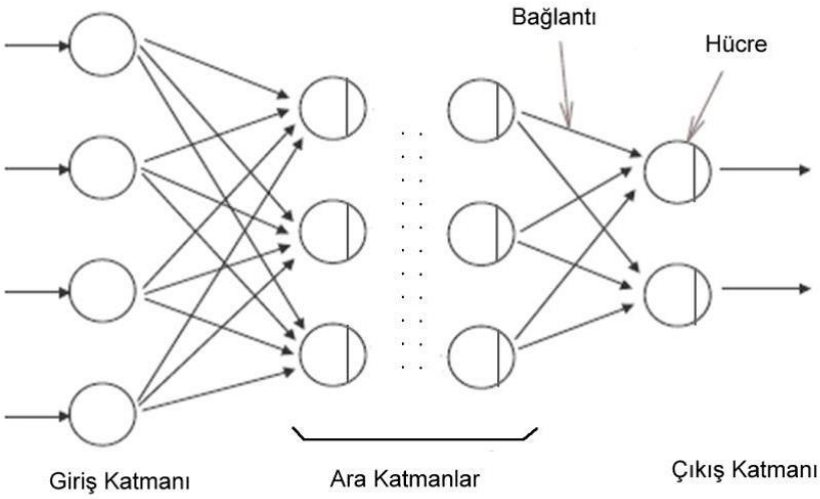
İçindekiler

Derin Öğrenme

Derin öğrenme, bir veya daha fazla gizli katman içeren yapay sinir ağları ve benzeri makine öğrenme algoritmalarını kapsayan çalışma alanıdır.



Şekil 1 Ağ Yapısı

Ağ Yapısı

Derin öğrenmede ağ yapısı yukarıda görüldüğü gibidir. Ağ yapısı üç farklı katmana ayrılır:

1. Giriş Katmanı
2. Gizli Katman(lar)
3. Çıkış Katmanı

Giriş katmanı giriş verilerini alır. Elinizdeki veri setinde kaç adet girdi(input) varsa onları alan ilk katmandır.

Gizli katmanlar matematiksel hesaplamalar yapar. Yapay sinir ağları oluşturmadaki zorluklardan biri, her bir katman için nöronların sayısının yanı sıra gizli katmanların sayısına da karar vermektir.

Derin Öğrenmedeki "Derin", birden fazla gizli katmana sahip olmayı ifade eder.

Çıktı katmanı, çıktı verilerini döndürür.

Ağırlıklar

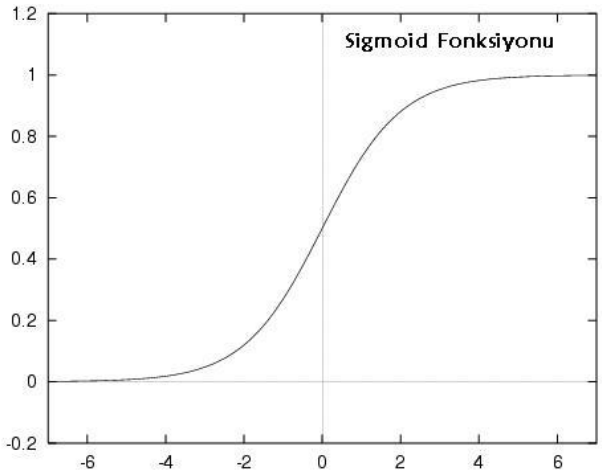
Nöronlar arasındaki her bağlantı bir **ağırlık**(weight) ile ilişkilidir. Bu ağırlık, girdi değerinin önemini belirler. İlk ağırlıklar rastgele ayarlanır. Bu rastgelelik istatistiksel dağılımlardan seçilir. Örneğin uniform dağılım gibi. Bu ağırlıkların sağlaması gereken 2 koşul vardır.

1. Ağırlıkların ortalamasının sıfıra eşit olması gerekir.
2. Ağırlıkların ne çok küçük değerlere ne de çok büyük değerlere sahip olmaması.

Aktivasyon Fonksiyonu

Yapay sinir ağlarında kullanılan fonksiyon doğrusal bir fonksiyondur. Bu fonksiyon doğrusal olmayan bir aktivasyon fonksiyonundan geçirilir. Bunun temel sebebi gerçek dünyada doğrusallığa değil doğrusal olmayan örneklere daha çok rastlanıyor olmasıdır. Bu nedenle aktivasyon fonksiyonunun varlığı önemlidir.

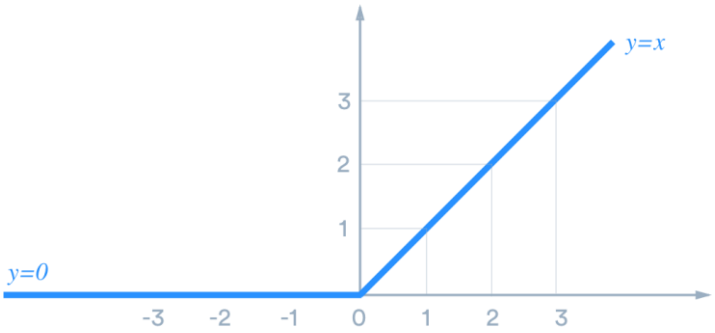
Günümüzde kullanılan birçok aktivasyon fonksiyonu bulunmaktadır. Bunlardan en bilineni sigmoid fonksiyonudur.



Şekil 2 Sigmoid

Sigmoid fonksiyonu yukarıda görüldüğü gibidir. Ağırlıkları belirlerken çok büyük veya çok küçük değerler verilmesini istememizin sebebi grafikte görüldüğü üzere türevlerin çok küçük ve çok büyük değerlerde sıfıra yakınsamasıdır.

Alınan türevlerde 0'a yakınsamayı önlemek adına genelde saklı katmanlarda sigmoid fonksiyonu yerine Relu adını verdiğimiz bir başka aktivasyon fonksiyonu kullanılır.



Şekil 3 Relu

Relu fonksiyonu x ekseninde 0'dan büyük hiçbir değerin türevini 0'a yakınsatmaz. X ekseninde 0 veya 0'dan küçük olan değerlerde eğim yine 0'a eşit olacaktır fakat pratikte bu karşılaşması çok zor bir durumdur.

Relu fonksiyonu saklı katmanlarda çok sık kullanılır fakat çıktı katmanlarında sigmoid fonksiyonu kullanılmaya devam edilir çünkü sigmoid fonksiyonu 0 ile 1 arasında bir sonuç döndürür. Bu bir olasılıktır ve beklediğimiz gözlemin gerçekleşme olasılığını öğrenmiş oluruz.

Kayıp Fonksiyonu (Loss Function)

Her bir tahminin(prediction) gerçek değerden ne kadar uzak olduğunu hesaplayan bir fonksiyondur. Dolayısıyla bu değerin olabildiğince küçük olması istenir ve bütün optimizasyon işlemleri bu fonksiyon üzerinden yapılır çünkü asıl amaç türevler aracılığı ile bu değeri olabildiğince küçültmektir.

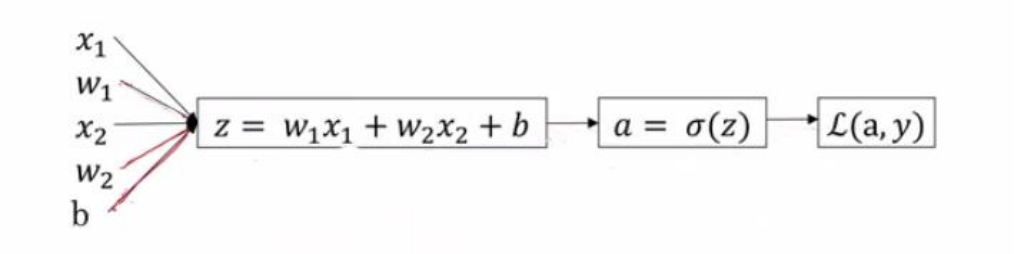
$$L(\hat{y}, y) = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$$

Denklem 1 Kayıp Fonksiyonu

Yukarıdaki fonksiyon en bilinen kayıp fonksiyonudur (Log Loss). Fonksiyondan anlaşılacağı üzere $\hat{y}$ ne kadar büyürse kayıp o kadar küçülür, aynı şekilde $\hat{y}$ ne kadar küçülürse kayıp değeri yine küçülür. Bu da kayıp değerini her durumda küçültebilmeyi sağlar.

Kayıp Fonksiyonunu Nasıl Azaltabiliriz?

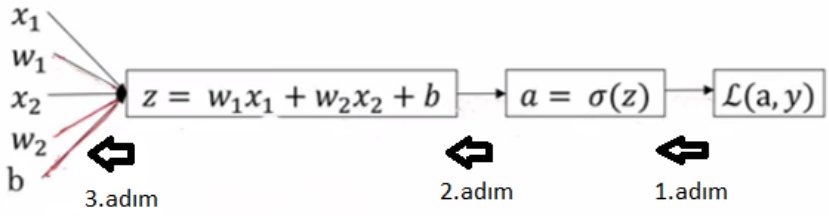
Günümüzde bunun için birçok teknik geliştirilmiştir. Bu tekniklerden ilki ve ilkel olanı Gradyan Düşüşüdür (Gradient Descent). Aşağıda ufak bir sinir ağında olayların nasıl gerçekleştiği verilmiştir.



Şekil 4 İleri Yayılım

Yukarıdaki resimde bir giriş bir de saklı katman olan bir sinir ağının nasıl çalıştığı görselleştirilmiştir.

Burada yapay sinir ağları için kullanılan formül $z = w'x + b$ dir. W ifadesi ağırlıkları temsil eder, kaç tane katman varsa o kadar w matrisi vardır, b ifadesi de yanlılığı(bias) temsil eder. Bunu takip eden ikinci ok sigmoid fonksiyonu kullanılarak yapılan aktivasyon fonksiyonunu gösterir. Bu nedenle eşitliğe 'a' ismi verilmiştir. L ile gösterilen terim Loss değerinin kısaltmasıdır bu da kayıp fonksiyonudur. Yukarıdaki şema ileri yayılımın nasıl yapıldığını göstermektedir. Bu noktadan sonra geri yayılım adını verdiğimiz optimizasyon işlemi başlamaktadır.



Şekil 5 Geri Yayılım

Buradaki adımlar zincir kuralına göre türev alınarak gerçekleştirilir.

$$L(a, y) = -(y \log(a) + (1-y)\log(1-a))$$

$$Z = w^T x + b$$

$$\hat{y} = \sigma(z) = a$$

1.adım

$$\frac{dL(a,y)}{da} = \frac{-y}{a} + \frac{1-y}{1-a} = d_a$$

2.adım

$$\frac{dL(a,y)}{dz} = \frac{dL(a,y)}{da} * \frac{da}{dz} = a - y = d_z$$

3.adım

$$\frac{dL(a,y)}{dw_1} = x_1 * d_z = 'dw_1'$$

$$\frac{dL(a,y)}{dw_2} = x_2 * d_z = 'dw_2'$$

$$\frac{dL(a,y)}{db} = d_z = 'd_b'$$

Yukarıdaki işlemler sonucunda geri yayılım adımı için w(ağırlıklar) ve b(bias) teriminden çıkarılması gereken 'dw1', 'dw2' ve 'db' değerleri bulunmuş oldu ki bu değerler modeli daha optimal hale getirmek için alınan türevlerin sonuçlarıdır.

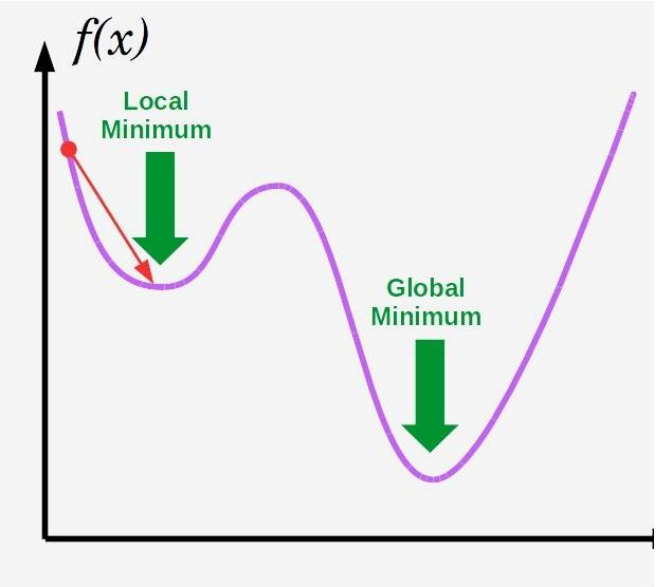
$$w_1 = w_1 - \alpha * dw_1$$

$$w_2 = w_2 - \alpha * dw_2$$

$$b = b - \alpha * d_b$$

Yukarıdaki bu işlemle geri yayılım(backpropagation) işlemi gerçekleşmiş oldu.

Burada kullanılan α(alfa) terimi öğrenme oranı (learning rate) olarak geçer ve derin öğrenmenin en önemli hiper parametrelerinden biri kabul edilir. Bu terim 0 ile 1 arasında değer verilerek hesaplattırılır. Bunun nedeni türev sonucu yaptığımız çıkarma işleminin local minimumlarda takılmasını önleyip global minimuma ulaşmak ve bu inişi yumuşak bir şekilde yapmaktır.



Şekil 6 Lokal Minimum

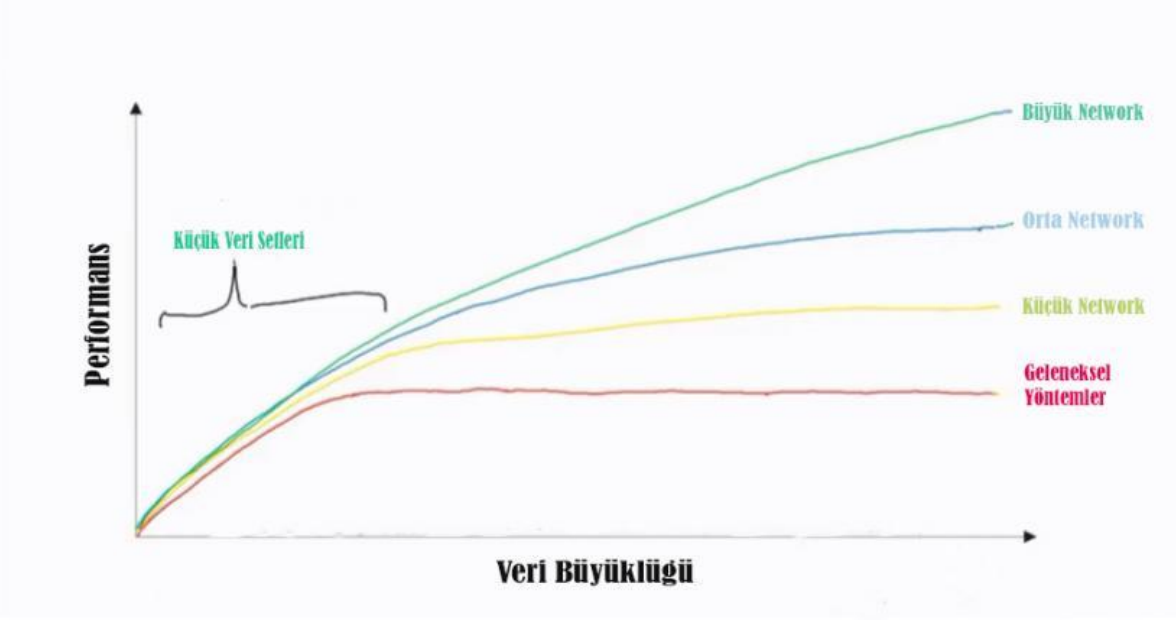
Bu optimizasyon işlemi gradyan iniş (gradient descent) olarak geçer. Eski yöntemlerden birisidir, günümüzde gelişmiş ve daha iyi sonuç veren yöntemler mevcuttur.

Ufak bir sinir ağının çalışma mantığı bu şekildedir. Derin öğrenme, bu ufak sinir ağlarının birleşmesi ve 10'larca belki 100'lerce katman ve nöronların kullanılmasıyla oluşur. Dolayısıyla bu işlemlerin tamamı bir matris işlemine dönüşür. Bu noktada doğrusal cebir devreye girmektedir.

Normalleştirme

Derin öğrenmede normalleştirme çok önemlidir. Normalleştirme sonucu, katmanlardaki değerler ne kadar değişirse değişsin her zaman aynı ortalama ve varyansa sahip olurlar. Bu durumda örneğin 2.katmandaki w ve b değerleri değişirse bu değişim 10.katmandaki değerleri daha az etkiler ve her katman aslında birbirinden bağımsız olur. Dolayısıyla bütün bir ağın öğrenme hızının artmasına sebep olur.

Derin Öğrenmenin Avantajları ve Dezavantajları



Yukarıdaki grafik derin öğrenmenin avantaj ve dezavantajını gösteren en iyi grafiktir.

X ekseni veri büyüklüğünü, y ekseni ise performansı yani modelin ne kadar iyi çalıştığını gösterir.

Görüldüğü üzere küçük veri setlerinde ister derin öğrenme ister geleneksel yöntemler olsun performanslar birbirine yakındır. Veri büyüklüğü artmaya başladıkça derin öğrenme, geleneksel yöntemleri performans olarak geçmeye başlar. Bu noktada derin öğrenme için veri büyüklüğünün çok önemli olduğu söylenebilir. Derin öğrenmenin içinde ise ağ yapısını büyütmeye başladıkça performansta artış olmaya başlar.

Derin öğrenme kullanmak her durumda avantajlı olmaz. Bunun sebebi matris işlemleri kullanılarak yapılan hesaplamalara bilgisayarın işlem gücü yetersiz gelebilir. Bu nedenle gerektiği durumda geleneksel yöntemler kullanılmalıdır.

Derin öğrenmenin en çok kullanıldığı 3 tane alan vardır.

1. Görüntü İşleme
2. Ses İşleme
3. Metin İşleme

Geleneksel yöntemler bu yukarıdaki 3 alan için yetersiz kalmaktadır. Derin öğrenme özellikle görüntü işleme alanında günümüzde birçok yeniliğe sebep olmuştur ve bu yenilikler giderek hızlanmaktadır.

Derin Öğrenme Uygulaması

Veri Setinin Tanımı

Veri seti, Kombine Çevrim Santralinin (CCPP) tam yük ile çalışmaya ayarlandığı ve 6 yıl boyunca toplanan (2006-2011) 9568 veri noktasını içermektedir. Özellikler, tesisin saatlik net enerji çıkışını (PE) tahmin etmek için ortam değişkenlerinin saatlik ortalama sıcaklığı(AT), Ortam Basıncı (AP), Bağıl Nem (RH) ve Egzoz Vakumundan (V) oluşmaktadır. Kombine çevrim enerji santrali (CCPP), gaz türbinleri (GT), buhar türbinleri (ST) ve ısı geri kazanım buhar jeneratörlerinden oluşur. Bir CCPP'de elektrik, bir döngüde birleştirilen ve bir türbinden diğerine aktarılan gaz ve buhar türbinleri tarafından üretilir. Vakum, buhar türbininden toplanır ve üzerinde etkili olurken, ortam değişkenlerinin diğer üçü GT performansını etkiler.

Uygulama

```
library(readxl)
veriseti <- read_excel("veriseti.xlsx")

head(veriseti)
```

```
 A tibble: 6 x 5
     AT     V    AP     RH     PE
   1 15.0   41.8 1024.  73.2  463.
   2 25.2   63.0 1020.  59.1  444.
   3  5.11  39.4 1012.  92.1  489.
   4 20.9   57.3 1010.  76.6  446.
   5 10.8   37.5 1009.  96.6  474.
   6 26.3   59.4 1012.  58.8  444.
```

Yukarıda görüldüğü üzere veri seti 5 farklı değişkenden oluşmaktadır. Burada ilk 4 değişken bağımsız, son değişken bağımlı değişkendir.

```
dim(veriseti)
```

```
[1] 9568    5
```

Veri setinin 9568 tane gözlemi vardır.

Her bir gözlem değerini normalleştirmek gerekmektedir. Bunun sebebi derin öğrenmenin değişkenler arasındaki sayısal uçurumlara çok duyarlı olmasıdır. Bu yüzden derin öğrenme algoritmalarında ölçeklendirme çok önemlidir.

```
scale <- function(x) {
  num <- x - min(x)
  denom <- max(x) - min(x)
  return (num/denom)
}

veriseti <- scale(veriseti)
```

Burada ölçeklendirme yöntemi olarak Min-Max uygulanmıştır.

Daha sonra eldeki veri setinin 4 değişkeni ve 1 çıktı değişkeni birbirinden ayrılmış ve bunlar %90, %10 olacak şekilde bölünmüştür.

```
ind <- sample(2, nrow(veriseti), replace=TRUE, prob=c(0.90, 0.10))

veriseti_training <- as.matrix(veriseti[ind == 1, 1:4])

veriseti_test <- as.matrix(veriseti[ind==2, 1:4])

veriseti_trainingtarget <- as.matrix(veriseti[ind == 1, 5])

veriseti_testtarget <- as.matrix(veriseti[ind==2, 5])
```

Bunun yapılmasındaki sebep eğitilmiş modeli daha önce görmediği test kümelerinde uygulayarak, modelin ne kadar iyi çalıştığını görmektir. Bundan sonraki adım modeli oluşturmaktır.

```
build_model <- function() {

  model <- keras_model_sequential() %>%
    layer_dense(units = 1500,kernel_initializer="glorot_normal" ,
                activation = "relu",
                input_shape = 4) %>%

    layer_dense(units = 1500,kernel_initializer="glorot_normal" ,
                activation = "relu") %>%

    layer_dense(units = 1)

  model %>% compile(
    loss = "mse",
    optimizer = optimizer_adam(),
    metrics = list("mean_absolute_error")
  )

  model
}

model <- build_model()
summary(model)
```

Kullanılan kütüphane Google'ın bir çalışanı tarafından tasarlanan Keras kütüphanesidir. Bu kütüphane arka planında Tensorflow denen başka bir kütüphaneyi kullanmaktadır. Bu kütüphaneyi ise Google şirketinin kendisi tasarlamıştır.

İlk parametre olarak 1500 sayısını girerek nöron sayısı girildi.

İkinci parametre kernel_initializer başlangıçta hangi istatistiksel dağılımı kullanarak nöronlara ağırlıklar verileceğinin belirlendiği kısımdır. Ağırlık verirken dikkat edilmesi gereken 2 tane husus vardır.

1. Her bir optimizasyon gerçekleştiğinde ağırlıkların ortalamasının sıfıra eşit olması gerekir.

2. Ağırlıkların kesinlikle sıfıra eşit olmaması ve ne çok küçük değerler ne de çok büyük değerlere eşit olmasıdır.

Bütün bu hususların ışığında en çok kullanılan 2 tane initializer Xavier ve He'dir. Bu çalışmada Xavier(glorot_normal) kullanılmıştır.

Diğer bir parametre aktivasyon fonksiyonudur, bu fonksiyon girişteki katsayıyı alıp kendi fonksiyonunda işlem uygulayarak çıktı katsayısı elde etmemizi sağlamaktadır. Daha sonra bu çıktı katsayısı ya direk sonucun kendisi olur ya da bir başka katmanın girdisi olur.

Aktivasyon fonksiyonlarının 2 tane ana özelliği vardır.

1. Derin Öğrenme kullanırken elinizdeki veri setinin karmaşıklığı o veri setinin çok büyük bir ihtimalle doğrusal olmayacağını ön görür. Bu sebeple bu fonksiyonlar doğrusal olmayan fonksiyonlardır.

2. Her bir geri beslemede bu aktivasyon fonksiyonlarının türevi alınarak en optimal sonuca ulaşmak amaçlanır. Bu nedenle kolay türevi alınabilen fonksiyonlar tercih edilir böylece zaten sistemi çok yoran derin öğrenme algoritmalarının iş yükü biraz daha hafifletilmiş olur.

Burada aktivasyon fonksiyonu olarak en popüleri olan **Relu** fonksiyonu kullanılması tercih edilmiştir.

En son parametre olarak da veri kümesinin değişken sayısının verilmesi gerekiyor. Bu bilgi veri kümesindeki girdi sayısı olan 4'tür.

Bundan sonraki aşamada bir optimizasyon fonksiyonu belirlenmesi gerekiyor. Bu fonksiyonun amacı ise her bir geri beslemede nöronların katsayılarını daha optimal hale getirip modelin performansını arttırmaktır. Burada 'Adam' optimizasyon algoritması kullanılmıştır.

Daha sonra yapılan modelin her bir optimizasyon da ne kadar iyi performans gösterdiğini ölçmek için hata kare ortalama hesaplanması istenmiştir.

Bundan sonraki aşamada model çalıştırılıp sonuçlar incelenecektir.

```
model %>% fit(
  veriseti_training,
  veriseti_trainingtarget,
  epochs=10
)
```

```
Train on 6324 samples
Epoch 1/10
6324/6324 [==============================] - 9s 1ms/sample - loss: 0.0023 - mean_absolute_error: 0.0223
Epoch 2/10
6324/6324 [==============================] - 5s 818us/sample - loss: 4.3246e-05 - mean_absolute_error: 0.0052
Epoch 3/10
6324/6324 [==============================] - 5s 863us/sample - loss: 3.2749e-05 - mean_absolute_error: 0.0045
Epoch 4/10
6324/6324 [==============================] - 6s 892us/sample - loss: 3.0108e-05 - mean_absolute_error: 0.0044
Epoch 5/10
6324/6324 [==============================] - 6s 908us/sample - loss: 2.8869e-05 - mean_absolute_error: 0.0042
Epoch 6/10
6324/6324 [==============================] - 6s 912us/sample - loss: 2.7685e-05 - mean_absolute_error: 0.0041
Epoch 7/10
6324/6324 [==============================] - 6s 897us/sample - loss: 3.8589e-05 - mean_absolute_error: 0.0049
Epoch 8/10
6324/6324 [==============================] - 6s 888us/sample - loss: 3.5203e-05 - mean_absolute_error: 0.0047
Epoch 9/10
6324/6324 [==============================] - 6s 871us/sample - loss: 2.9853e-05 - mean_absolute_error: 0.0043
Epoch 10/10
6324/6324 [==============================] - 5s 854us/sample - loss: 3.2255e-05 - mean_absolute_error: 0.0045
```

Model fit fonksiyonu ile çağrılmış ve 10 kere optimizasyon yapılması söylenmiştir. Bu noktada kayıp(loss) değerinin her bir adımda düşmüş olması beklenir. Normal şartlarda 10 kere optimizasyon işlemi yeterli olmayabilir. Bu noktadan sonra oluşturulmuş olan model üzerinden test seti ile tahmin yapılması gerekmektedir.

```
test_predictions <- model %>% predict(veriseti_test)
test_predictions[ , 1][0:10]
```

```
 [1] 0.4534746 0.4485265 0.4505319 0.4499769 0.4666794 0.4313463 0.4484209 0.4249985
 [9] 0.4317252 0.4322457
```

Yukarıda daha önce oluşturulan test veri setinin ilk 10 değerinin tahmin sonuçları gözükmektedir.

Bu noktadan sonra istenilen herhangi bir değerin tahmini gerçekleştirilebilir.

Başvurular

Coursera. (tarih yok). *Deep Learning Specialization*. https://www.coursera.org/specializations/deep-learning adresinden alındı

Repository, U. M. (2014). *Combined Cycle Power Plant Data Set*. https://archive.ics.uci.edu/ml/datasets/combined+cycle+power+plant adresinden alındı

Vikipedi. (2019). *Derin Öğrenme*. https://tr.wikipedia.org/wiki/Derin_%C3%B6%C4%9Frenme adresinden alındı

HAZIRLAYANLAR : MERT YANIK – ŞÜKRİYE NUR ŞENCAN