551015688546342
79	6.188989708149038	7.551015688546342
80	6.134316789184347	7.551015688546342
81	6.193977918644855	7.551015688546342
82	6.167275614449226	7.551015688546342
83	6.1551886461452385	7.551015688546342
84	6.15117147242613	7.551015688546342
85	6.150326754826681	7.551015688546342
86	6.156662867956826	7.551015688546342
87	6.192768057108898	7.551015688546342
88	6.151412442743691	7.551015688546342
89	6.117448713854242	7.551015688546342
90	6.155134573722092	7.551015688546342
91	6.189542417311897	7.551015688546342
92	6.1601364271253995	7.551015688546342
93	6.1826662846471825	7.551015688546342
94	6.1977204627233755	7.551015688546342
95	6.204991568666926	7.551015688546342
96	6.199905131626515	7.831901595818998
97	6.152062932184881	7.831901595818998
98	6.128345989980976	7.831901595818998
99	6.20941059806739	7.950397666835871

En iyi sonuç (beklenen : [0,1,1,0])

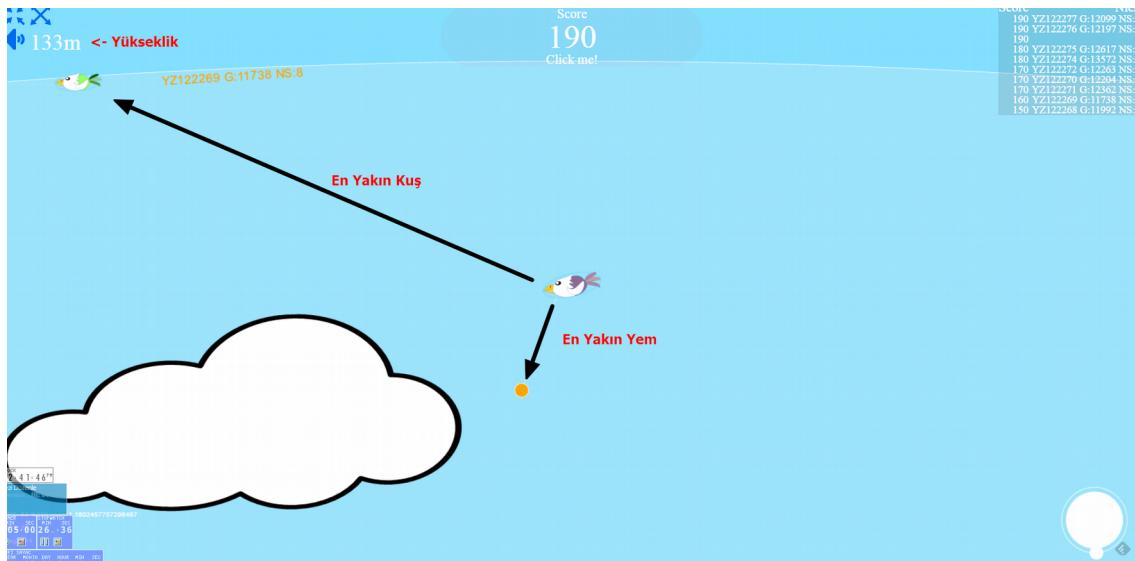
[0.0025113558982609715, 0.9554067390221931, 0.9984582624084958, 0.0009559786965573613]

En iyi sonucun genetik kod

```
{"inputNN":2,"outputNN":1,"hiddenNN":[5,5,4],"W":[[[-  
0.5206460468681131,0.14697962394732,0.7051736150494592,0.14697962394732,0.14697962394732  
],[0.3575030980015881,-0.32597186458807803,-0.5946007278727836,-0.10416497326425356,-  
0.32597186458807803],[-  
0.05599332381021682,0.5805447252935712,0.31917397347558385,0.8434973716303769,0.31917397  
347558385]], [[-0.3783256488956259,0.5807485890039374,-0.36445421782678045,-  
0.3783256488956259,-0.36445421782678045],[0.4523983411685668,0.7807098193949282,-  
0.09950785159119846,-0.3783256488956259,0.4523983411685668],[-  
0.3783256488956259,0.5807485890039374,-0.36445421782678045,-0.3783256488956259,-  
0.3783256488956259],[0.9078283661131228,0.8342563883466951,0.9395598139035855,-  
0.3783256488956259,-0.3783256488956259],  
[0.9078283661131228,0.8342563883466951,0.9395598139035855,-  
0.3783256488956259,0.9395598139035855], [-0.3783256488956259,0.5807485890039374,-  
0.36445421782678045,-0.3783256488956259,-0.3783256488956259]], [[0.3387170965313673,-  
0.7202171327934543,0.3387170965313673,0.3387170965313673],[-  
0.9785173221503713,0.738118881527408,0.738118881527408,-0.9785173221503713],  
[0.10968387424771375,0.10968387424771375,0.10968387424771375,0.10968387424771375],[-  
0.7518037453157289,-0.7518037453157289,-0.7518037453157289,-0.7518037453157289],[-  
0.86301194971493,0.5982339116422084,-0.86301194971493,0.5982339116422084],[-  
0.86301194971493,0.5982339116422084,0.5982339116422084,0.5982339116422084]],  
[[0.3387170965313673],[0.738118881527408],[0.10968387424771375], [-0.7518037453157289],  
[0.5982339116422084]]}
```

6. Kartal.io^[5]

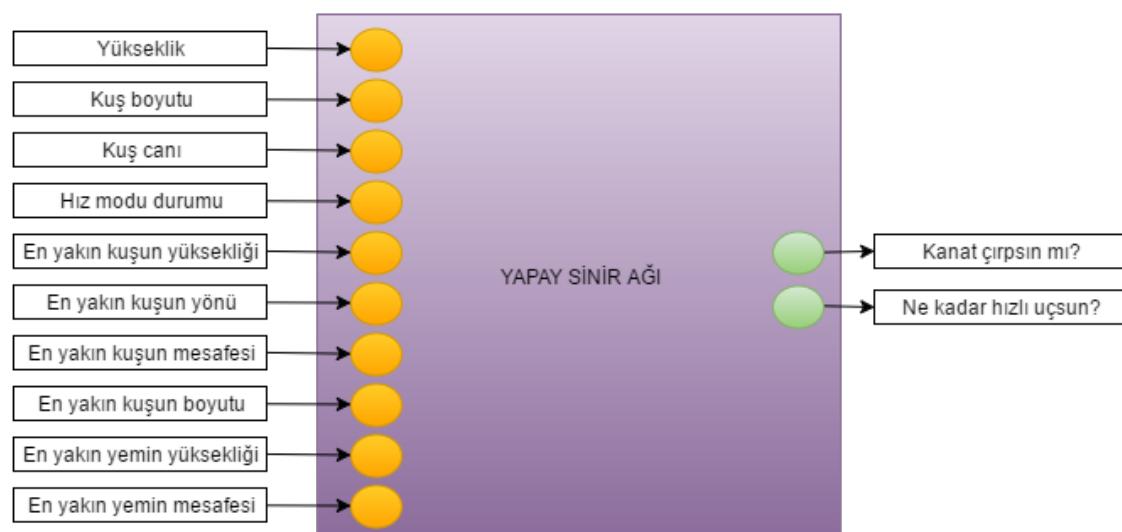
Kanat çırpma kontrolüyle oynanan, yemleri veya diğer kuşları yemeye çalışılan çok kişilik gerçek zamanlı oyun.



Resim 1: kartal.io Ekran Görüntüsü

6.1 Kartalın sinir yapısı

Şekil 3'te kartal.io'da olması gereken yapay sinir ağı gösterilmiştir.



Şekil 3: Kartalın sinir yapısı

6.1.1 Girişler

- Yükseklik
- Kuşun boyutu
- Kuşun canı
- Hız Modu Açık mı?
- En yakın kuşun yüksekliği
- En yakın kuşun yönü
- En yakın kuşun mesafesi
- En yakın kuşun boyutu
- En yakın yemin yüksekliği
- En yakın yemin mesafesi

6.1.2 Çıkışlar

- Kanat Çırp
- Hızlı uç

6.1.3 Yapay sinir ağları

- Giriş katmanı : 10 nöron
- Çıkış katmanı : 2 nöron
- Arakatman ve ağırlıklar : Evrimsel yapı sebebiyle kromozomlar tarafından belirlenir.

6.2 Fitness

- Hayatta kalmak : Yenmemek, yere düşmemek, büyük kuşlara yaklaşmamak, ters yöndeği kuşlardan uzak durmak
- Yemek yiyeilmek : Belli aralıklarda beslenmek, aç kalmamak
- Rekabet : Diğer kartallarla ve insan tarafından yönetilen kartallar ile aralarında rekabet vardır.

NOT : Yapay Zeka olan her kuşa "YZ{no} G{nesil} NS {gizli katman bilgisi}"(ÖRN . YZ58932 G65 NS 5+5) adı verilmiştir.

6.3 Analiz

Kartal populasyonu 17 saat boyunca test edilmiştir.

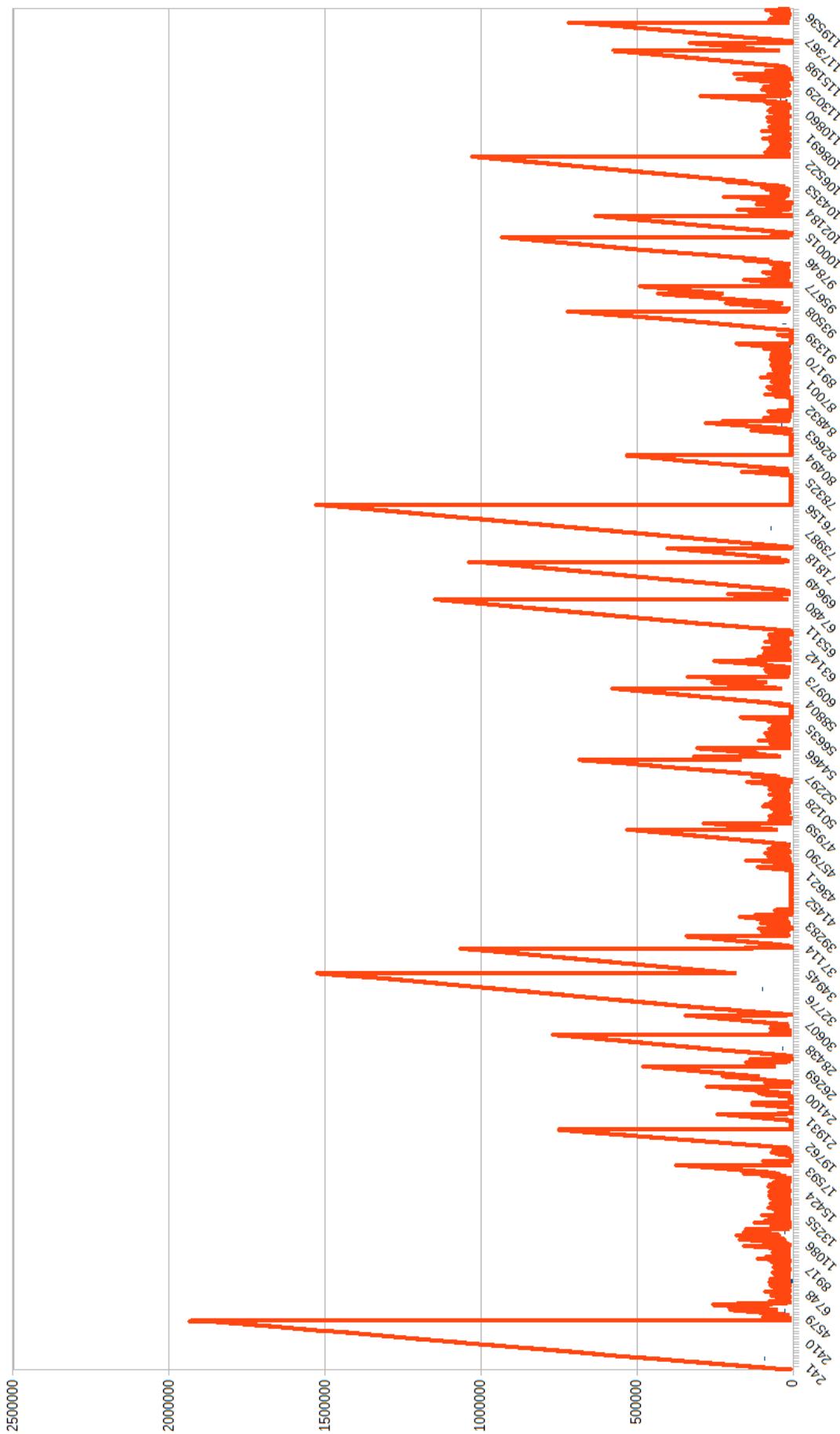
Bu süre boyunca 120000 yeni kartal doğmuştur. Bütün kartalların yapay sinir ağlarına test edilmiştir.

Kartallar zamanla uçabilme yeteneği, diğer kartalları ve yemleri yiyebilme yeteneği kazanmıştır. Hatta bazı kartallar birbirine çarpmadan uçabilmek için yeni yöntemler geliştirmiştir.

Test sonucunda en uzun yaşıyan kartal 0.55 saat (1951000ms) uçmuştur. Aşağıdaki şekilde görüldüğü gibi bireylerin ömrü sürekli değişmiştir.

Populasyon boyutu çok az olduğu için, 10 kişilik bir populasyon ile sürekli sağlanamamıştır. Gerçek zamanlı çalışan bu simülasyon, 10 kişilik populasyonda %90 işlemci kullanıma sahip olduğu için, ideal populasyon boyutu olan 100 bireylilik populasyonu test edemedik. Çünkü o parametreler 1000 kat daha fazla işlemci gücü gerektirmektedir.

Şekil 4te yeni kartallara göre, kartal ömrü grafiğinde, süreksizliği görülmektedir.



Şekil 4: kartal.io Nöroevrimsel Kartal Populasyonu

(x ekseni : kuş sırası y ekseni: kuş ömrü)

7. Sonuç

Bir probleme yönelik yapay sinir ağını, hem standart genetik algoritmalar hem de gerçek zamanlı genetik algoritmalar ile üretebiliyor. Bu nöroevrimsel yöntem, sadece XOR ve kartal.io'ya değil, bir çok şeye de uygulanabilir. Sadece girdi ve çıktıları belirleyerek, istediğimiz yapay sinir ağını bu yöntem ile üretebiliriz. Gerçek zamanlı similasyon olan kartal.io'da, kartalların hiç kendine öğretilmeyen yetenekler kazanması , bu yöntemin ne kadar özgün çözümler sunabildiğini göstermektedir. Hatta canlıların evrimsel süreçlerini bu yöntem ile simüle edebiliriz. Kartalların yemlerin azaldığı dönemde birbirini yemeye başlaması, daha sonra daha iyi bir kartalın bundan vazgeçip, yere yakın yemleri yiyebilme yeteneği kazanıp, diğerlerine üstün gelmesi, bu yöntemin ne kadar doğaya benzer çözümler ürettiğinin bir işaretidir.

Kaynakça

- [1] HOLLAND, JOHN - Adaptation in Natural and Artificial Systems
- [2] WIKIPEDIA - https://tr.wikipedia.org/wiki/Genetik_algoritma (Erişim Tarihi: 1.5.2016)
- [3] KARABOĞA, DERVİŞ - Yapay Zeka Optimizasyon Algoritmaları
- [4] W3SCHOOLS – Javascript dili için dökümantasyon <http://www.w3schools.com/js/>
- [5] KARTAL.IO – Javascript ile yazılmış Nöroevrimsel Yapay Zeka içeren gerçek zamanlı oyun <http://kartal.io/>

8. Ekler

8.1 Evolution.js

```
// #
// # *** evolution.js ***
// #
// # by Faruk CAN (farukcan.net) @ 2016
// # license : GNU GPL
/*
GA : object
Evolution : class
Population : class
Member : class
Chromosome : function returns new Gene().TYPE(CHROMOSOME)
Gene : class
GenePart : class
*/
var GA = {
    defaultParameters : {
        mutation_rate : 0.2,
        population_size : 20,
        iterations : 20,
        elitism : true,
        selectionMethod : 0, // GA.SELECTION.ROULETTE
        crossing_over : true,
        crossing_overRate : 0.5,
        crossing_overMethod : 1, // GA.CO_TYPES MULTIPARTIALLY
        crossing_overUniformRate : 0.5,
        crossing_overPartNum : 3,
        elit_num : 1,
        algorithm : 0 // GA.ALGORITHMS.STANDART
    },
    charSet :
    "ABCÇDEFGĞHIJKLMNOPÖOPQRSŞTUÜVWXYZabçeđfgħiñjklmnoöpqrsştüuvwxyz0123456789 ,!'^+"
    "%&/()-@æß",
    TYPE : {
        BIT : 0,
        INT : 1,
        UNIPOLAR : 2,
        BIPOLAR : 3,
        STRING : 4,
        CHROMOSOME : 5
    },
    CO_TYPES : {
        PARTIALLY : 0,
        MULTIPARTIALLY : 1,
        UNIFORM : 2
    },
    SELECTION : {
        ROULETTE : 0,
        SORT : 1
    },
    ALGORITHMS : {
        STANDART : 0,
        DIEANDBORN : 1
    }
},
```

```

random : function() { return Math.random(); },
randomINT : function(min,max) { return Math.round(Math.random()*(max-min) + min); },
chance : function(rate) { return rate > GA.random() },
copyGene : function(oldGene) {
    var gene = new Gene().TYPE(oldGene.type);
    for (key in oldGene)
        if (typeof oldGene[key] != 'function' && key != 'val')
            gene[key] = oldGene[key];
    if(oldGene.type==GA.TYPE.CHROMOSOME){
        if(oldGene.val instanceof Array)
            gene.val = [];
        else
            gene.val = {};
        for(key in oldGene.val){
            gene.val[key] = GA.copyGene(oldGene.val[key]);
        }
    } else{
        gene.val = oldGene.val;
    }
    if(oldGene.RULEfunc) gene.RULE(oldGene.RULEfunc);
    return gene;
},
INS : function(geneArray){
    var i = GA.randomINT(0,geneArray.length-1);
    var copy = GA.copyGene(geneArray[i]);
    geneArray[geneArray.length] = copy;
},
RMV : function(geneArray){
    if(geneArray<=0) return;
    geneArray.splice(GA.randomINT(0,geneArray.length-1),1);
},
SWP : function(geneArray){
    var i,j;
    do{
        i = GA.randomINT(0,geneArray.length-1);
        j = GA.randomINT(0,geneArray.length-1);
    } while(i==j && geneArray.length>1)
    var swap = geneArray[i];
    geneArray[i] = geneArray[j];
    geneArray[j] = swap;
},
splitToParts : function(chromosome){
    var parts = [];
    var part;
    var split = function(chromosome,way,root){
        var ROOT = typeof way == 'undefined' ? chromosome : root;
        var WAY = typeof way == 'undefined' ? [] : way;
        var parts = [];
        for(key in chromosome.val){
            if(chromosome.val[key].chg){
                if(chromosome.val[key].type==GA.TYPE.CHROMOSOME)
                {
                    parts = parts.concat(split(chromosome.val[key],WAY.concat([key]),ROOT));
                }
                else
                {
                    parts.push(new GenePart(WAY.concat([key]),ROOT))
                }
            }
        }
        return parts;
    };
    return split(chromosome);
},
crossingOverable : function(partsA,partsB){
    var r = [[],[]];

```

```

    for(var i=0; i <partsA.length; i++){
        for(var j=0; j <partsB.length; j++){
            if(partsA[i].isSameWith(partsB[j])){
                if(partsA[i].getGene().type==partsB[j].getGene().type){
                    r[0].push(partsA[i]);
                    r[1].push(partsB[j]);
                }
                break;
            }
        }
    }
    return r;
},
crossingOver : function(chromosomeA,chromosomeB,CO_TYPE,PARTNUM_OR_UNIFORMRATE){
    var partsA = GA.splitToParts(chromosomeA);
    var partsB = GA.splitToParts(chromosomeB);
    var crossingOverable = GA.crossingOverable(partsA,partsB);
    partsA = crossingOverable[0];
    partsB = crossingOverable[1];
    switch(CO_TYPE){
        case GA.CO_TYPES.PARTIALLY:
            // 1 nokta seç , o noktadan yer değiştir
            if(partsA.length>1){
                var i = GA.randomINT(1,partsA.length-1);
                var A1 = partsA.slice(0,i);
                var A2 = partsA.slice(i, partsA.length);
                var B1 = partsB.slice(0,i);
                var B2 = partsB.slice(i, partsB.length);
                partsA = B1.concat(A2);
                partsB = A1.concat(B2);
            }
            break;
        case GA.CO_TYPES.MULTIPARTIALLY:
            var PARTNUM = PARTNUM_OR_UNIFORMRATE;
            if(partsA.length>PARTNUM){
                var partition = [];
                var sum = PARTNUM;
                var num=sum;
                while(num--) partition.push(1);
                while((sum<partsA.length-1) && GA.chance(0.5)){
                    partition[GA.randomINT(0,partition.length-1)]++;
                    sum++;
                }
                var partitionIndex= 0;
                var mode = true;
                for(var i=0; i < partsA.length;i++){
                    if(mode){
                        var swap = partsA[i];
                        partsA[i] = partsB[i];
                        partsB[i] = swap;
                    }
                    if(typeof partition[partitionIndex] != 'undefined'){
                        partition[partitionIndex]--;
                        if(partition[partitionIndex]==0) {
                            mode = !mode;
                            partitionIndex++;
                        }
                    }
                }
                // yukardakini birkaç noktadan yap
            }
            break;
        case GA.CO_TYPES.UNIFORM:
            // sanlsa göre parçalari değiştir.
            for(var i=0; i < partsA.length;i++){
                if(GA.chance(PARTNUM_OR_UNIFORMRATE)){

```

```

        var swap = partsA[i];
        partsA[i] = partsB[i];
        partsB[i] = swap;
    }
}
break;
}
if(GA.chance(0.5)){
    // swap parts
    var swap = partsA;
    partsA = partsB;
    partsB = swap;
}
var childA = GA.copyGene(chromosomeA);
var childB = GA.copyGene(chromosomeB);
partsA.forEach(function(part){
    var gene = part.getGene(childA);
    gene.val = part.getGene().val;
});
partsB.forEach(function(part){
    var gene = part.getGene(childB);
    gene.val = part.getGene().val;
});
return [childA,childB]; // iki kromozom döndür.
},
crossingOverRULED : function(chromosomeA,chromosomeB,CO_TYPE,PARTNUM_OR_UNIFORMRATE){
    var r = GA.crossingOver(chromosomeA,chromosomeB,CO_TYPE,PARTNUM_OR_UNIFORMRATE);
    if(r[0].RULEfunc) r[0].RULEfunc(r[0].val);
    if(r[1].RULEfunc) r[1].RULEfunc(r[1].val);
    return r;
};
function Evolution(fitnessFunc,createFunc,bornFunc){
    this.population;
    this.parameters = GA.defaultParameters;
    this.fitnessFunction = fitnessFunc; // f(member)
    this.createPopulation = createFunc;
    this.bornFunction = typeof bornFunc === 'function' ? bornFunc : function(member){};
    this.onNewMember = function(member){}; // algoritma tarafından setlenir
}
Evolution.prototype = {
    setParameters : function(param){
        for(key in param){
            this.parameters[key] = param[key];
        }
        return this;
    },
    start : function(){
        switch (this.parameters.algorithm){
            case GA.ALGORITHMS.STANDART:
                this.createPopulation(this.parameters.population_size);
                this.population.calcFitness();
                var iteration=0;
                while(iteration<this.parameters.iterations){
                    console.log('iteration: '+iteration)
                    console.log("avg fitness:" +this.population.avgFitness)
                    console.log("max fitness:" +this.population.maxFitness)
                    console.log("mem"+this.population.members.length)
                    console.log('select')
                    this.population.selection();
                    console.log('cross')
                    this.population.crossing_over();
                    console.log('mutate')
                    this.population.mutation();
                    this.population.calcFitness();
                    iteration++;
                }
        }
    }
}

```

```

        }
        break;
    case GA.ALGORITHMS.DIEANDBORN:
        this.onNewMember = function(member){
            this.bornFunction(member);
            member.kill = function(){
                //fitness
                this.population.calcFitness();
                //selection
                var chromosomes = this.population.select2chromosome();
                //crossing over
                if(this.population.evolution.parameters.crossing_over &&
GA.chance(this.population.evolution.parameters.crossing_overRate)){
                    if(this.population.evolution.parameters.crossing_overMethod==GA.CO_TYPES.MULTIPARTIALLY)
                        chromosomes=
GA.crossingOver(chromosomes[0],chromosomes[1],this.population.evolution.parameters.crossing_overMethod,this
s.population.evolution.parameters.crossing_overPartNum);
                    else
                        chromosomes=
GA.crossingOver(chromosomes[0],chromosomes[1],this.population.evolution.parameters.crossing_overMethod,this
s.population.evolution.parameters.crossing_overUniformRate);
                }
                //tekini seç
                this.chromosome = chromosomes[0];
                //mutasyona uğrat
                this.chromosome.mutate();
                this.generation++;
                this.id = this.population.lastMemberId++;
                //yeniden doğ
                this.population.evolution.bornFunction(this);
                return true;
            }
        };
        this.createPopulation(this.parameters.population_size);
        break;
    }
};

function Population(evo){
    this.members = [];
    this.bestMember;
    this.evolution=evo;
    this.avgFitness;
    this.maxFitness;
    this.minFitness;
    this.totalFitness;
    this.lastMemberId = 0;
    evo.population = this;
}
Population.prototype = {
    calcFitness : function(){
        this.totalFitness=0;
        this.maxFitness=-1e+99;
        this.minFitness=1e+99;
        var _this=this;
        this.members.forEach(function(member){
            member.fitness = _this.evolution.fitnessFunction(member);
            _this.totalFitness+=member.fitness;
            if(_this.maxFitness<member.fitness){
                _this.maxFitness=member.fitness;
                _this.bestMember = member;
            }
            _this.minFitness = Math.min(_this.minFitness,member.fitness);
        });
        this.avgFitness=_this.totalFitness/this.members.length;
    }
};

```

```

        console.log("AVG: ",_this.avgFitness,"MAX: ",_this.maxFitness);
    },
    select : function(oldPop){
        var r = GA.random();
        for(var i=0;i<oldPop.length;i++){
            if(oldPop[i]._rate>r){
                var newMem = new Member(this);
                newMem.chromosome = GA.copyGene(oldPop[i].chromosome);
                newMem.generation = oldPop[i].generation;
                return newMem;
            }
        }
    },
    rate : function(oldPop){
        var _this=this;
        oldPop.sort(function(a,b){
            return a.fitness - b.fitness;
        });
        switch (this.evolution.parameters.selectionMethod) {
            case GA.SELECTION.ROULETTE:
                var _rate = 0;
                oldPop.forEach(function(member){
                    member._rate = _rate+member.fitness/_this.totalFitness;
                    _rate = member._rate;
                });
                return oldPop;
                break;
            case GA.SELECTION.SORT:
                var _rate = 0;
                oldPop.forEach(function(member,i){
                    member._rate = _rate+(i+1)/(_this.members.length*(_this.members.length+1)/2);
                    _rate = member._rate;
                });
                return oldPop;
                break;
        }
    },
    select2chromosome : function () {
        var _this=this;
        switch (this.evolution.parameters.selectionMethod) {
            case GA.SELECTION.ROULETTE:
                var _rate = 0;
                this.members.forEach(function(member){
                    member._rate = _rate+member.fitness/_this.totalFitness;
                    _rate = member._rate;
                });
                break;
            case GA.SELECTION.SORT:
                var _rate = 0;
                var sorted = this.members.slice(0).sort(function(a,b){
                    return a.fitness - b.fitness;
                });
                sorted.forEach(function(member,i){
                    member._rate = _rate+(i+1)/(_this.members.length*(_this.members.length+1)/2);
                    _rate = member._rate;
                });
                break;
        }
        // rateler ayarlandi.
        // iki tane seq
        var chromo1 ;
        var chromo2 ;
        var r = GA.random();
        for(var i=0;i<this.members.length;i++){
            if(this.members[i]._rate>r){
                chromo1 = GA.copyGene(this.members[i].chromosome);

```

```

        }
    }
    r = GA.random();
    for(var i=0;i<this.members.length;i++){
        if(this.members[i]._rate>r){
            chromo2 = GA.copyGene(this.members[i].chromosome);
        }
    }
    return [chromo1,chromo2];
},
selection : function(){
    switch (this.evolution.parameters.selectionMethod){
        case GA.SELECTION.ROULETTE:
            var oldPop = this.members;
            this.members = [];
            oldPop = this.rate(oldPop);
            if(this.evolution.parameters.elitism){
                for(var i=0;i<this.evolution.parameters.elit_num;i++){
                    this.members.push(oldPop[oldPop.length-(i+1)]);
                }
                while(this.members.length<oldPop.length){
                    this.select(oldPop);
                }
                break;
            }
        case GA.SELECTION.SORT:
            var oldPop = this.members;
            this.members = [];
            oldPop = this.rate(oldPop);
            if(this.evolution.parameters.elitism){
                for(var i=0;i<this.evolution.parameters.elit_num;i++){
                    this.members.push(oldPop[oldPop.length-(i+1)]);
                }
                while(this.members.length<oldPop.length){
                    this.select(oldPop);
                }
                break;
            }
    }
},
crossing_over : function(){
    if(this.evolution.parameters.crossing_over==false) return;
    if(this.evolution.parameters.elitism){
        start = this.evolution.parameters.elit_num;
    }else{
        start = 0;
    }
    for(var i=start;i<(this.members.length-1);i+=2){
        if( !GA.chance(this.evolution.parameters.crossing_overRate)) continue;
        if(this.evolution.parameters.crossing_overMethod==GA.CO_TYPES.MULTIPARTIALLY)
            var childs=
GA.crossingOver(this.members[i].chromosome,this.members[i+1].chromosome,this.evolution.parameters.crossing_overMethod,this.evolution.parameters.crossing_overPartNum);
        else
            var childs=
GA.crossingOver(this.members[i].chromosome,this.members[i+1].chromosome,this.evolution.parameters.crossing_overMethod,this.evolution.parameters.crossing_overUniformRate);
        var generation = Math.max(this.members[i].generation,this.members[i+1].generation)+1;
        this.members[i].generation = generation;
        this.members[i+1].generation = generation;
        this.members[i].chromosome = childs[0];
        this.members[i+1].chromosome = childs[1];
    }
},
mutation : function(){
    var elitism = this.evolution.parameters.elitism;
    if(elitism) {
        for (var i = this.evolution.parameters.elit_num; i < this.members.length; i++)

```

```

        this.members[i].chromosome.mutate();
    }
    else{
        this.members.forEach(function(member){
            member.chromosome.mutate();
        });
    }
};

function Member(pop,chromosome){
    this.generation=0;
    this.chromosome=chromosome;
    this.fitness;
    this.population=pop;
    this.id = pop.lastMemberId++;
    this.index = pop.members.length;
    pop.members.push(this);
    pop.evolution.onNewMember(this);
}
function Chromosome(){
    return new Gene().TYPE(GA.TYPE.CHROMOSOME);
}
function Gene(){
    this.type = GA.TYPE.BIT;
    this.val = 0;
    this.mutation_rate = GA.defaultParameters.mutation_rate;
}
Gene.prototype = {
    mutate : function(){
        switch(this.type){
            case GA.TYPE.BIT:
                if( this.chg && GA.chance(this.mutation_rate) ){
                    this.val = this.val==0 ? 1 : 0; // 0 ve 1 yer değiştirir
                }
                break;
            case GA.TYPE.INT:
                if( this.chg && GA.chance(this.mutation_rate) ){
                    this.setRandom();
                }
                break;
            case GA.TYPE.UNIPOLAR:
                if( this.chg && GA.chance(this.mutation_rate) ){
                    this.setRandom();
                }
                break;
            case GA.TYPE.BIPOLAR:
                if( this.chg && GA.chance(this.mutation_rate) ){
                    this.setRandom();
                }
                break;
            case GA.TYPE.STRING:
                // CHG
                if(this.chg && GA.chance(this.mutation_rate)){
                    for(var i=0;i<this.val.length;i++){
                        if(GA.chance(this.chg_rate)){
                            this.val = this.val.substr(0, i) + GA.charSet.substr(GA.randomINT(0,GA.charSet.length-1), 1)+this.val.substr(i+1);
                        }
                    }
                }
                // INS
                if(this.ins && GA.chance(this.ins_rate) && this.val.length>0){
                    var i = GA.randomINT(0,this.val.length-1);
                    this.val = this.val.substr(0, i)+ GA.charSet.substr(GA.randomINT(0,GA.charSet.length-1), 1)+this.val.substr(i);
                }
        }
    }
}

```

```

//RMV
if(this.rmv && GA.chance(this.rmv_rate) && this.val.length>0 ){
    var i = GA.randomINT(0,this.val.length-1);
    this.val = this.val.substr(0, i)+ this.val.substr(i+1);
}
//SWP
if(this.swp && GA.chance(this.swp_rate) && this.val.length>0 ){
    var i = GA.randomINT(0,this.val.length-1);
    var m = this.val.substr(i,1);
    var j = GA.randomINT(0,this.val.length-1);
    var n = this.val.substr(j,1);
    this.val = this.val.substr(0, i) + n + this.val.substr(i+1);
    this.val = this.val.substr(0, j) + m + this.val.substr(j+1);
}
break;
case GA.TYPE.CHROMOSOME:
    if(this.val instanceof Array){
        var length = this.val.length;
    }else{
        var length = 0;
        for(key in this.val)
            length++;
    }
    if(length>0){
        if(this.ins && GA.chance(this.ins_rate) && length<this.size){
            GA.INS(this.val)
        }
        if(this.rmv && GA.chance(this.rmv_rate) && length>1){
            GA.RMV(this.val);
        }
        if(this.swp && GA.chance(this.swp_rate)){
            GA.SWP(this.val);
        }
    }
    if(this.chg && GA.chance(this.mutation_rate) ){
        // kendine bağlı bütün genleri mutasyona uğrat
        for(key in this.val){
            this.val[key].mutate();
        }
    }
    break;
}
if(this.RULEfunc) this.RULEfunc(this.val);
},
setRandom : function(){
    switch(this.type){
        case GA.TYPE.BIT:
            this.val = GA.random() > 0.5 ? 1 : 0;
            break;
        case GA.TYPE.INT:
            this.val = GA.randomINT(this.min,this.max);
            break;
        case GA.TYPE.UNIPOLAR:
            this.val = GA.random();
            break;
        case GA.TYPE.BIPOLAR:
            this.val = GA.random()*2-1;
            break;
        case GA.TYPE.STRING:
            this.val = "YOU-CANNOT-SET-RANDOM-STRING";
            break;
        // GA.TYPE.CHROMOSOME cannot set random
    }
},
VAL : function(value){
    if(typeof value == 'undefined'){

```

```

        this.setRandom();
    }
    else{
        this.val = value;
        if(this.type==GA.TYPE.CHROMOSOME){
            if(this.val instanceof Array){ // val is Array[]
                this.RMV(true);
                this.INS(true);
                this.SWP(true);
            }
            else{ // val is object{
                this.RMV(false);
                this.INS(false);
                this.SWP(false);
            }
        }
    }
    return this;
},
TYPE : function(TYPE){
    this.type = TYPE;
    switch(TYPE){
        case GA.TYPE.BIT:
            this.VAL();
            this.CHG(true);
            break;
        case GA.TYPE.INT:
            this.min=-128;
            this.max=128;
            this.VAL();
            this.CHG(true);
            break;
        case GA.TYPE.UNIPOLAR:
            this.VAL();
            this.CHG(true);
            break;
        case GA.TYPE.BIPOLAR:
            this.VAL();
            this.CHG(true);
            break;
        case GA.TYPE.STRING:
            this.CHG(true);
            this.RMV(true);
            this.INS(true);
            this.SWP(true);
            this.chg_rate = GA.defaultParameters.mutation_rate;
            this.ins_rate = GA.defaultParameters.mutation_rate;
            this.rmv_rate = GA.defaultParameters.mutation_rate;
            this.swp_rate = GA.defaultParameters.mutation_rate;
            this.VAL("evolution.js");
            break;
        case GA.TYPE.CHROMOSOME:
            this.VAL({});
            this.CHG(true);
            this.ins_rate = GA.defaultParameters.mutation_rate;
            this.rmv_rate = GA.defaultParameters.mutation_rate;
            this.swp_rate = GA.defaultParameters.mutation_rate;
            this.size = 65555;
            break;
    }
    return this;
},
MIN : function(min){ this.min=min;this.VAL();return this;},
MAX : function(max){ this.max=max;this.VAL();return this;},
CHG : function(bool){ this.chg=bool;return this;},
INS : function(bool){ this.ins=bool;return this;},

```

```

RMV : function(bool){ this.rmv=bool;return this;},
SWP : function(bool){ this.swp=bool;return this;},
MUTATION_RATE : function(r){ this.mutation_rate=r;return this;},
CHG_RATE : function(r){ this.chg_rate=r;return this;},
INS_RATE : function(r){ this.ins_rate=r;return this;},
RMV_RATE : function(r){ this.rmv_rate=r;return this;},
SWP_RATE : function(r){ this.swp_rate=r;return this;},
SIZE : function(r){ this.size=r;return this;},
RULE : function(func){
    this.RULEfunc = func;
    return this;
},
toJSON : function(){
    if(this.type==GA.TYPE.CHROMOSOME){
        if(this.val instanceof Array){ // val is Array[]
            var r = [];
            for(key in this.val){
                r[key] = this.val[key].toJSON();
            }
            return r;
        }
        return this.val;
    }
};
function GenePart(way,root){
    this.parrents = way;
    this.root = root;
}
GenePart.prototype = {
    getGene : function(from){
        var current = typeof from != 'undefined' ? from : this.root;
        this.parrents.forEach(function(child){
            current = current.val[child];
        });
        return current;
    },
    isSameWith : function(genepart){
        if(this.parrents.length!=genepart.parrents.length) return false;
        for(var i = 0 ; i < this.parrents.length ; i++)
            if(this.parrents[i]!=genepart.parrents[i])
                return false;
        return true;
    }
}

```

8.2 ANN.js

```

// #
// # *** ANN.js ***
// #
// # by Faruk CAN (farukcan.net) @ 2016
// # license : GNU GPL
// @class ANN
function ANN(){
    this.layers = [];// YSA nin katmalari bu dizide tutulur
    this.TYPE="CUSTOM";// YSA'nin tipi
}
// @prototype ANN : fonksiyonlar...
ANN.prototype = {

```

```

inputLayer : function() { // bu fonk. ilk katmanı döndürür.
    return this.layers[0];
},
outputLayer : function() { // buda son " "
    return this.layers[this.layers.length-1];
},
fire : function(inputArray){ // tüm YSAyi atesler
    if(inputArray)
        this.setOutputs(inputArray);
    this.layers[1].fireConsecutive();
},
getOutputs : function() { //son katmanın çıkış değerlerini döndürür.
    return this.outputLayer().getOutputs();
},
setOutputs : function(outputs){ // ilk katmanın outputlarını günceller.
    this.inputLayer().setOutputs(outputs);
}
};

// @arcitures ANN Burada belirli YSA modelleri oluşturma fonksiyonları tutulur.
// @arc PERCEPTRON #### kullanım ysa = new ANN().PERCEPTRON(5,[4,3,3],2);
ANN.prototype.PERCEPTRON =
function(inputNeuronNum,arrayof_hiddenLayersNeuronNum,outputNeuronNum){
    this.TYPE = "PERCEPTRON";
// |# OLUŞTURUCU
// ilk katman : inputLayer
    var addLayer = function(L,n){
        L.push(new ANN_Layer(n););
        return L[L.length-1];
    };
    var layerBefore = addLayer(this.layers,inputNeuronNum);
// gizli katmanlar
    var _this = this;
arrayof_hiddenLayersNeuronNum.forEach(function(hiddenNum){
    var hidden = addLayer(_this.layers,hiddenNum);
    layerBefore.connectWithNeurons(hidden);
    layerBefore = hidden;
});
// outputLayeri yarat ve bağla
    var outputLayer = addLayer(this.layers,outputNeuronNum);
    layerBefore.connectWithNeurons(outputLayer);
// |# OLUŞTURUCU SONU
// |# FONKSİYONLAR
    this.setWeights = function(wArray){ // Bu fonksiyon Perceptron ağıının ağırlıklarını günceller.
        var _this = this;
        wArray.forEach(function(w,i){
            _this.layers[i].neurons.forEach(function(neuron,m){
                neuron.axon.forEach(function(axon,n){
                    axon.w = w[m][n];
                });
            });
        });
        return this;
    };
    this.biases = [];
    this.setBias = function(biasArray){
        if(this.biases.length==0){
            for(var i=0;i<biasArray.length;i++){
                var bias = new ANN_Bias(biasArray[i]);
                this.layers[i+1].neurons.forEach(function(neuron){
                    bias.connect(neuron);
                });
                this.biases[i] = bias;
                this.layers[i].neurons.push(bias);
            }
        }else{
            for(var i=0;i<biasArray.length;i++){

```

```

        this.biases[i].out = biasArray[i];
    }
}
return this;
};

// |# FONKSIYONLAR SONU
return this;
};

// @class ANN_Layer
function ANN_Layer(neuron_num){
    this.neurons = []; // katmanın nöronları
    this.dendriteLayers=[]; // solundaki katmanlar
    this.axonLayers=[]; // sağindaki katmanlar
    // nöron sayısı belirtildiyse nöronları oluştur.
    if(typeof neuron_num=='number'){
        while(neuron_num>0){
            this.neurons.push(new ANN_Neuron());
            neuron_num--;
        }
    }
}

// @prototype ANN_Layer
ANN_Layer.prototype = {
    connect : function(layer){ // iki katmani birbirine bağlar
        var connection = new ANN_Connection();
        connection.input = this;
        connection.output = layer;
        // layerları bağlama
        layer.dendriteLayers.push(connection);
        this.axonLayers.push(connection);
    },
    connectWithNeurons : function(layer){ // iki katmani birbirine nöronlarıyla birlikte bağlar.
        this.connect(layer);
        // her nöronu teker teker bağla
        this.neurons.forEach(function(leftNeuron) {
            layer.neurons.forEach(function (rightNeuron) {
                leftNeuron.connect(rightNeuron);
            });
        });
    },
    fire : function(){ // katmandaki bütün nöronları ateşler
        this.neurons.forEach(function(neuron){
            neuron.fire();
        });
    },
    fireConsecutive : function(){ // bu katmani ve bu katmanın sağindaki bütün katmanları ateşler
        this.fire();
        if(this.axonLayers.length!=0{
            this.axonLayers.forEach(function(connection){
                connection.output.fireConsecutive();
            });
        }
    },
    setOutputs : function (outputArray) { // katmandaki nöronları çıktılarını günceller
        var neurons = this.neurons;
        outputArray.forEach(function(output,i){
            neurons[i].out = output;
        });
    },
    getOutputs : function () { // katmandaki nöronların çıktılarını alır
        var outputs = [];
        this.neurons.forEach(function(neuron){
            outputs.push(neuron.out);
        });
        return outputs;
    }
}

```

```

};

// @class ANN_Neuron
function ANN_Neuron(){
    this.functions = {
        sum : ANN_f_sum,
        transfer : [ANN_f_sigmoid]
    };
    this.out=0; // Sinir çıktısı saklanır
    // Sinirin bağlı olduğu şeyler
    this.dentrite = []; // nörona gelen bağlantılar
    this.axon = []; // nörondan giden bağlantılar
}

// @prototype ANN_Neuron
ANN_Neuron.prototype = {
    connect : function (neuron){
        var connection = new ANN_Connection;
        connection.input = this;
        connection.output = neuron;
        neuron.dentrite.push(connection);
        this.axon.push(connection);
        return connection;
    },
    fire : function (){
        var x = [],w = [];
        this.dentrite.forEach(function(c){
            x.push(c.input.out);
            w.push(c.w);
        });
        var o = this.functions.sum(w,x);
        this.functions.transfer.forEach(function(f){
            o = f(o);
        });
        return this.out = o;
    }
};

// @class ANN_Bias
function ANN_Bias(out) {
    this.out = out;
    this.axon = [];
}

// @prototype ANN_Bias
ANN_Bias.prototype.connect = ANN_Neuron.prototype.connect;
ANN_Bias.prototype.fire = function(){};

// @class ANN_Connection
function ANN_Connection (w){ // KATMANLARI veya NÖRONLARI bağlar.
    this.input;
    this.output;
    this.w=ANN_default_w; // sadece nöronlarda gerekli
    if(typeof w == 'number') this.w = w;
}

// #
// # (+) Toplama Fonksiyonları
// #
var ANN_f_sum = function (w,x){ // ağırlığa göre topla
    var r = 0;
    for(var i=0; i<w.length;i++)
        r += w[i]*x[i];
    return r;
};
// #
// # (f) Transfer Fonksiyonları
// #
var ANN_f_sign = function (o) { return Math.sign(o); }; // 0'dan konuma göre -1 veya 1
var ANN_f_sigmoid = function (t) { return 1/(1+Math.pow(Math.E, -t)); }; // sigmoid fonksiyonu
var ANN_default_w = 0;

```

9. Özgeçmiş

Ömer Faruk Can, 27 Ağustos 1994'de Antalyada doğdu. Mustafa Barut Lisesinden mezun olup, İstanbul Ticaret Üniversitesi'nde , Bilgisayar mühendisliği okudu. Şuandı İstanbulda yaşamaktadır.



Resim 2: Faruk CAN

E-mail : omer@farukcan.net

9.1 Eğitim

2012+

İstanbul Ticaret Üniversitesi

Computer Engineering

-

2012+

Anadolu Üniversitesi

Business Management Faculty

9.2 Tecrübeler

2014

Garanti Server (AKA Bilişim)
Intern, System Engineering, Network

-

2015+

TÜBİTAK 114E427 Project
Software Engineer, Database Management & Web Develop

-

2012+

Freelancer Web Developer/Master

-

2013+

Indie Game Developer

9.3 Bildiği Teknolojiler

- PHP : Phalcon,Laravel
- JavaScript : NodeJS, Jquery, Phaser, Angular.js, socket.io
- SQL / MySQL
- HTML / 5
- CSS : Bootstap, JqueryUI
- C/C++
- C#
- Java
- V. Basic 6
- OpenGL
- MongoDB
- Embedded Systems: Arduinio , Raspberry