YAPAY SİNİR AĞLARI

PROJE FİNAL RAPORU
REGRESYON İLE EV FİYAT TAHMİNİ

Prof. Dr.Neslihan Serap Şengör

Osman Kaan Kurtça

040160090

Hasan Göktuğ Kayan

040160072

İÇİNDEKİLER

Sayfa

1. Veri Setinin İncelenmesi ve Ön İşleme

Öncelikle problemin çözümü için kullanılacak olan veri seti içeri aktarıldı ve incelendi.

```
In [2]: df = pd.read_csv("housing.csv")
```

```
In [3]: df.head()
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Şekil1.1 Veri setinin ilk 5 satırı

Veri setimiz 10 sütun ve 20640 satırdan oluşuyor. Her bir satır farklı bir bölgeyi(district) temsil ediyor. Sütunlardan 9'u ağımıza sunacağımız giriş verilerini temsil ederken, 'median_house_value' sütunu tahmin edilmesi gereken ev fiyatlarını temsil ediyor. Evlerin ortalama fiyat değerlerine sahip olduğumuz için problemimiz eğiticili öğrenme problemidir. Öncelikle, veri setinin boyutu, sütunlardaki boş değer sayıları, ortalamaları ve standart sapmaları gibi temel özellikleri incelendi.

```
df.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

```
df.shape
```

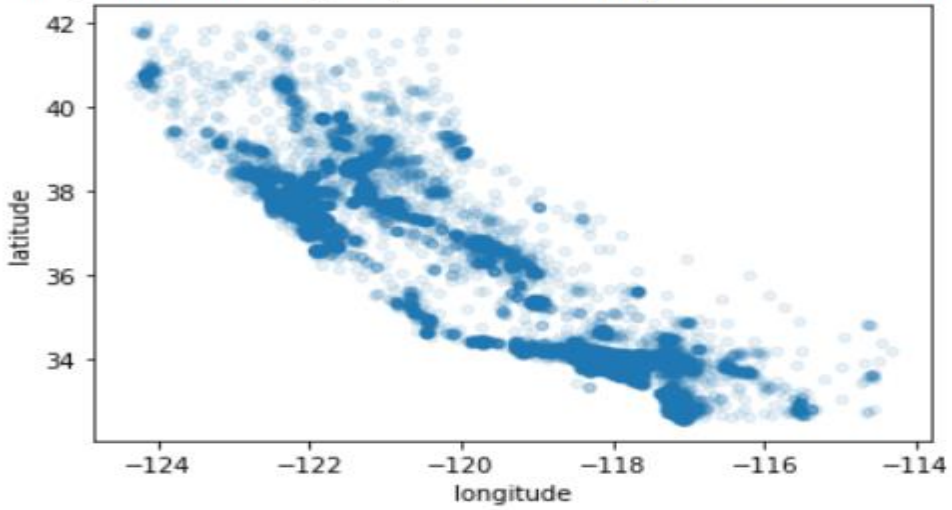
```
(20640, 10)
```

Şekil1.2 Veri setine ait temel istatistik özellikleri

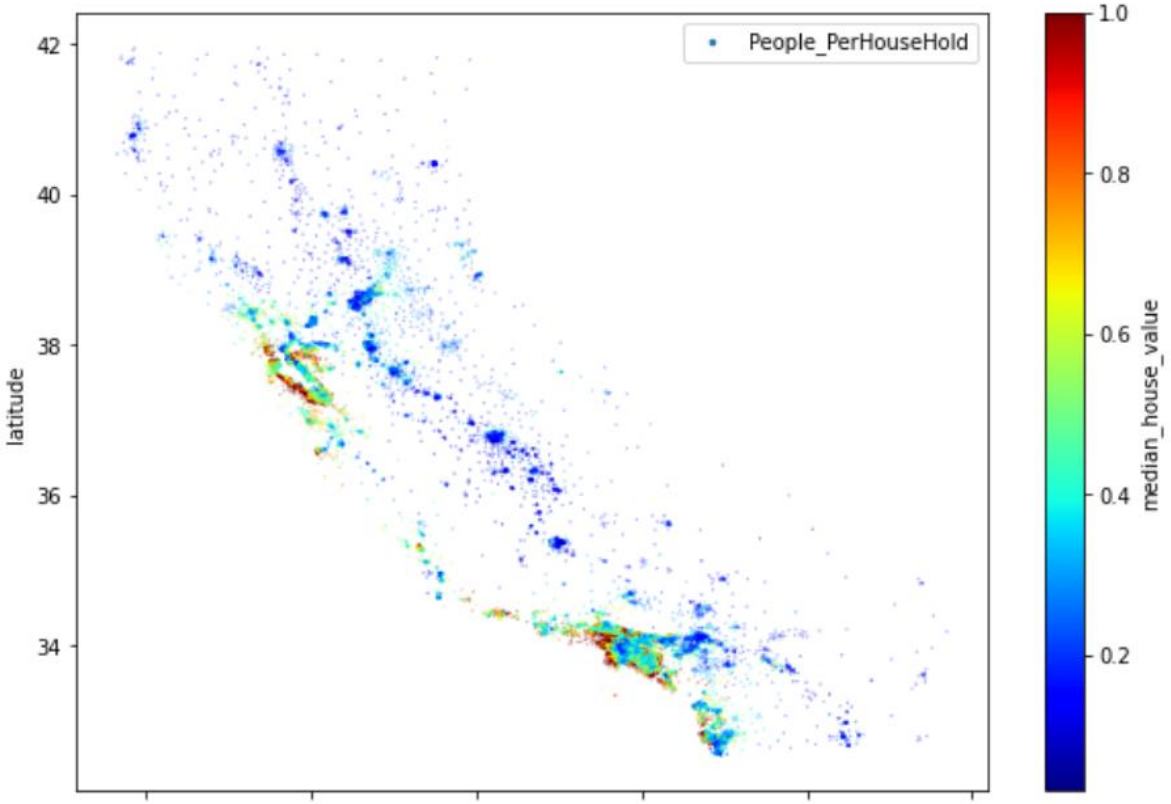
Şekil1.3'te görülebileceği gibi her nokta bir bölgeyi temsil ederken , noktaların koyuluğu da bölgedeki nüfusun yoğunluğuna(daha koyu nokta daha yoğun nüfusa işaret etmektedir) göre değişmektedir. Şekilin sol kısmında noktaların belirlediği sınır okyanusa karşılık gelmektedir, okyanusa daha yakın bölgelerin nüfuslarının daha yoğun olduğunu söyleyebiliriz. Buradan da sonuç olarak "ocean_proximity" özelliği her ne kadar sayısal bir değer ifade etmese de bölgedeki evlerin ortalama fiyatını belirlemede önemli bir katkısı olabileceğini çıkarabiliriz.

```
df.plot(kind="scatter", x="longitude", y="latitude",alpha=0.1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff463efa048>
```

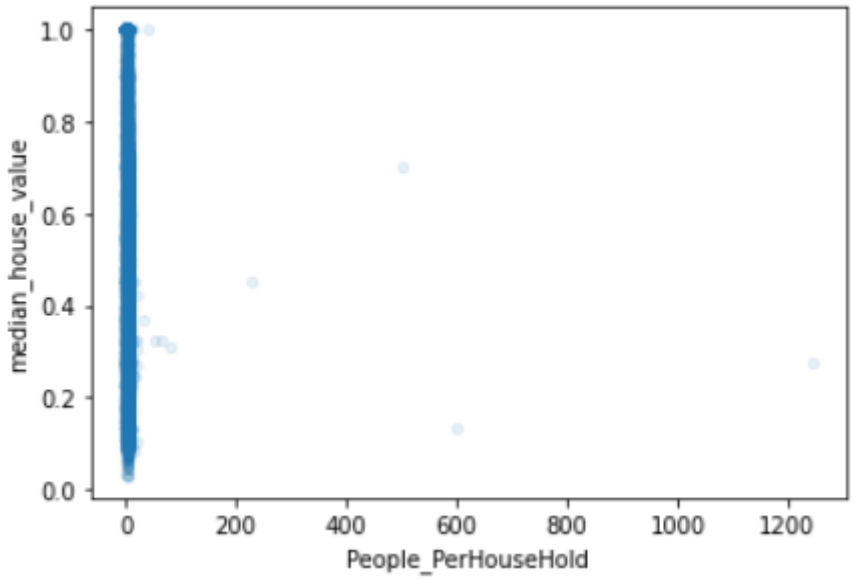


Şekil1.3 Coğrafi konuma göre Kaliforniya eyaletindeki bölgelerdeki(district) nüfus yoğunluğu



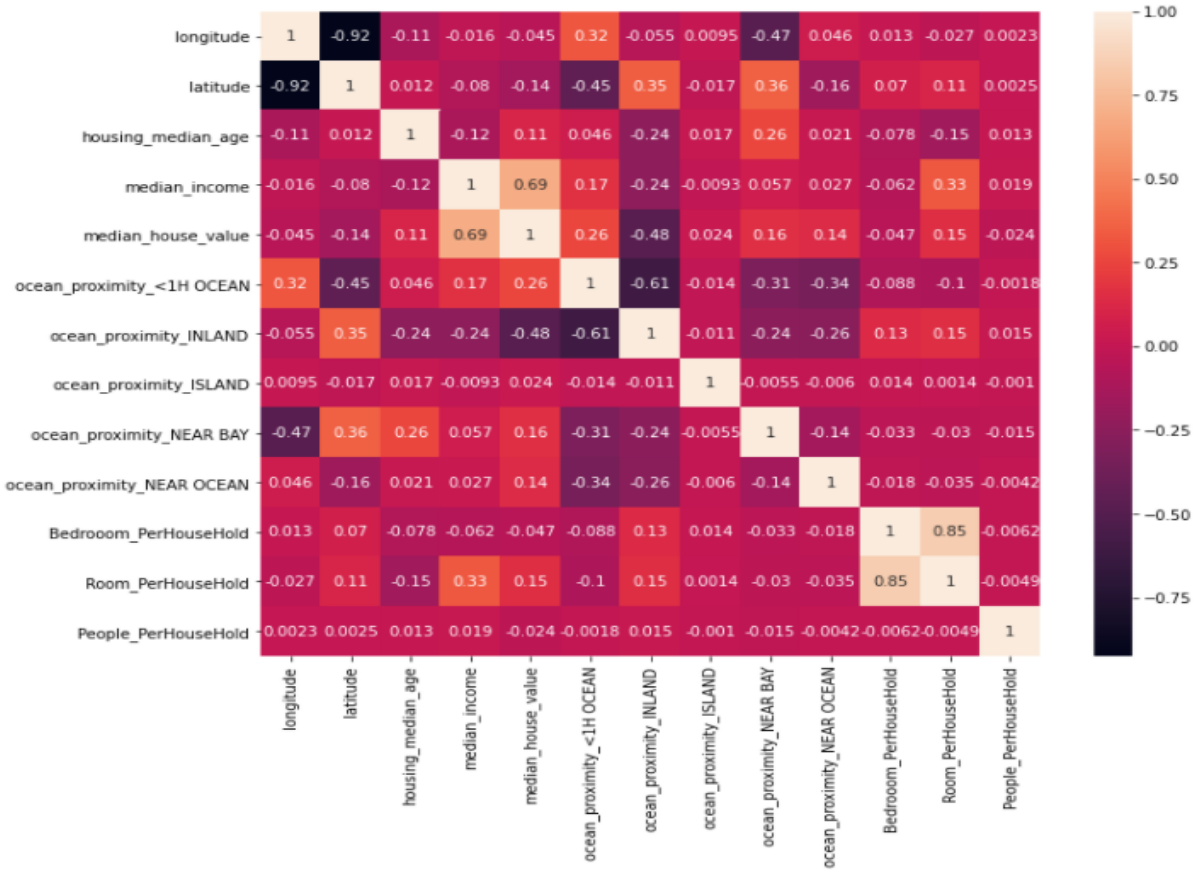
Şekil1.4 Coğrafi konuma göre bölgelerdeki ortalama ev fiyatları

Şekil1.4'e baktığımızda yukarıda yaptığımız varsayımın doğrulandığını görüyoruz, bölgenin okyanusa yakınlığı ile bölgedeki evlerin ortalama fiyatı arasında pozitif bir korelasyon var: Bölgenin okyanusa yakınlığı arttıkça ortalama ev fiyatı da yükselmektedir. Şekilden tam olarak belli olmasa da dairelerin çapı hane başına düşen kişi sayısını temsil etmektedir, fakat bu görselden bu nitelik ile ortalama ev fiyatı arasındaki ilişkiyi görmek mümkün olmamıştır. Bunu görmek için: "Population" ve "households" değerlerini kullanarak oluşturduğumuz ve her evdeki kişi sayısına karşı düşen "People_PerHouseHold" değeri ile ortalama ev fiyatları arasındaki ilişkiyi gösteren Şekil1.5'e baktığımızda iki parametre arasında herhangi bir korelasyon görülmemiştir. "median_house_value" özelliği çok katmanlı algılayıcı ağımızda çıkış katmanından önce sigmoid aktivasyon fonksiyonu kullandığımız için 0 ile 1 değerleri arasına sıkıştırılmıştır.



Şekil1.5 Hane başına düşen kişi sayısı ile ortalama ev fiyatı arasındaki ilişki

Şekil1.6'daki tablodan da teyit ettiğimizde hane başına düşen kişi sayısı ile ortalama ev fiyatı arasındaki korelasyon katsayı -0.024' eşittir. Ortalama ev fiyatıyla en çok ilişkili nitelik 0.69 katsayısıyla "ortalama gelir" olarak gözükmektedir. Daha önce belirtildiği üzere bölgenin okyanusa olan mesafesi de ortalama ev fiyatını önemli derecede etkilemektedir, Şekil1.6'dan kontrol edersek özellikle bölge okyanusa 1 saatlik mesafeden daha yakınsa ortalama ev fiyatı ile bu nitelik arasında 0.26'lık pozitif bir korelasyon bulunduğunu görebiliriz. Bu tablodan diğer niteliklerin de ortalama ev fiyatının belirlenmesinde ne kadar önemli bir rol oynadığını görebildik.



Şekil1.6 Özellikler arasındaki korelasyon tablosu

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Şekil1.6 Veri setindeki kayıp değerler

Toplam yatak odasına karşılık gelen sütunda 207 adet verinin eksik olduğunu görüyoruz.20640 örnekten oluşan veri seti için ihmal edilebilecek bir oran olduğunu düşündük ve veri setinden boş değerlere sahip örnekler kaldırıldı. Bu niteliğe sahip olmayan 207 bölgeyi veri setimizden kaldırmış olduk.

Girişlerimizden birisi (ocean_proximity) kategorik değişkenlerden oluşuyor. Veriyi eğitim için ağ'a sunabilmemiz için "ocean_proximity" sütunu sayısallaştırılması ve yerine 3 farklı sütun oluşturulması gerekiyor. Okyanusa uzaklık niteliği için aşağıdaki şekilde görüleceği üzere 5 farklı seçenek var . Bu seçenekleri sayısal olarak ifade etmenin bir yolu bu seçenekleri kendi aralarında ikili(binary) sayı sistemine göre kodlamaktır. Örneğin bölgenin okyanusa uzaklık niteliği "NEAR BAY" ise bu bölgenin sadece Near_Bay sütunu 1(hot) olacak diğer ilgili sütunlar 0 (cold) olacaktır. (One-Hot Encoding)

```
df.ocean_proximity.value_counts()
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

Şekil1.7 Okyanusa uzaklık için hangi kategoriden kaç örnek olduğunu gösteren tablo

Sütunun içerdiği kategorik değişkenler incelendiğinde, "ISLAND" değişkeninin sadece 5 tane olduğu görünüyor. Her değişken için farklı bir sütun oluşturulması eğitim süresini uzatabileceğinden ve sadece 5 tane bulunan bir değişkenin modeli aşırı öğrenmeye eğilimli hale getirebileceğinden dolayı bu değişkeni içeren örnekler kaldırıldı.

```
df2 =  pd.get_dummies(df1.ocean_proximity, prefix='ocean_proximity')
```

ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_NEAR BAY	ocean_proximity_NEAR OCEAN
0	0	1	0
0	0	1	0
0	0	1	0
0	0	1	0
0	0	1	0

Şekil1.8Kategorik değişkenler sayısallaştırıldıktan sonra verisetinde oluşan 4 yeni sütun

Eğitim ve testte daha iyi sonuç alınabilmesi için veri setinin girişlerinin ölçeklenmesi de faydalı olacaktır. Bunun için "sklearn" paketinden eğitim ve test sonuçlarına göre MinMaxScaler, StandardScaler ve RobustScaler kullanılmıştır ve StandardScaler ile RobustScaler ile daha iyi sonuçlar alındığı gözlemlenmiştir.

Standard Scaler	$\frac{x_i - \text{mean}(\boldsymbol{x})}{\text{stdev}(\boldsymbol{x})}$
MinMax Scaler	$\frac{x_i - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})}$
Robust Scaler	$\frac{x_i - Q_1(\boldsymbol{x})}{Q_3(\boldsymbol{x}) - Q_1(\boldsymbol{x})}$

Şekil1.9 Girişlerin ölçeklenmesi için kullanılabilecek yöntemler.

	longitude	latitude	housing_median_age	median_income
0	-122.23000	37.88000	41.00000	8.32520
1	-122.22000	37.86000	21.00000	8.30140
2	-122.24000	37.85000	52.00000	7.25740
3	-122.25000	37.85000	52.00000	5.64310
4	-122.25000	37.85000	52.00000	3.84620
5	-122.25000	37.85000	52.00000	4.03680

	0	1	2	3
0	-1.32731	1.05172	0.98216	2.34516
1	-1.32232	1.04236	-0.60621	2.33263
2	-1.33230	1.03767	1.85577	1.78294
3	-1.33730	1.03767	1.85577	0.93297
4	-1.33730	1.03767	1.85577	-0.01314
5	-1.33730	1.03767	1.85577	0.08721

Şekil1.10 Ölçeklemeden önce ve sonra veri setinin ilk 5 sütunu.

Bunun yanı sıra verisetindeki birbiriyle ilişkili sütunların ilişkisinden yola çıkarak yeni sütunlar oluşturulmuştur ve ilişkili olan asıl sütunlar kaldırılmıştır. Örneğin; popülasyon ve toplam hane sayısı sütunlarının yerine popülasyonu hane sayısına bölerek, hane başına düşen kişi sayısı sütunu eklenmiştir. Aynı işlemler toplam oda sayısı ve toplam yatak odası sayısı için de yapılmıştır.

```
df_encoded["Bedrooom_PerHouseHold"] = df_encoded["total_bedrooms"] / df_encoded["households"]
df_encoded.drop(columns=["total_bedrooms"],inplace=True)

df_encoded["Room_PerHouseHold"] = df_encoded["total_rooms"] / df_encoded["households"]
df_encoded.drop(columns=["total_rooms"],inplace=True)

df_encoded["People_PerHouseHold"] = df_encoded["population"] / df_encoded["households"]
df_encoded.drop(columns=["population","households"],inplace=True)
```

Şekil1.11 Sütunlar üzerinde yapılan manipülasyon

Son olarak, tahmin edilmesi gereken hedef değişkenimiz olan “median_house_value” sütunu maksimum değerine bölünerek 0 ile 1 arasına getirilmiştir. Böylece çok katmanlı

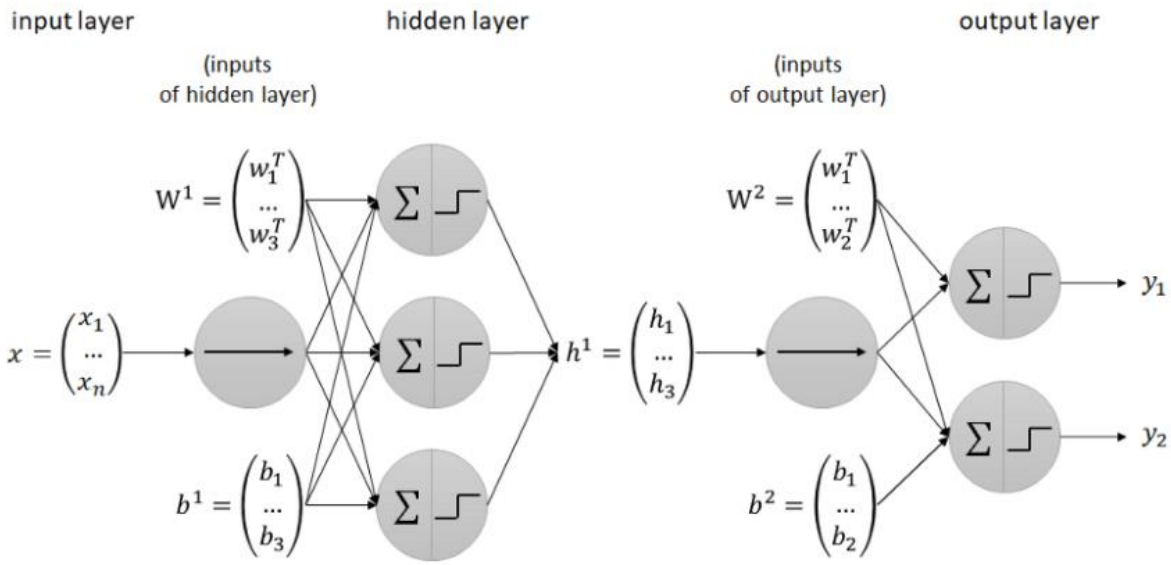
algılayıcımızın son katmanında aktivasyon fonksiyonu olarak “relu” nun yanısıra “sigmoid” ile kullanabiliriz.

2.Yöntem Önerisi

Problemimiz bir tür eğiticili öğrenme örneğidir. Ortalama ev fiyatının oluşmasında 8 farklı nitelik belirleyici olduğundan ve yukarıda Şekil1.6 korelasyon tablosunda gösterildiği üzere her nitelik tahmin edilmek istenen değeri farklı oranda etkilediğinden, bu etkiyi en iyi temsil eden ağırlık değerlerinin bulunması için giriş katmanından sonra gizli katmanlara ihtiyaç vardır. Giriş katmanı ve çıkış katmanı arasına sıralanmış gizli katmanlardan oluşan yapay sinir ağ yapısına çok katmanlı algılayıcı denmektedir. Bu ağda yapılanları ileri yol ve geriye yayılım olarak iki aşamada açıklayabiliriz:

1) İleri yol ağında her katmanın çıkışındaki değerler o katmana ilişkin ağırlıklarla çarpılarak bir değerler kümesi elde edilir bu kümenin tüm elemanları probleme göre belirlenen aktivasyon fonksiyonundan geçirilerek bir sonraki katmana giriş olarak sunulur. Aktivasyon fonksiyonu son katmanda çıkış değerlerine göre verinin sınıflandırılmasını sağlar. Her katman için aynı aktivasyon fonksiyonları tanımlanabileceği gibi bizim de bu problem çözerken kullandığımız gibi farklı aktivasyon fonksiyonları da seçilebilir.

2)Geriye yayılım algoritmasıyla yapılan , bu problem bir eğiticili öğrenme örneği olduğu için test kümesindeki her örüntünün tahmin edilen çıktıları ile eğitim için ağa verilen örüntülerin etiketleri arasındaki farkı hata olarak hesaplayıp bu hatayı her katmana ilişkin yerel gradyen terimleri, öğrenme hızı ve giriş değerlerini dikkate alarak ağırlıkların güncellenmesi için kullanılmasıdır.



Şekil2.1 Çok katmanlı algılayıcının temel yapısı

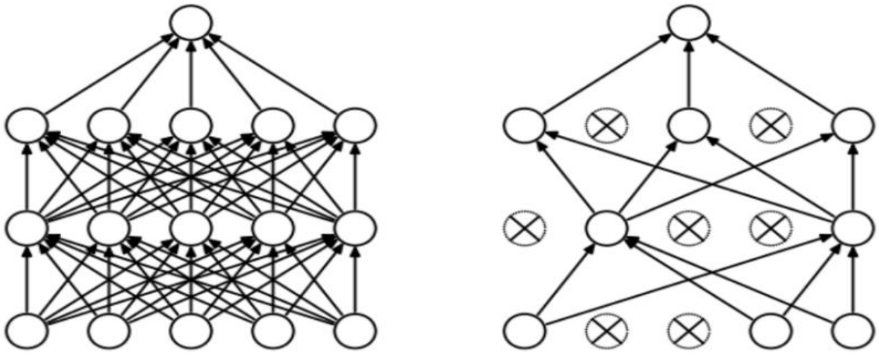
2.1 Dropout katmanı

Dropout, temel olarak yapay sinir ağı modelindeki aşırı öğrenmeye olan eğilimi önlemek için kullanılan bir düzenleme metodudur. Aşırı öğrenme kısaca; modelin eğitim setindeki veriyi az hata oranı ile başarılı bir şekilde öğrenip test verisinde bu başarıyı gösterememesidir. Yani bir nevi eğitim verisini doğru özelikleri ile öğrenemeyip ezberlemesi ve daha önce görmediği bir veri ile karşılaştığında başarısız tahminler yapmasıdır.

Bu metotta, bir dropout oranı belirlenir ve eğitim boyunca her iterasyonda, bu orana göre gizli katmandaki bazı nöronlar ihmal edilir yani sadece belirli nöronların ağırlıkları güncellenir. Metot, her iterasyonda farklı mimarilere sahip ağların eğitilmesine yol açar diyebiliriz. Peki burada aşırı öğrenme nasıl engellenir ? Örneğin, eğitim boyunca belirli oranda bazı ağırlıklar ihmal edildiğinden dolayı modelin özellikle bir girişe göre öğrenme eğilimi azaltılmış olur böylece nöronlar girişlerin gereksiz ayrıntılarını öğrenmez ve daha güvenli bir eğitim süreci gerçekleşir.

Dropout, çıkış katmanı için kullanılmaz, gizli katmanlar ile birlikte giriş katmanı için de kullanılabilir fakat bu da pratikte pek tavsiye edilen bir kullanım değildir. Bir başka önemli nokta ise, dropout eğitim sürecinde hem ileri yol hem geriye yayılım için kullanılırken, test aşamasında kullanılmaz. Eğitimden sonra verilerin girişlerini test için ileri yol ağına verdiğimizde tüm nöronlar ve ağırlıklar kullanılır. Buradaki önemli nokta ise, eğitim sürecinde dropout kullanılan gizli katmanlarda test için ileri yolda gittiğimizde; girişler ağırlıklar ile birlikte dropout oranı ile de çarpılır.

Örneğin dropout oranı olarak birinci gizli katmanda $p = 0.8$ kullanılması, eğitim sürecinde o gizli katmandaki nöronların her iterasyonda rastgele $(1 - p)$ oranı kadarının ihmal edilmesi, ağırlıklarının güncellenmemesi anlamına gelmektedir. Test aşamasında ileri yolda giderken bu katmandan geçildiğinde ise girişler ağırlıklarla çarpıldıktan sonra bir de dropout oranı $(p = 0.8)$ ile çarpılır.



Şekil2.2 Standart ve dropout katmanı uygulanan yapay sinir ağı

Numpy MLP için dropout katmanı implementasyonu

```
def feedForward(self,X,dropout1=0.3,dropout2=0.3):

    if dropout1 != 0 and dropout2 != 0:
        deactivated1=np.random.choice(self.firstLayer,size=int(dropout1*self.firstLayer),replace=False)
        deactivated2=np.random.choice(self.secondLayer,size=int(dropout2*self.secondLayer),replace=False)
        temp1=self.w1[deactivated1,:]; temp2=self.w2[:,deactivated1]
        temp3=self.w2[deactivated2,:]; temp4=self.w3[:,deactivated2]
        self.w1[deactivated1,:], self.w2[:,deactivated1] = 0,0
        self.w2[deactivated2,:], self.w3[:,deactivated2] = 0,0
```

Eğitim sürecinde ileri yolda gidilirken, belirlenen dropout oranına göre her iterasyonda rastgele seçilen nöronlara bağlı olan ağırlıklar sıfırlandı. Böylece, geriye yayılımda yerel gradyen hesabı yapılıp ağırlıklar güncellenirken, sıfırlanan ağırlıklara ait yerel gradyenler sıfır bulundu ve güncellenmedi.

```
if dropout1!=0 and dropout2!=0:
    self.w1[deactivated1, :], self.w2[:, deactivated1] = temp1, temp2
    self.w2[deactivated2, :], self.w3[:, deactivated2] = temp3, temp4
```

Bu sıfırlanan ağırlıkların asıl değerleri geçici değişkenlerde (temp1 ,temp2, temp3, temp4) tutulmuştu. Eğitimde ileri yolda gidiş tamamlandıktan sonra sıfırlanan değerler yerine asıl değerler konuldu. Böylece ihmal edilen (dropout) ağırlıklar, güncellenmeyip asıl değerinde kalmış oldu.

2.2 ReLu aktivasyon fonksiyonu

Konut fiyatları tahmini, bir regresyon problemi olduğu için gizli katmanların çıkışlarında aktivasyon fonksiyonu olarak ReLu (doğrultucu) kullanılacaktır. ReLu göründüğünün aksine doğrusal bir fonksiyon değildir, 0'dan küük değerler 0'a eşitlenirken 0'dan büyük değerler için tıpkı lineer aktivasyon fonksiyonu gibi davranır ve aynı değere eşitler. ReLu sınıflandırma problemlerinin gizli katmanlarında da kullanılabildiği gibi regresyon problemlerinde çıkış katmanında da kullanılır. ReLu'nun en büyük problemi; sıfır noktasında türevlenebilir olmamasıdır, hatayı minimize etmek için kullandığımız "en dik iniş yöntemi" için sıfırda bulunan değerler problem yaratabilir. Bunun için 0'daki türev keyfi olarak 0 ya da 1 seçilebilir.

```
def act(self, x, act_fonc):
    self.act_fonc=act_fonc
    if self.act_fonc=="sigmoid":
        y = 1.0 / (1 + np.exp(-x))
    elif self.act_fonc =="tanh":
        y = (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
    elif self.act_fonc=="relu":
        y = np.maximum(x,0)
    return y
```

```
def act_derivative(self, x, act_dx):
    self.act_dx=act_dx
    if self.act_dx=="sigmoid":
        y = x * (1.0 - x)
    elif self.act_dx =="tanh":
        y= 1 - x**2
    elif self.act_dx=="relu":
        y=np.where(x>0,1,0)
    return y
```

Şekil2.3 Numpy ReLU implementasyonu

Not: Verisetinde tahmin etmek istedimiz değer olan “median_house_value” sütunu maksimum değerine bölünerek, değerler 0 ve 1 arasında getirilebilir. Bu durumda, çok katmanlı algılayıcının çıkış katmanında aktivasyon fonksiyonu olarak ‘sigmoid’ kullanılarak da ağ test edilebilir.

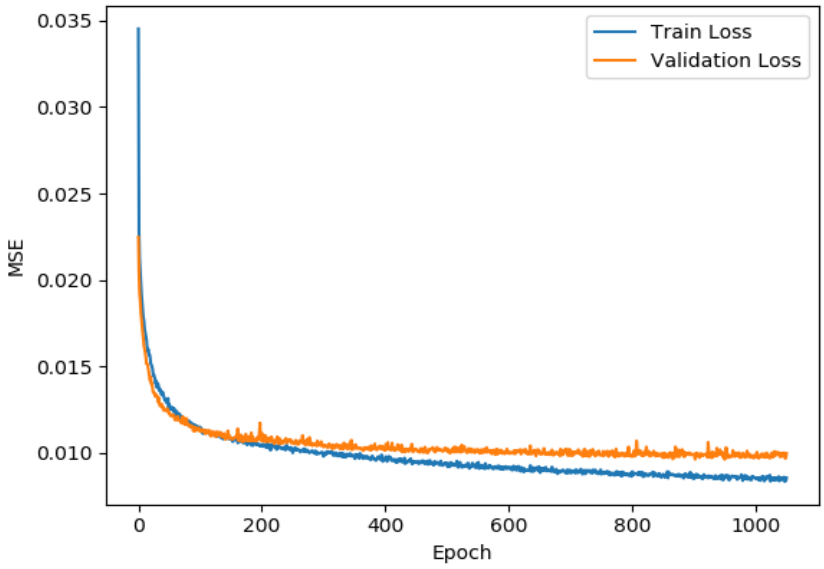
2.3 Değerlendirme kümesi(Validation set)

Değerlendirme kümesi, test kümesinden farklıdır. Veri, eğitim ve test olarak ayrıldıktan sonra eğitim setinin bir kısmı da doğrulama seti olarak ayrılır. Yani doğulama kümesi eğitim esnasında kullanılır. Veriler her iterasyonda eğitilip hiperparametreler ayarlanırken, doğrulama kümesi ile yeni bir veri için ne kadar başarılı olduğunu çeşitli metrikler ile test eder (ortalama kare hata, doğruluk). Değerlendirme kümesi, eğitim yapılırken ağırlık güncellemesi için kullanılmaz. Böylece bu küme ile eğitim esnasında tarafsız bir model başarılık değerlendirmesi yapılabilir.

Değerlendirme kümesinin bize sağladığı avantaj şu olacak: Ağı 500 iterasyon boyunca eğitmek istiyoruz ve her iterasyondan sonra değerlendirme kümesi için hata hesabı yapılıyor. Eğer belirli bir yerden sonra değerlendirme kümesi hatasında bir düşüş yaşanmıyor ve aksine hata artıyorsa, iterasyon sayısı tamamlanmadan eğitimi erken bitirebileceğiz. Böylece test kümesi için de bu sayede daha iyi bir sonuç elde edebiliriz.

```
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=100)
history = model.fit(X_train, y_train, batch_size=1024, epochs=500, validation_split=0.2, callbacks=[early_stop])
```

Şekil2.4 Değerlendirme kümesi ve erken durdurma kriteri implementasyonu



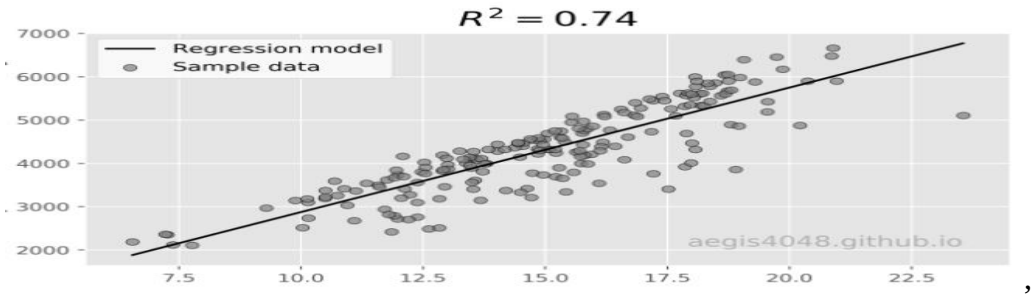
Şekil2.5 Değerlendirme kümesi ve eğitim kümesi hata grafiği örneği

3. Paket Program Kullanmadan Numpy ile Gerçeklediğimiz Çok Katmanlı Algılayıcı'nın Performans Ölçümü

Performans metriği olarak test kümesi için ortalama kare hata, determinasyon katsayısı (coefficient of determination) ve çok katmanlı algılayıcı tarafından tahmin edilen ortalama ev fiyatları ile gerçek ev fiyatlarını karşılaştıran grafik kullanılmıştır.

Determinasyon katsayısı(R2 Score)

Regresyon problemlerinde oluşturulan modelin verilen veriyi ne kadar iyi temsil ettiğini gösteren bir metriktir. Bizim problemimizdeki gibi çıkış parametresinin birden fazla değişken tarafından belirlendiği durumlarda çıkış değişkenindeki herhangi bir değişimin yüzde kaç olarak ifade edilebileceğini gösterir, 0 ile 1 arasında değer alır. Aşağıda Şekil2.5'te görülebileceği gibi 2 boyutlu bir regresyon probleminde , R2=0.74 olduğu durumda şekildeki doğru verideki varyansın yalnız %74'ünü temsil edebilmiştir.



Şekil3.1 2 boyutlu regresyon problemi için R2 skor örneği

3.1 Katmanlardaki nöron sayısına göre performans ölçümü

Nöron sayısı değişirken sabit tutulan diğer parametreler:

1)Öğrenme hızı=0.1.

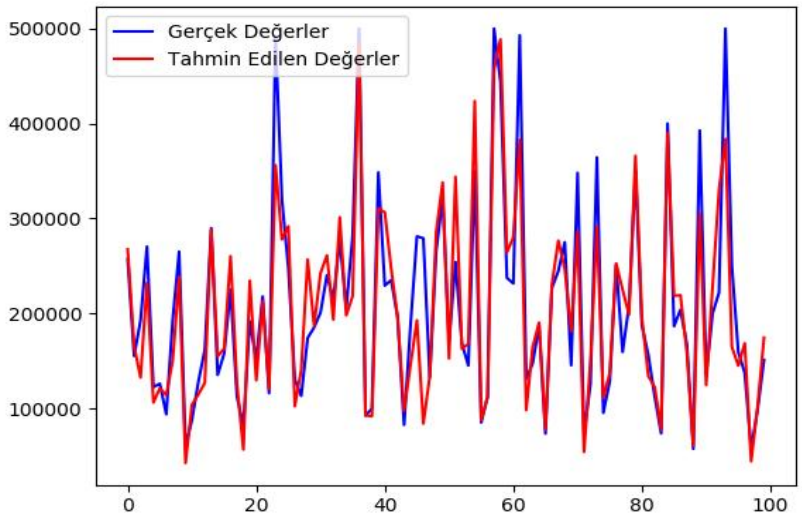
2)Her iki gizli katmanının çıkışına da dropout eklenmedi.

3)Eğitim aşaması 300 epochta tamamlandı.

Giriş ile çıkış katmanı arasına 2 gizli katman yerleştirdiğimizde ve her birinin nöron sayısını **64** olarak seçtiğimizde elde ettiğimiz sonuçlar:

```
Eğitim için Ortalama Kare Hata: [[0.00707282]], iterasyon sayısı: 300


r2 score: 0.8030377226467982
mean squared error: 0.01059728089297176
```

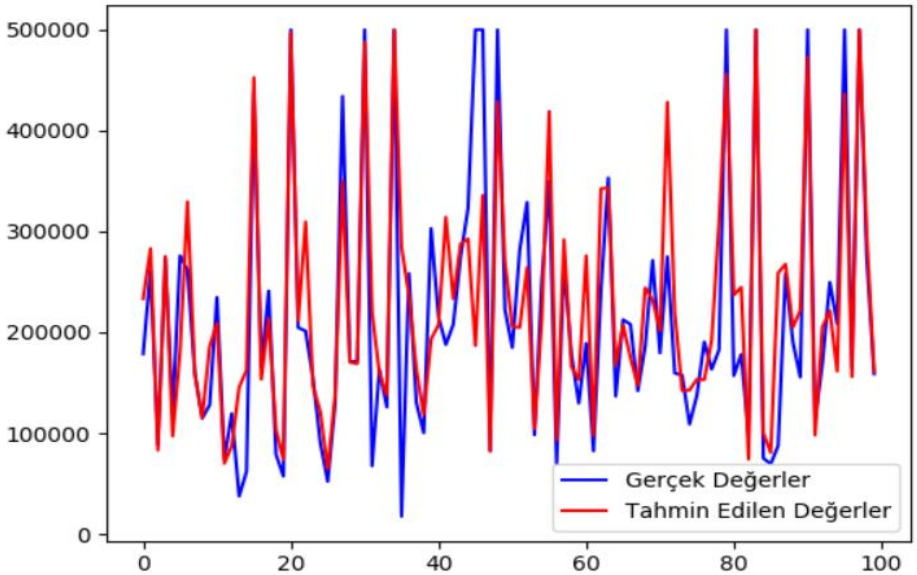


Şekil3.2 Her iki gizli katman için de nöron sayısı 64 olarak seçildiğinde

İlk katmanın nöron sayısını 32 ikinci katmanın nöron sayısını 16 olarak seçtiğimizde elde ettiğimiz sonuçlar:

```
Eğitim için Ortalama Kare Hata: [[0.00877118]], iterasyon sayısı: 300


r2 score: 0.7906361775118518
mean squared error: 0.011164735485737437
```

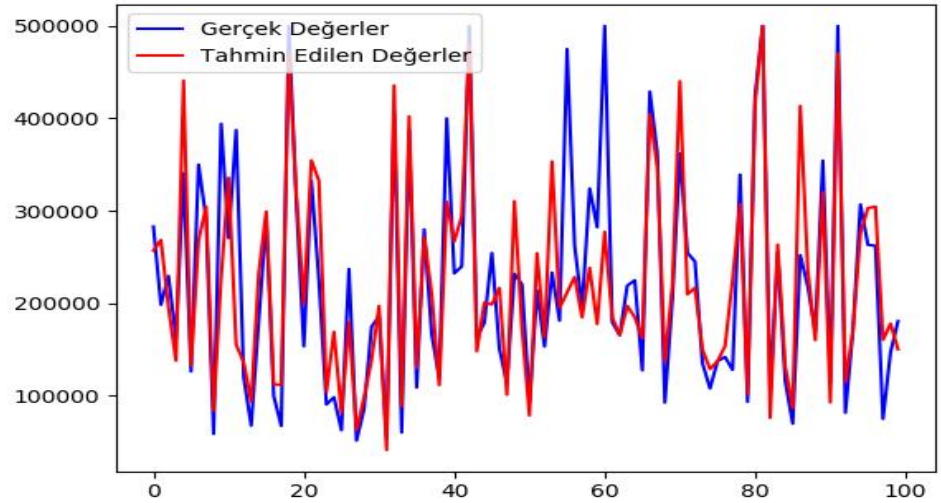


Şekil3.3 İlk katman 32 ikinci katman 16 nörondan oluştuğunda elde edilen sonuçlar

İlk katmanın nöron sayısını **8** ikinci katmanın nöron sayısını **8** olarak seçtiğimizde elde ettiğimiz sonuçlar:

```
Eğitim için Ortalama Kare Hata: [[0.01133432]], iterasyon sayısı: 300


r2 score: 0.7502385303247427
mean squared error: 0.013099642060963016
```



Şekil3.4 İlk katman 8 ikinci katman 4 nörondan oluştuğunda elde edilen sonuçlar

Ortalama ev fiyatını belirleyen 9 parametre bulunuyor, ilk katmanda 8 nöron olduğu durumda girişimizin boyutu nöron sayısından fazla olduğu için r2 skorun daha düşük olması tutarlı bir sonuçtur.

Ortalama kare hata ile determinasyon katsayısının ters orantılı olması beklediğimiz bir sonuçtu: Yüksek r2 skoru, verinin daha iyi temsil edildiğini gösterir bu durumda da tahmin edilen değerlerle gerçek değerler arasındaki mesafenin normu azalır. 3 farklı nöron sayılarıyla yapılan denemeler değerlendirildiğinde, ağın eğitimi için harcanan süre de dikkate alınarak, dropout katmanı eklenerek test edilecek ağın ilk katman 16 ikinci katman 8 nörondan oluşan ağ olması belirlenmiştir.

3.2 Dropout katmanı eklenmiş ağ için performans ölçümü

Dropout oranı değişirken sabit tutulan diğer parametreler:

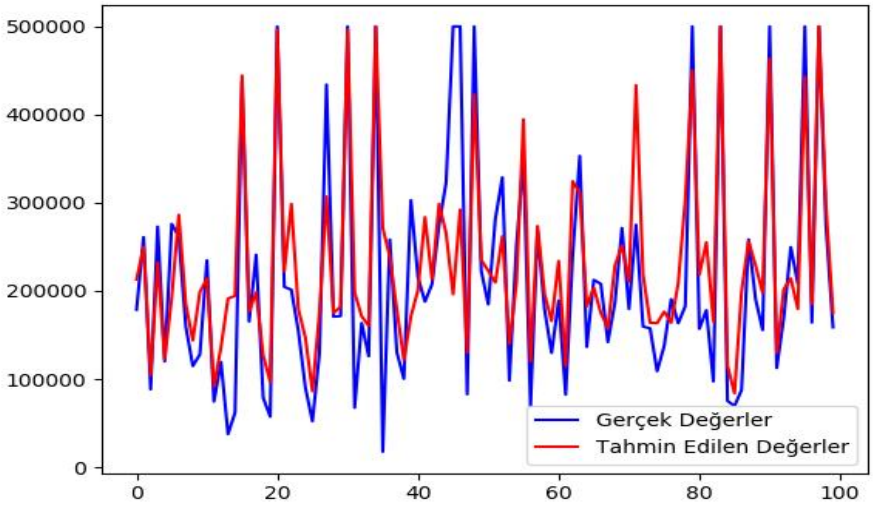
1)Öğrenme hızı=0.1

2) 3.1 bölümünde nöron sayıları üzerinde yapılan denemelere dayanarak nöron sayılarını İlk katmanda 32 ikinci katmanda 16 olacak şekilde seçtik. (Eğitim süresi de göz önüne alındı.)

3)Eğitim aşaması 300 epochta tamamlandı.

```
Eğitim için Ortalama Kare Hata: [[0.01200298]], iterasyon sayısı: 300


r2 score: 0.7604421524061147
mean squared error: 0.012905901027988546
```

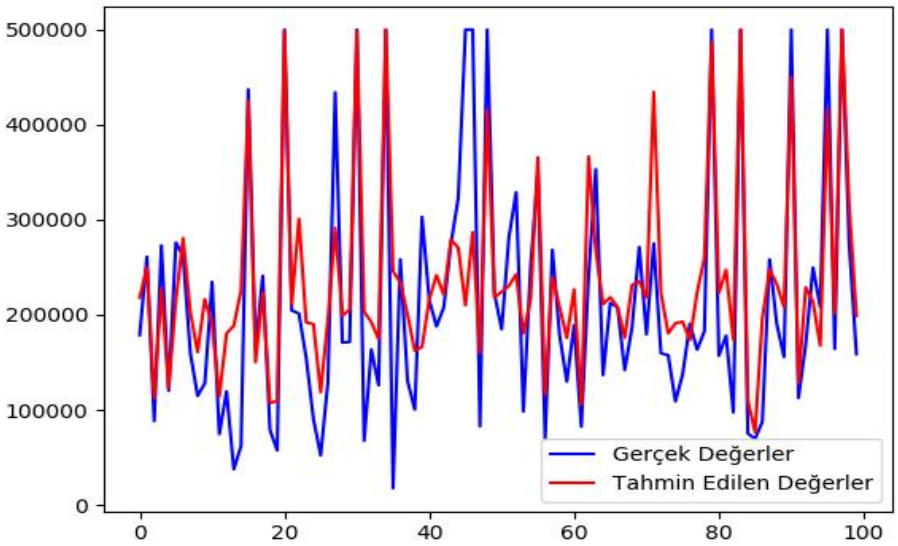


Şekil3.5 Her iki gizli katman çıkışına da 0.2 oranında dropout katmanı eklenince elde edilen sonuçlar

Dropout katmanı ile normal duruma göre daha iyi bir skor almadığımızı gözlemliyoruz. Fakat normal durumda son iterasyondaki eğitim hatası ile test hatası arasında fark vardı, test verileri için hatamız eğitimden daha yüksek çıkıyordu. Dropout katmanı ile daha iyi bir skor alınmasa da bu sorun çözülmüş oldu (eğitim ve test hatalarının çok yakın olduğu gözlemleniyor). Böylece, dropout'un aşırı öğrenmeyi önlediğini söyleyebiliriz.

```
Eğitim için Ortalama Kare Hata: [[0.01558495]], iterasyon sayısı: 300


r2 score: 0.6982799688412658
mean squared error: 0.01625481652722796
```

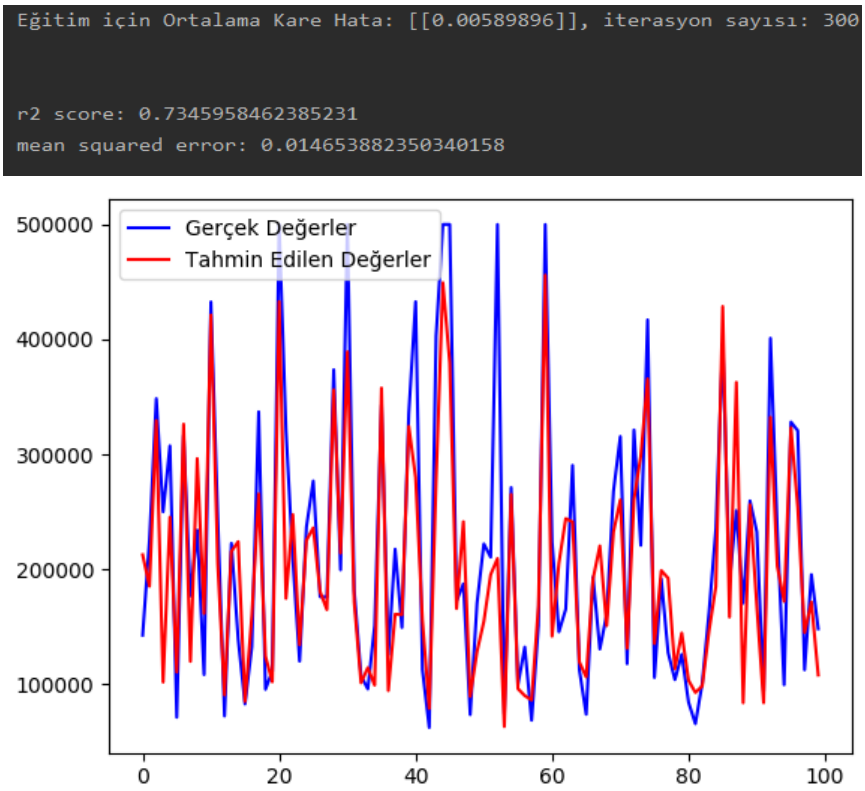


Şekil3.6 Her iki gizli katman çıkışına da 0.5 oranında dropout katmanı eklenince elde edilen sonuçlar

0.5 oranlı dropout kullanımında daha belirgin bir şekilde düşük bir skor aldık. Grafikte de uç değerlerin iyi tahmin edilemediği gözleniyor. Yine de burada da iyi bir sonuç olarak gösterebileceğimiz nokta, dropout katmanı kullanılmayan duruma göre eğitim ve test hatasının birbirine daha yakın olması olduğunu söyleyebiliriz.

3.3 Veri setinin daha az bir bölümü kullanılarak yapılan denemeler

Numpy kütüphanesi ile yazılan çok katmanlı algılayıcının en önemli sorunu, eğitim süresi olduğunu söyleyebiliriz. Dropout katmanı eklendiğinde de her iterasyonda ihmal edilen ağırlıkların belirlenmesinden dolayı eğitim süresi daha da uzuyor. Bu durumda, ağın performansını veri setini en iyi şekilde temsil eden 3000 tane örnek seçerek test ettik. Bu durumda eğitim süresi kısa süreceği için gizli katmanlara 64 nöron koyuldu ve dropout'suz ve dropout'lu durumlar karşılaştırıldı.

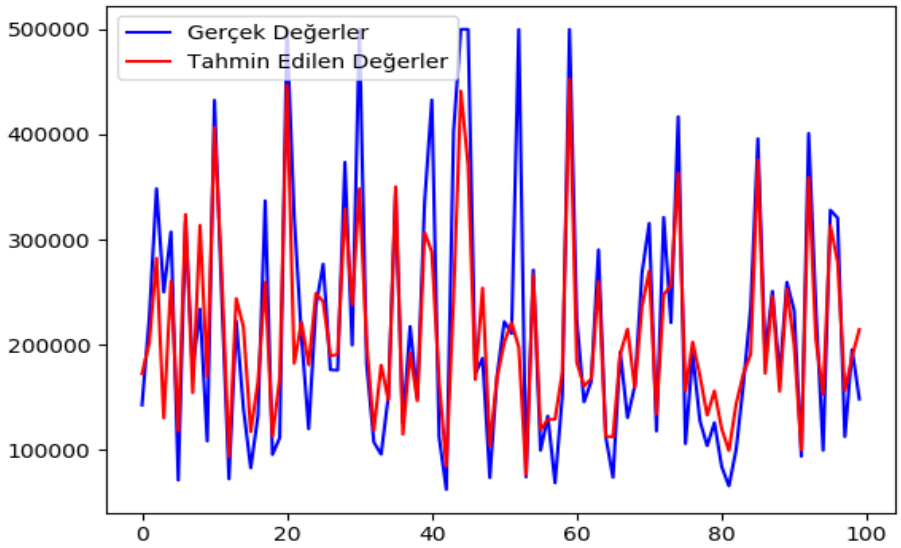


Şekil3.7 0.1 öğrenme hızı ve 300 iterasyon ile 3000 örnek için dropoutsuz test sonuçları

3000 örnek ile de çok kötü bir skor almadık fakat eğitim hatası test hatasından yaklaşık 3 kat daha düşük bunun önemli bir sorun olduğunu ve ağın bu şekilde yine aşırı öğrenmeye eğilimli olduğunu söyleyebiliriz.

```
Eğitim için Ortalama Kare Hata: [[0.01066791]], iterasyon sayısı: 300


r2 score: 0.7658243423279285
mean squared error: 0.012929648945600477
```



Şekil3.8 0.1 öğrenme hızı ve 300 iterasyon ile 3000 örnek için dropout katmanı (0.2) eklenmiş test sonuçları

3000 örnek kullanarak yaptığımız eğitim ve testlerde dropout katmanı ile daha iyi skor aldığımızı ve eğitim ile test hatalarını birbirine yaklaştırdığımızı gözlemliyoruz.

Yine de, veri setimiz belirli bir bölgedeki tüm enlem ve boylamlara ilişkin bilgiler içerdiğinden, veri setine yeni veri eklemek ya da veri setini küçültmek pek doğru bir yaklaşım olmayacaktır. Bunu, verinin tamamını kullandığımızda daha iyi sonuçlar aldığımızı gözlemleyerek de anlayabiliyoruz. Buradaki amacımız, veri setini daha hızlı eğitip test ettiğimizdeki sonuçları karşılaştırmaktı.

Çok katmanlı algılayıcımız, "veri uyarlamalı" olarak çalıştığından dolayı tüm veri kullanıldığında yaklaşık 4800000 kez ağırlık güncellemesi yapıyor (Eğitim verisi sayısı(16000)*iterasyon sayısı(300)) . Bu da, ağın hazır paketler kullanılarak gerçeklenen ağa göre daha yavaş çalışmasına sebep oluyor.

4. Tensorflow-Keras Paketi ile Gerçeklenen Çok Katmanlı Algılayıcı (MLP) Performans Ölçümü

Keras ile gerçeklenen MLP grup uyarlamalı modda (batch mode) çalıştığı için eğitim süremiz daha kısa sürüyor. Bu yüzden gizli katmanlardaki nöron sayısını ve maksimum iterasyon sayısını daha yüksek seçerek deneyebiliriz. Ayrıca ağırlıkların güncellenmesi, hatanın minimize edilmesinde farklı optimizasyon algoritmaları da kullanabiliriz.

Not: Test sonuçlarındaki skor ve ortalama kare hata değerleri, Ağ 5 kez çalıştırılıp ortalama değerleri alınarak hesaplanmıştır.

4.1 Farklı optimizasyon algoritmalarına göre performans ölçümü

Optimizasyon algoritmaları değişirken sabit tutulan diğer parametreler:

1)Aktivasyon fonksiyonu olarak Çıkışta sigmoid, gizli katmanlarda ReLu kullanıldı

2)Her iki gizli katmanının çıkışına da dropout eklenmedi.

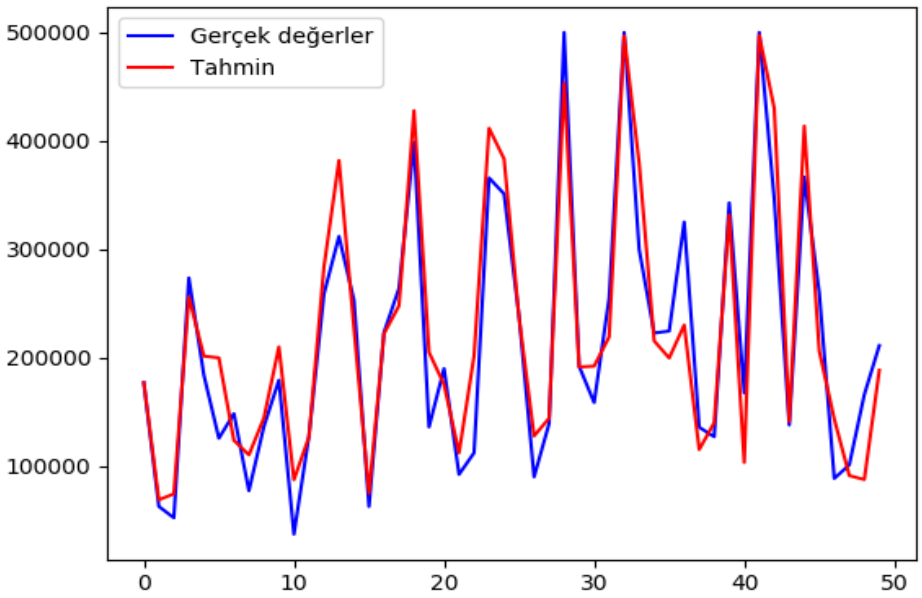
3)Maksimum iterasyon sayısı 1000.

4)0.2 oranında değerlendirme kümesi oluşturdu ve değerlendirme hatasına göre erken durdurma koşulu belirlendi

5)İlk gizli katman 128, 2. Gizli katman 64 nörondan oluşuyor.

```
Epoch 2000/2000
14707/14707 [==============================] - 0s 5us/step - loss: 0.0132 - mse: 0.0132


r2 score: 0.7837401247574731
mean squared error: 0.011624562063437892
```

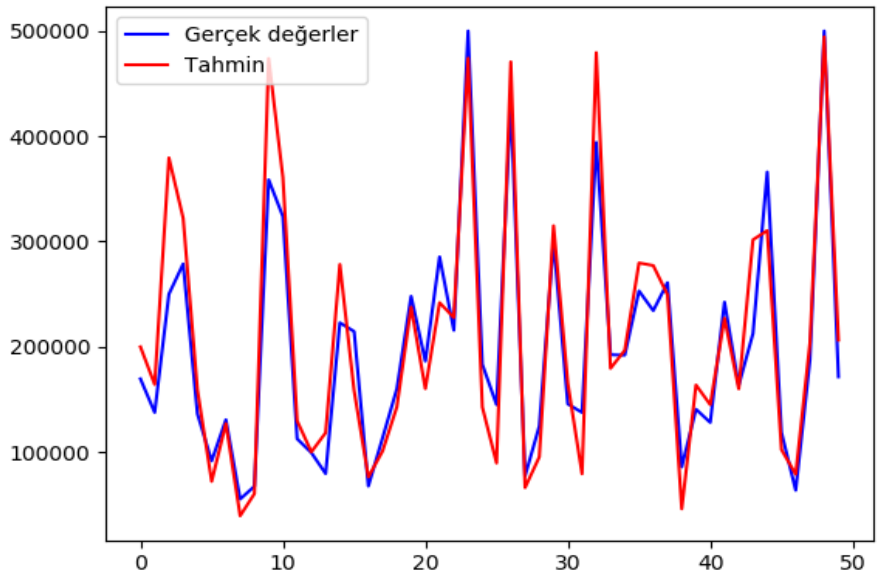


Şekil4.1 Stochastic gradient descent (0.1 öğrenme hızı) algoritması ile elde edilen sonuçlar

Gradient Descent methodu ile yapılan eğitimde erken durdurma koşulu sağlanmadı ve maksimum iterasyon sayısına kadar eğitim devam etti. Hatamızın yavaş bir şekilde düştüğünü gözlemledik.

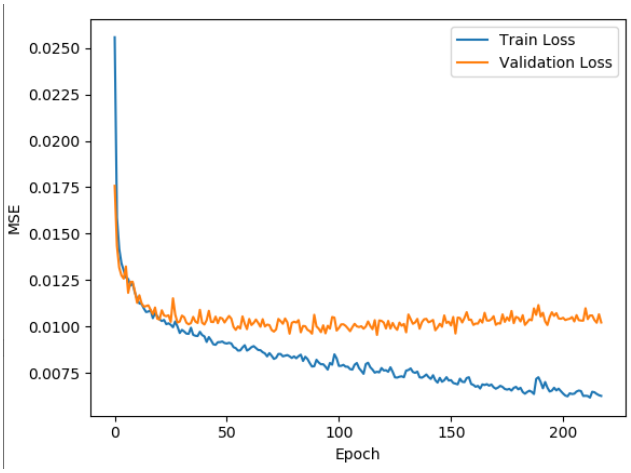
```
Epoch 218/2000
14707/14707 [==============================] - 0s 3us/step - loss: 0.0063 - mse: 0.0063


r2 score: 0.7897579007810729
mean squared error: 0.011506380423175581
```



Şekil4.2 Adam optimizasyon algoritması (0.01 öğrenme hızı) ile elde edilen sonuçlar

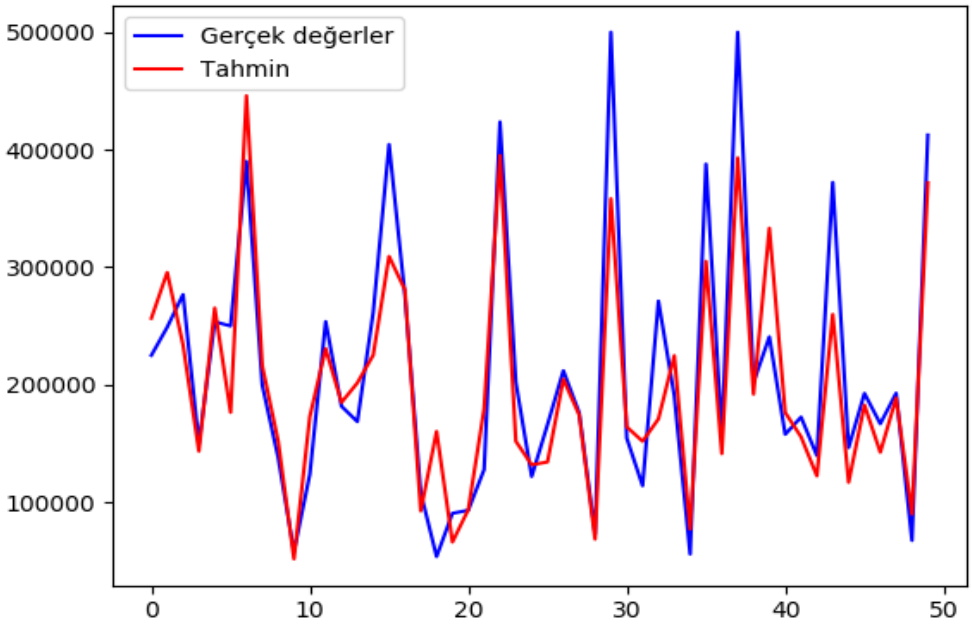
Adam optimizasyon methodunda ise hatanın daha hızlı bir şekilde minimize edildiğini ve eğitimin henüz 200'lü iterasyon sayılarında erken durdurma koşuluna girdiğini görüyoruz. Burada da yine eğitim ve test hatası arasında önemli bir fark gözlemliyoruz fakat skor ve eğitim süresi açısından daha iyi sonuç aldığımız söylenebilir.



Burada da değerlendirme kümesi hatasının bir yerden sonra artma eğilimine girdiğini ve bu noktada erken durdurma koşulunun devreye girdiğini söyleyebiliriz.

```
Epoch 319/2000
14707/14707 [==============================] - 0s 3us/step - loss: 0.0090 - mse: 0.0090


r2 score: 0.8031030511621071
mean squared error: 0.010217759321681537
```



Şekil4.3 RMSProp optimizasyon algoritması (0.001 öğrenme hızı) ile elde edilen sonuçlar

RMSProp algoritmasında dropout katmanı kullanılmamasına rağmen çok iyi bir eğitim ve test hata sonucu aldığımızı söyleyebiliriz. Yine ağın erken durdurma koşuluna girerek eğitimin erken bitirildiğini görüyoruz. Burada alınan skor şimdiye kadar alınan en iyi skor oldu.

4.2 Gizli katmanlarda farklı nöron sayılarına göre performans ölçümü

Nöron sayıları değişirken sabit tutulan diğer parametreler:

1)Aktivasyon fonksiyonu olarak Çıkışta sigmoid, gizli katmanlarda ReLu kullanıldı

2)Her iki gizli katmanının çıkışına da dropout eklenmedi.

3)Maksimum iterasyon sayısı 3000.

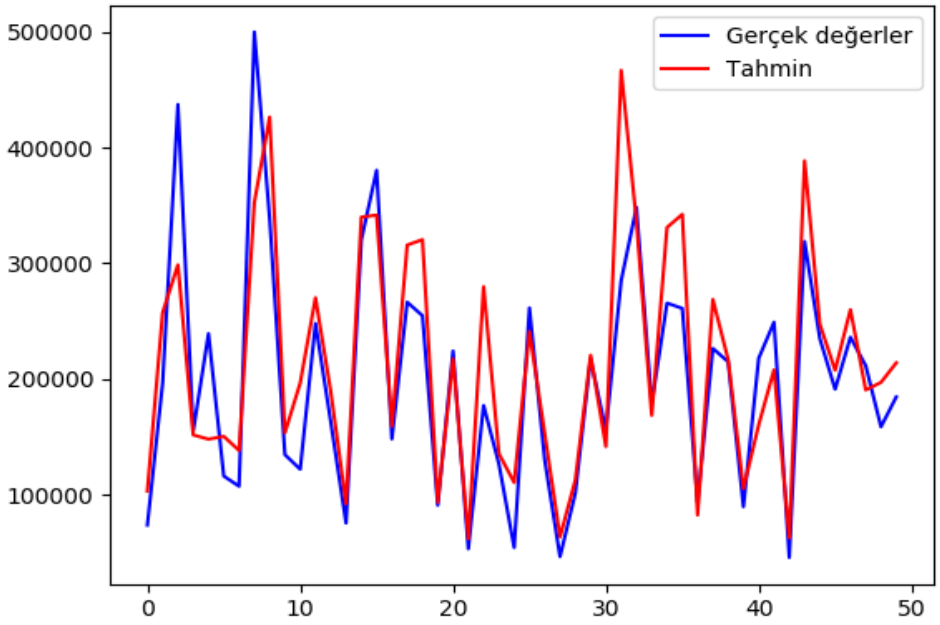
4)0.2 oranında değerlendirme kümesi oluşturdu ve değerlendirme hatasına göre erken durdurma koşulu belirlendi

5)RMSProp optimizasyon algoritması kullanıldı.

Not: En son yapılan denemede nöron sayıları 128, 64 seçildiği için burada tekrar denenmeyecektir.

```
Epoch 431/2000
14707/14707 [==============================] - 0s 5us/step - loss: 0.0068 - mse: 0.0068


r2 score: 0.7754481625834391
mean squared error: 0.011758505650411734
```



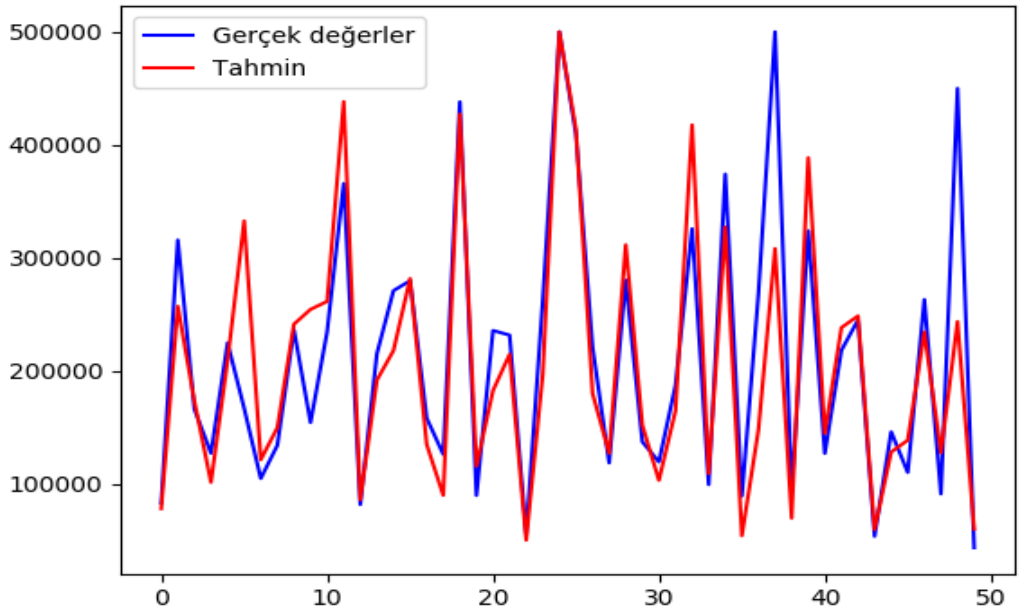
Şekil4.4 İlk katman 256 ikinci katman 128 nöron için elde edilen sonuçlar

Katmanlardaki nöron sayıları 2 katına çıkarıldığında daha düşük bir eğitim hatası almamıza karşın daha yüksek bir test hatası ve daha düşük bir skor elde edildi. Bu yüzden diğer denemelerde 128 ve 64'ün altına inerek denemek daha mantıklı olacaktır.

İlk katman 64 ikinci katman 32 nöron için elde edilen sonuçlar:

```
Epoch 627/2000
14707/14707 [==============================] - 0s 3us/step - loss: 0.0085 - mse: 0.0085


r2 score: 0.8093647862741901
mean squared error: 0.010564547322985673
```



Şekil4.5 İlk katman 64 ikinci katman 32 nöron için elde edilen sonuçlar

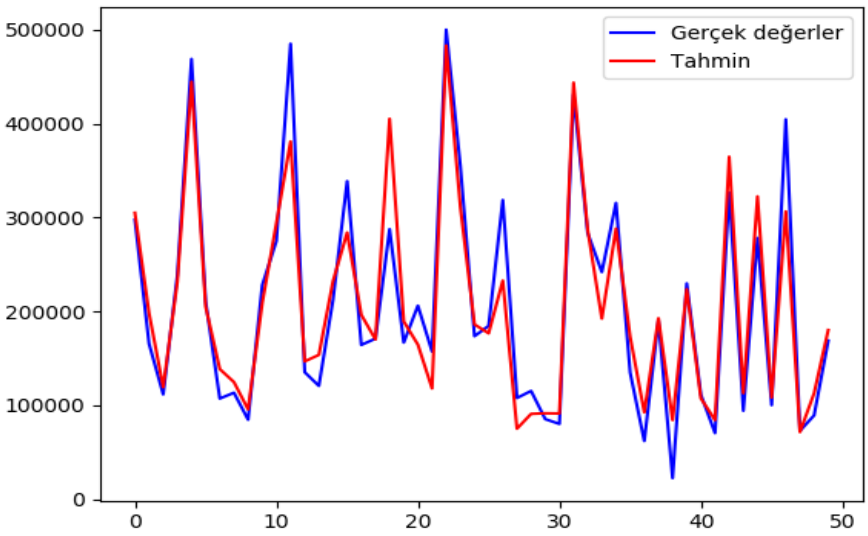
128 ve 64 nöron bulunan duruma göre birbirine benzer ve yakın sonuçlar aldığımızı gözlemliyoruz.

4.3 Dropout Katmanı ile göre performans ölçümü

Son olarak ise amacımız, dropout katmanını da işin içine dahil ederek test hatasını olabildiğince eğitim hatasına yaklaştırmak ve daha iyi bir skor elde etmek. Dropout katmanı eklenerek yapılan eğitimde de önceki parametreler seçildi. Bunun yanında optimizasyon algoritması RMSProp, gizli katman nöron sayıları 128'e 64 olarak belirlendi.

```
Epoch 519/2000
14707/14707 [==============================] - 0s 5us/step - loss: 0.0092 - mse: 0.0092


r2 score: 0.8340285642699671
mean squared error: 0.008781997861373195
```

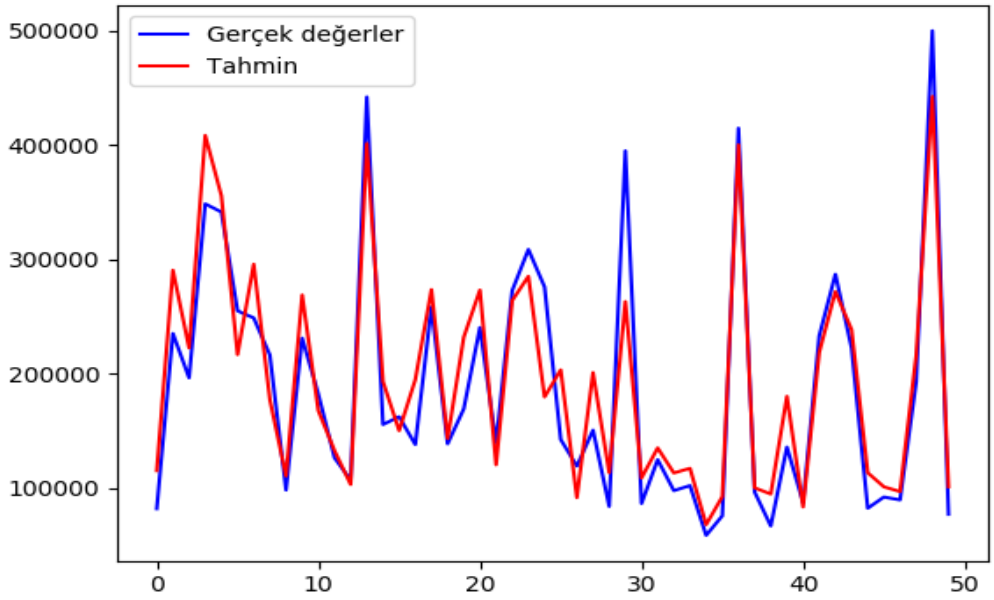


Şekil4.6 Gizli katman çıkışlarında 0.2 oranında dropout kullanılarak elde edilen sonuçlar

Amacımıza ulaştığımızı ve şu ana kadar yapılan denemelere göre en iyi skoru elde ettiğimizi görüyoruz. Peki dropout oranını biraz daha yükselttiğimizde daha iyi bir sonuç alabilir miyiz? Görelim.

```
Epoch 513/2000
14707/14707 [==============================] - 0s 5us/step - loss: 0.0094 - mse: 0.0094


r2 score: 0.8211638878802558
mean squared error: 0.009574715498893313
```



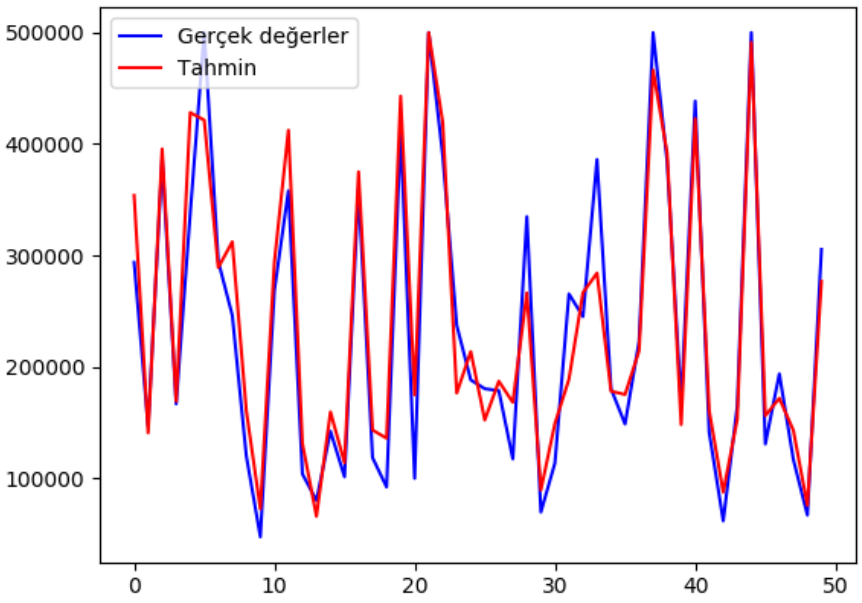
Şekil4.7 Gizli katman çıkışlarında 0.3 oranında dropout kullanılarak elde edilen sonuçlar

Kısmen daha düşük bir skor aldığımızı görsek de, burada da test hatası ile eğitim hatasını aynı noktaya getirdiğimizi söyleyebiliriz.

Sırasıyla gizli katman çıkışlarında 0.5 ve 0.2 oranında dropout kullanılarak elde edilen sonuçlar:

```
Epoch 533/2000
14707/14707 [==============================] - 0s 5us/step - loss: 0.0105 - mse: 0.0105


r2 score: 0.8077709085224916
mean squared error: 0.010400932650039005
```



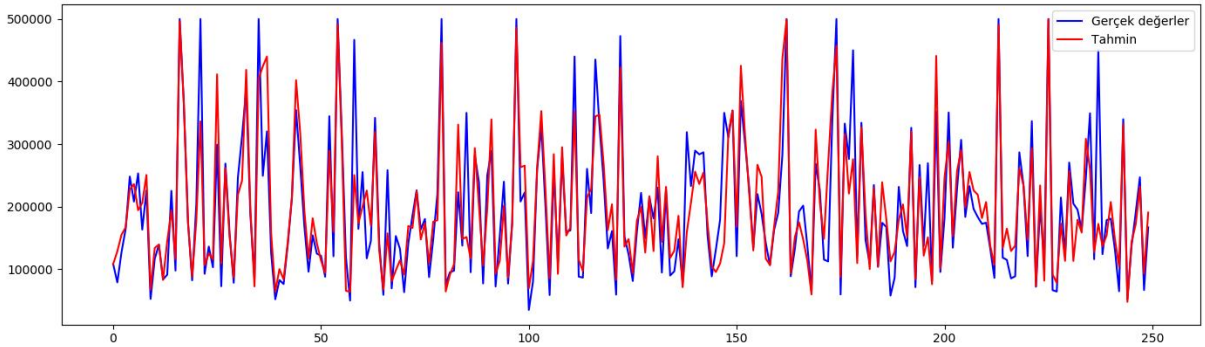
Şekil4.8 Sırasıyla gizli katman çıkışlarında 0.5 ve 0.2 oranında dropout kullanılarak elde edilen sonuçlar

Dropout oranını 0.5'e yükselttiğimizde de ağın test hatasının eğitim hatasının üstünde olmadığını görüyoruz. Fakat oranı arttırdıkça eğitim hatasının daha yüksek çıktığını ve skorumuzun düştüğünü görüyoruz.

Bu durumda farklı parametrelerle yapılan denemeler sonucu optimum çok katmanlı algılayıcımızın parametreleri ve özellikleri aşağıdaki gibidir:

1) Optimizasyon algoritması: RMSprop.
2) 2 gizli katmandan oluşuyor. Nöron sayıları: 128, 64.
3) Her iki katman için Dropout oranı: 0.2.
4) Aktivasyon fonksiyonları: Gizli katmanlar için ReLU, çıkışta sigmoid (çıkışta da ReLU kullanıldığında benzer sonuçlar elde ediliyor. Biz, çıkışımızı maksimum değere bölerek 0 ve 1 arasına ölçeklediğimiz için çıkışta sigmoid tercih ettik).
5) Maksimum iterasyon sayısı: 2000
6) Bir grupta güncellenen ağırlık sayısı (batch_size): 1000

Bonus: En iyi modelimiz için 250 test ve tahmin verisi karşılaştırma grafiği:



Şekil4.9 Elde ettiğimiz optimum model için gerçek ve tahmin değerlerini gösteren grafik

5.SONUÇ

Projemizde, Çok Katmanlı Algılayıcı (MLP) ile 1990 nüfus sayım verileri kullanılarak hazırlanan California Konut Fiyatları verisetini kullanarak regresyon analizi ile fiyat tahmini yaptık. Amacımız yapay sinir ağı kullanarak daha önce bu problemin çözümü için denenmiş Makine Öğrenmesi yöntemlerinden (Lineer Regresyon, Karar Ağaçları, Rastgele Karar Ormanı) daha iyi bir sonuç elde etmekti.

Öncelikle veri setini iyileştirmek ve eğitim için uygun forma getirebilmek için veri önişleme yapıldı. Burada bazı sütunların birbirleriyle olan ilişkiler incelenerek bazı değişiklikler yapıldı ve kategorik değişkenler sayısallaştırıldı. Son olarak ise veriseti ölçeklenerek (Standard Scaler) eğitime hazır hale getirildi.

Problemin çözümü için daha önce numpy paketi ile sıfırdan yazdığımız MLP'ye ek olarak Tensorflow-Keras ile gerçeklenen MLP yapısını da kullandık. İki ağ yapısı için de model farklı parametrelerle çalıştırılarak (optimizasyon algoritmaları, gizli kataman nöron sayıları, dropout katmanı oranı) elde edilen eğitim ve test sonuçları incelendi. Her iki MLP yapısı için de nöron sayılarının ve Dropout katmanının etkisi gösterildi. Dropout katmanı kullandığımız durumlarda, normal duruma göre eğitim ve test için ortalama kare hata değerinin birbirine daha tutarlı halde olduğu gözlendi.Yani gizli katman çıkışlarında Dropout kullanarak, modelin aşırı öğrenmeye olan eğiliminin önlendiği söylenebilir.

Numpy MLP ile en iyi durumda **0.80** determinasyon katsayısı (r-squared score) elde ederken, Keras ile gerçeklenen MLP'de en iyi durumda **0.83** skorunu elde ettik. En iyi skoru aldığımız ağ yapısında eğitim ve test için aldığımız yaklaşık ortalama kare hata değeri **0.009** civarında gözlemledik. Burada belirtilmesi gereken nokta; hata değeri, tahmin edilen fiyat değerinin 0 ve 1 arasına ölçeklendiği durum için bu şekilde çıkarken, çıkışlar gerçek değerine getiriliğinde daha yüksek çıkmaktadır. Fakat skor (determinasyon katsayısı) her iki durum için de aynı elde edilmektedir bu yüzden öncelikli değerlendirme metriği olarak skor kullanıldı.

Makine Öğrenmesi yöntemleri ile yapılan bazı çalışmalar incelendiğinde en iyi sonucun Rastgele Karar Ormanları ile elde edildiğini ve yaklaşık **0.79-0.80** skoru alındığı gözlemlendi. Bu anlamda, yapay sinir ağı kullanarak daha iyi bir sonuç elde ettiğimiz söylenebilir.

Sonuç olarak, problem çözümü için önerilen çok katmanlı algılayıcı ağ yapısı ve parametreleri aşağdaki gibidir:

- Giriş katmanı 12 nöron, çıkış katmanı 1 nöron. Gizli katmanlar sırasıyla 128, 64 nöron
- İki gizli katman için de 0.2 oranlı Dropout Katmanı
- Optimizasyon Algoritması: RMSprop (0.001 öğrenme hızı)
- Aktivasyon fonksiyonu: Gizli katman çıkışı için ReLu, çıkışta sigmoid (pozitif değerli çıkışlar maksimum değere bölünerek 0 ve 1 arasına ölçeklendiği için).
- Maksimum iterasyon sayısı 2000 ve değerlendirme kümesi hatasına göre erken durdurma.
- Grup uyarlamalı eğitim.

KAYNAKLAR

Géron Aurélien, in *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*, Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo: O'Reilly, 2019, pp. 37–85.

D. H, "Mathematical Representation of a Perceptron Layer (with example in TensorFlow)," *Medium*, 31-Jan-2019. [Online]. Available: https://medium.com/@daniel.hellwig.p/mathematical-representation-of-a-perceptron-layer-with-example-in-tensorflow-754a38833b44.

"California Housing Prices," *California Housing Prices*. [Online]. Available: https://jmyao17.github.io/Kaggle/California_Housing_Prices.html.

Basic regression: Predict fuel efficiency | TensorFlow Core. (2021). From https://www.tensorflow.org/tutorials/keras/regression

Brownlee, J. (2018). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. From https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

About Train, Validation and Test Sets in Machine Learning. (2020). From https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7