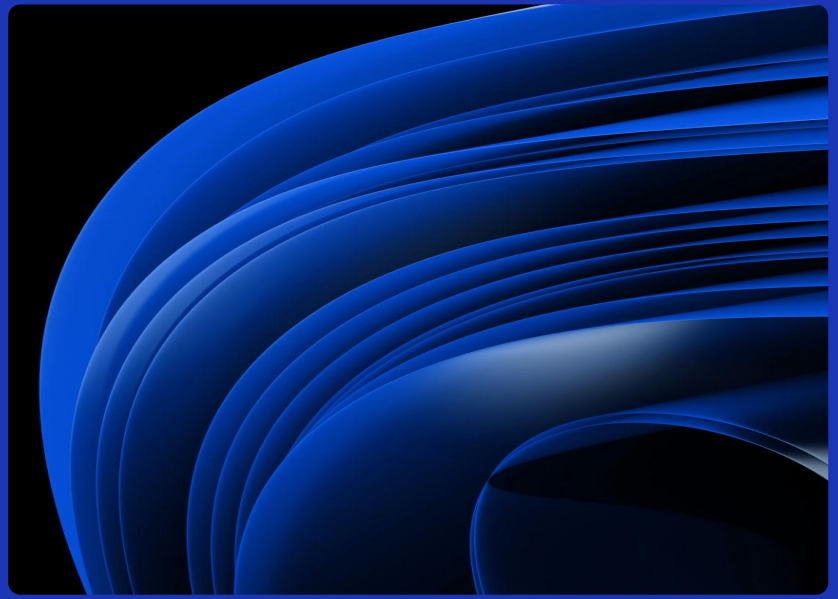
TensorFlow ile Sinir Ağı Regresyonu



TensorFlow ile Sinir Ağı Regresyonu

Bir regresyon problemi, girdilere dayalı olarak sürekli bir sayısal değerin tahmin edilmesini içerir. Amaç, bağımlı ve bağımsız değişkenler arasındaki ilişkiyi bulmaktır; örneğin, ev fiyatlarını boyut ve konuma göre tahmin etmek gibi.

Bu yazıda, girdilerden (verileriniz) oluşan bir örnek almayı, bu girdilerdeki desenleri keşfetmek için bir sinir ağı oluşturmayı ve ardından bu girdilere dayalı bir tahmin (bir sayı şeklinde) yapmayı nasıl gerçekleştirebileceğinize dair temel bilgileri oluşturacağız.

TensorFlow ile Sinir Ağı Regresyonu

Hiperparametre (Hyperparameter)	Tipik değer
Giriş katmanı şekli (Input layer shape)	Özellik sayısıyla aynı şekil (örneğin, konut fiyatı tahmininde yatak odası sayısı, banyo sayısı ve araba park yeri sayısı için 3).
Gizli katman(lar) (Hidden layer(s))	Probleme özgü, minimum = 1, maksimum = sınırsız.
Gizli katman başına nöron sayısı	Probleme özgü, genellikle 10 ile 100 arasında.
Çıkış katmanı şekli (Output layer shape)	İstenilen tahmin şekliyle aynı şekil (örneğin, ev fiyatı için 1).
Gizli katman aktivasyonu (Hidden activation)	Genellikle ReLU (düzeltilmiş doğrusal birim).
Çıkış aktivasyonu	Hiçbiri, ReLU, lojistik/tanh.
Kayıp fonksiyonu	MSE (ortalama kare hata) veya MAE (ortalama mutlak hata)/Huber (MAE/MSE kombinasyonu) eğer aykırı değerler varsa.
Optimizasyon algoritması	SGD (stohastik gradyan inişi), Adam.

Not: Makine öğrenmesinde bir hiperparametre, bir veri analisti veya geliştiricinin kendisinin belirleyebileceği bir şeydir, oysa bir parametre genellikle bir modelin kendi başına öğrendiği bir şeyi tanımlar (analist tarafından açıkça belirlenmeyen bir değer).

İlk adım, TensorFlow modülünü içe aktarmaktır.

TensorFlow'u kullanmak için, onu yaygın takma adı olan tf (TensorFlow'un kısaltması) olarak içe aktaracağız.

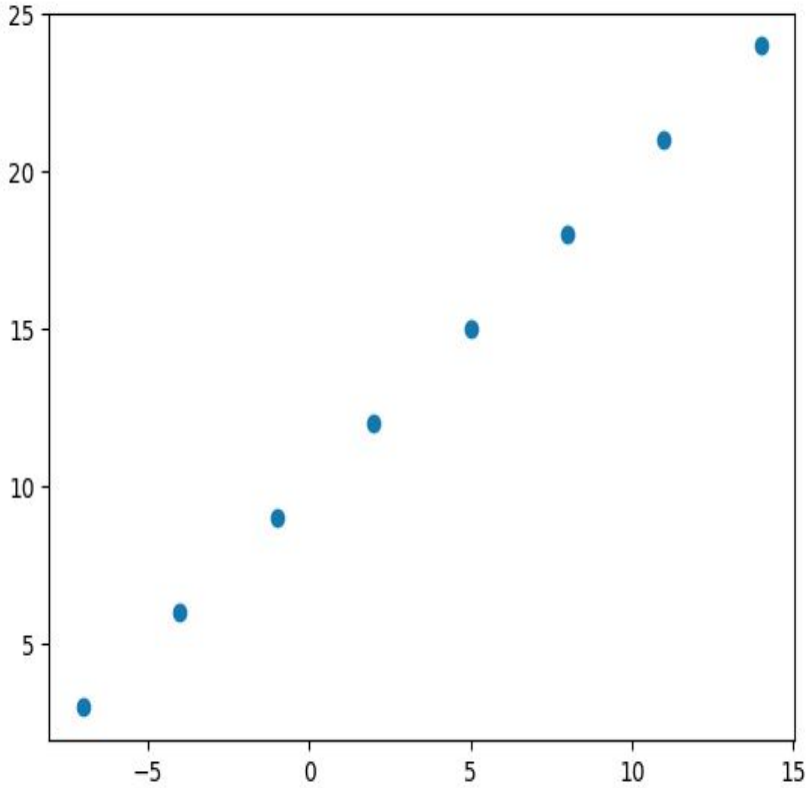
```
import tensorflow as tf

print(tf.__version__) sürümü kontrol et (2.x+ olmalı)

import datetime

print(f"Not defteri son çalıştırma (başlangıçtan sona): {datetime.datetime.now()}")
```

Veriyi görüntülemek ve uyarlamak için oluşturma

Basit bir regresyon problemi için (bir sayıyı tahmin etmek), modellemek amacıyla bazı doğrusal veriler (düz bir çizgi) oluşturalım.

```
import numpy as np

import matplotlib.pyplot as plt

#Özellikler oluşturma

X = np.array([-7.0, -4.0, -1.0, 2.0, 5.0, 8.0, 11.0, 14.0])

#Etiketler oluşturma

y = np.array([3.0, 6.0, 9.0, 12.0, 15.0, 18.0, 21.0, 24.0])

#Görselleştir

plt.scatter(X, y);
```

Veriyi görüntülemek ve uyarlamak için oluşturma



1. Herhangi bir modelleme yapmadan önce, X ile Y arasındaki desenin hesaplanıp hesaplanamayacağını sorabilir misiniz?
2. Örneğin, size bu verilere dayanarak X 17.0 olduğunda y değeri ne olurdu diye sorsam?
3. Ya da X -10.0 olduğunda ne olur?

Bu tür desen keşfi, sinir ağlarını bizim için inşa etmeye çalışacağımız şeyin özüdür.

$$Y = mX + b$$

Regresyon giriş şekilleri ve çıkış şekilleri

Sinir ağlarıyla çalışırken en önemli kavramlardan biri giriş ve çıkış şekilleridir.

Giriş şekli, modelinize giren verinizin şeklidir.

Çıkış şekli, modelinizden çıkmasını istediğiniz verinin şeklidir.

Bunlar, üzerinde çalıştığınız probleme bağlı olarak farklılık gösterecektir.

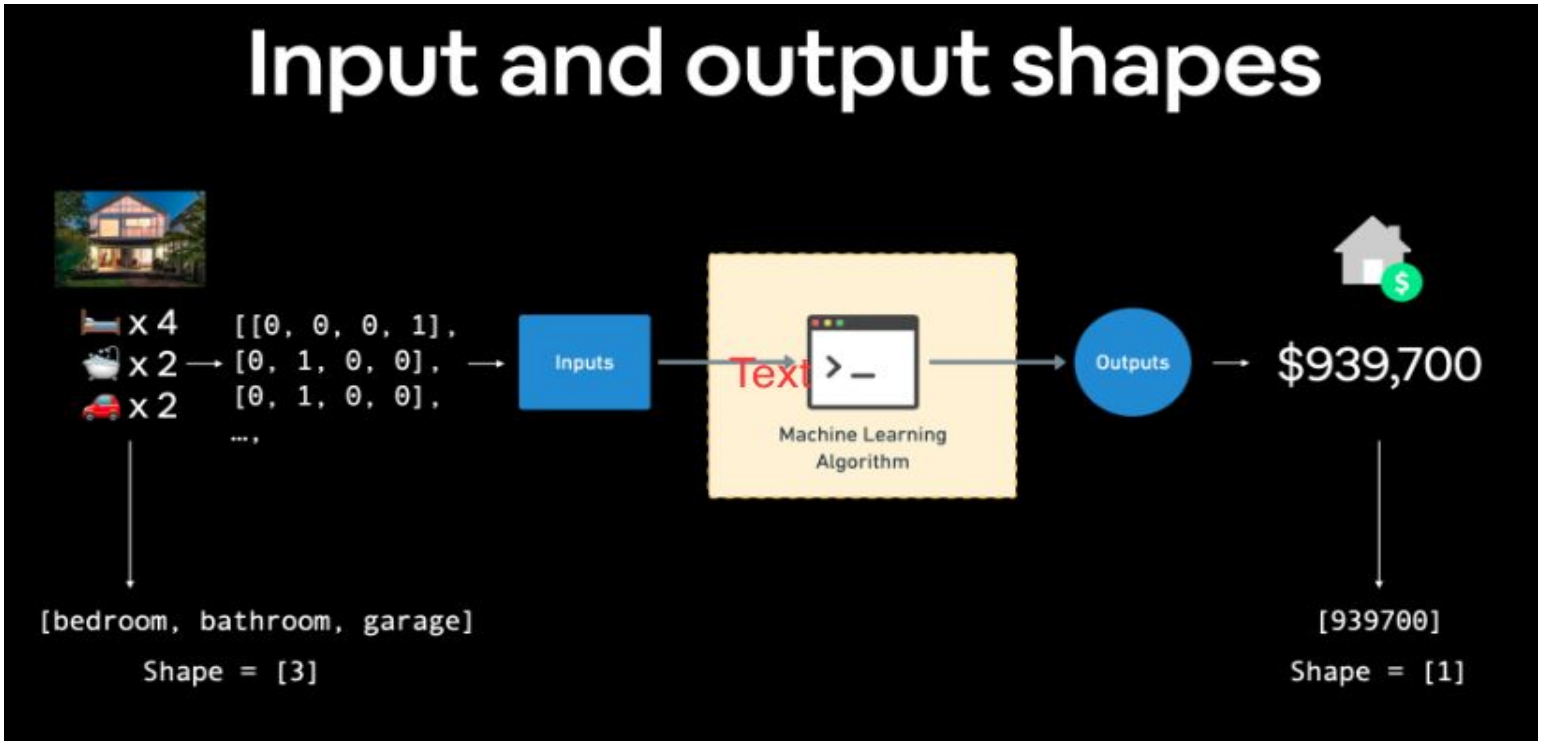
Sinir ağları sayıları kabul eder ve sayılar çıkarır. Bu sayılar genellikle tensörler (veya diziler) olarak temsil edilir.

```
# Bir regresyon modelinin örnek giriş ve çıkış şekilleri

ev_bilgisi = tf.constant(["yatak odası", "banyo", "garaj"])

ev_fiayati = tf.constant([939700])

ev_bilgisi, ev_fiayati

ev_bilgisi.shape
```

Regresyon giriş şekilleri ve çıkış şekilleri

Buradaki amacımız, X'i kullanarak y'yi tahmin etmektir.

Yani, girdiğimiz X olacak ve çıktımız y olacak.



TensorFlow ile modelleme adımları:

TensorFlow'da, bir model oluşturma ve eğitme için genellikle 3 temel adım vardır.

1. **Model oluşturma** - bir sinir ağı katmanlarını kendiniz birleştirin (Fonksiyonel veya Ardışık API kullanarak) veya daha önce yapılmış bir modeli içe aktarın (transfer öğrenme olarak bilinir).
2. **Model derleme** - bir modelin performansının nasıl ölçülmesi gerektiğini (kayıp/metrikler) ve nasıl iyileşmesi gerektiğini (optimizatör) tanımlama.
3. **Modeli eğitme** - modelin verilerdeki desenleri bulmasına izin verme (X'in nasıl y'ye dönüştüğü).

TensorFlow ile modelleme adımları:

```
# Rastgele tohum belirleme (Set random seed)

tf.random.set_seed(42)

# Sequential API kullanarak bir model oluşturma

model = tf.keras.Sequential([

  tf.keras.layers.Dense(1)

])

#Modeli derleme

model.compile(loss=tf.keras.losses.mae,   # mae,ortalama mutlak hata için kısaltmadır

              optimizer=tf.keras.optimizers.SGD(), # SSGD, stohastik gradyan inişi
(stochastic gradient descent) için kısaltmadır

              metrics=["mae"])

#Modeli eğitme

# model.fit(X, y, epochs=5)

model.fit(tf.expand_dims(X, axis=-1), y, epochs=5)
```

```
Epoch 1/5
1/1 [==============================] - 6s 6s/step - loss: 19.2976 - mae: 19.2976
Epoch 2/5
1/1 [==============================] - 0s 18ms/step - loss: 19.0164 - mae: 19.0164
Epoch 3/5
1/1 [==============================] - 0s 12ms/step - loss: 18.7351 - mae: 18.7351
Epoch 4/5
1/1 [==============================] - 0s 13ms/step - loss: 18.4539 - mae: 18.4539
Epoch 5/5
1/1 [==============================] - 0s 13ms/step - loss: 18.1726 - mae: 18.1726
<keras.callbacks.History at 0x7f00663d2680>
```

Yukarıdaki sonuçlar sadece bir örnektir, bu nedenle sizin sonuçlarınız ile yukarıdaki sonuçlar farklı olabilir.

Sadece X ile Y arasındaki desenleri bulması için bir model eğittik.

```
#X ve Y'yi kontrol et

X, y
```

Modelimize 17.0 bir X değeri verirsek, sonuç ne olmalı sizce?

```
# Model ile bir tahmin yap

model.predict([17.0])
```

Bir modeli iyileştirme

Modeli iyileştirmek için, önceki 3 adımın hemen hemen her kısmını değiştiririz.

Model oluşturma - burada daha fazla katman eklemek, her katmandaki gizli birimlerin (aynı zamanda nöronlar olarak da bilinir) sayısını artırmak, her katmanın aktivasyon fonksiyonlarını değiştirmek isteyebilirsiniz.

Model derleme - optimizasyon fonksiyonunu seçmek veya belki de optimizasyon fonksiyonunun öğrenme oranını değiştirmek isteyebilirsiniz.

Modeli eğitme - belki modelinizi daha fazla epoch için eğitebilirsiniz (daha uzun süre eğitim yapmasını sağlamak) veya daha fazla veri üzerinde (modele öğrenmesi için daha fazla örnek sağlamak) eğitebilirsiniz.

Başlamak için, modelimizi uzun süre eğiterek yani epoch sayısını artırarak başlayalım.

Bir modeli iyileştirme

```
# Rastgele tohum belirleme (Set random seed)

tf.random.set_seed(42)

# Sequential API kullanarak bir model oluşturma

model = tf.keras.Sequential([

  tf.keras.layers.Dense(1)

])

#Modeli derleme

model.compile(loss=tf.keras.losses.mae,   # mae,ortalama mutlak hata için kısaltmadır

              optimizer=tf.keras.optimizers.SGD(), # SSGD, stohastik gradyan inişi
(stochastic gradient descent) için kısaltmadır

              metrics=["mae"])

#Modeli eğitme

# model.fit(X, y, epochs=5)

model.fit(tf.expand_dims(X, axis=-1), y, epochs=100) #Modeli 100 epoch için eğitme
```

Bir modeli iyileştirme

```
Epoch 95/100
1/1 [==============================] - 0s 9ms/step - loss: 6.8888 - mae: 6.8888
Epoch 96/100
1/1 [==============================] - 0s 9ms/step - loss: 6.8831 - mae: 6.8831
Epoch 97/100
1/1 [==============================] - 0s 10ms/step - loss: 6.8775 - mae: 6.8775
Epoch 98/100
1/1 [==============================] - 0s 15ms/step - loss: 6.8719 - mae: 6.8719
Epoch 99/100
1/1 [==============================] - 0s 10ms/step - loss: 6.8663 - mae: 6.8663
Epoch 100/100
1/1 [==============================] - 0s 9ms/step - loss: 6.8606 - mae: 6.8606
 <keras.callbacks.History at 0x7f0065eabcd0>
```

```
#X ve y'nin ne olduğunu hatırlayalım

X, y

#X 17.0 olduğunda y'nin ne olacağını tahmin etmeye çalışalım

Sonuc = model.predict(np.array([17.0])) #Doğru cevap 27.0'dır (y = X + 10)
```

Model için daha büyük bir veri kümesi oluşturma

```
#Daha büyük bir veri kümesi oluşturma

X = np.arange(-100, 100, 4)

print(X)

#Veri kümesi için etiketler oluşturma (önceki desenle aynı şekilde)

y = np.arange(-90, 110, 4)

print(y)

#Yukarıdakiyle aynı sonuç

y = X + 10

print(y)
```

Veriyi eğitim/test setlerine ayırma.

Her setin belirli bir amacı vardır:

- Eğitim seti - Model bu veriden öğrenir, genellikle mevcut toplam verinin %70-80'ini oluşturur (dönem boyunca çalıştığınız ders materyalleri gibi).
- Doğrulama seti - Model bu veriyle ayarlanır, genellikle toplam verinin %10-15'ini oluşturur (final sınavından önce yaptığınız deneme sınavı gibi).
- Test seti - Modelin öğrendiklerini test etmek için kullanılır, genellikle toplam verinin %10-15'ini oluşturur (dönemin sonunda girdiğiniz final sınavı gibi).

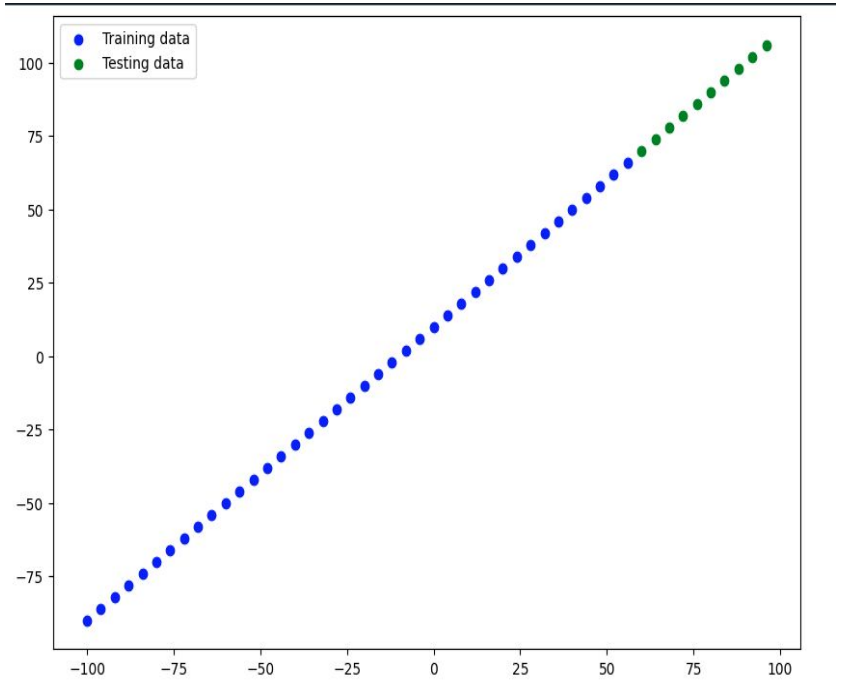
```
#Kaç örneğe sahip olduğumuzu kontrol et

len(X)

#Veriyi eğitim ve test setlerine ayırma

X_train = X[:40] #İlk 40 örnek (verinin %80'i)

y_train = y[:40]

X_test = X[40:] # Son 10 örnek (verinin %20'si)

y_test = y[40:]

len(X_train), len(X_test)
```

Veriyi görselleştirme

Eğitim ve test verilerini oluşturduktan sonra, görselleştirmek iyi bir fikirdir.

İkisini ayırt etmek için renklerle bir grafik çizelim.

```
plt.figure(figsize=(10, 7))

# Eğitim verisini mavi renkte grafiğe dökme

plt.scatter(X_train, y_train, c='b',
label='Eğitim veri')

#Test verisini yeşil renkte grafiğe dökme

plt.scatter(X_test, y_test, c='g',
label='Testing veri')

# Legendi gösterme


plt.legend();
```



MODEL OLUŞTURMA

Veriyi görselleştirdikten sonra, bir sonraki adım modeli oluşturmaktır.

```
# Rastgele tohum belirleme (Set random seed)

tf.random.set_seed(42)

# Sequential API kullanarak bir model oluşturma

model = tf.keras.Sequential([

  tf.keras.layers.Dense(1,input_shape=[1])])

#Modeli derleme

model.compile(loss=tf.keras.losses.mae,   # mae,ortalama mutlak hata için kısaltmadır

              optimizer=tf.keras.optimizers.SGD(), # SSGD, stohastik gradyan inişi
(stochastic gradient descent) için kısaltmadır

              metrics=["mae"])
```

Modelin doğasını görselleştirme

```
model.summary()
```

model.summary() komutunu çağırmak, modelin içerdiği katmanları, çıktı şekillerini ve parametre sayısını gösterir.

Toplam parametreler - modeldeki toplam parametre sayısı.

Eğitilebilir parametreler - bunlar, modelin eğitim sırasında güncelleyebileceği parametreler (desenler).

Eğitilemeyen parametreler - bu parametreler eğitim sırasında güncellenmez (bu, transfer öğrenme sırasında başka modellerden zaten öğrenilen desenler getirildiğinde yaygın bir durumdur).

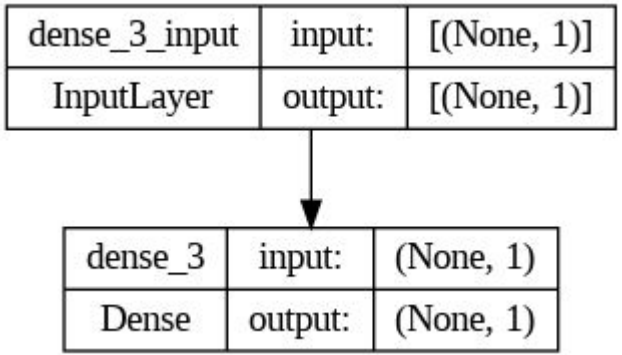
Modelinizi nasıl adlandırdığınıza bağlı olarak, model.summary() komutunun sonuçları aşağıdaki gibi olacaktır.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_3 (Dense)              (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

Modelin doğasını görselleştirme

Özetin yanında, plot_model() kullanarak modelin 2D grafiğini de görüntüleyebilirsiniz.

```
from tensorflow.keras.utils import plot_model

plot_model(model, show_shapes=True)
```



```
# Modeli eğitim verilerine uyarlayın.

model.fit(X_train, y_train, epochs=100, verbose=0) #verbose, çıktının ne kadarının gösterileceğini kontrol eder.
```

Tahminlerin görselleştirilmesi

TTahminleri görselleştirmek için, genellikle bunları gerçek etiketlerle karşılaştırmak iyi bir fikirdir. Genellikle bu, y_test vs. y_pred (gerçek etiketler vs. tahminler) şeklinde görselleştirilir. İlk olarak, test verileri (X_test) üzerinde bazı tahminler yapacağız, unutmayın ki model hiç test verisi görmemiştir.

```
# Tahminler yapın.

y_preds = model.predict(X_test)

# Tahminleri görüntüleyin.

y_preds
```

Hadi, modelin performansını grafiksel olarak görselleştirelim.

```
def plot_predictions(train_data=X_train, train_labels=y_train, test_data=X_test,
test_labels=y_test,predictions=y_preds):

        """

        Eğitim verilerini, test verilerini çizip tahminleri karşılaştırır.

        """

        plt.figure(figsize=(10, 7))

        Eğitim verilerini mavi renkte çiz.

        plt.scatter(train_data, train_labels, c="b", label="Training data")

        #Test verilerini yeşil renkte çiz.

        plt.scatter(test_data, test_labels, c="g", label="Testing data")

        #Tahminleri kırmızı renkte çiz (tahminler test verileri üzerinde yapıldı).

        plt.scatter(test_data, predictions, c="r", label="Predictions")

        # Lejendi göster.

  plt.legend();
```

Hadi, modelin performansını grafiksel olarak görselleştirelim.

```
plot_predictions(train_data=X_train,
                 train_labels=y_train,
                 test_data=X_test,
                 test_labels=y_test,
                 predictions=y_preds)
```

Tahmin yaptıktan sonra modelimizi değerlendirmemiz gerekiyor.

```
model.evaluate(X_test, y_test)
```

Hadi, modelin performansını grafiksel olarak görselleştirelim.

Üzerinde çalıştığınız probleme bağlı olarak, farklı modellerin farklı değerlendirme metrikleri vardır.

Regresyon problemleri için kullanılan iki ana metrik şunlardır:

- Ortalama mutlak hata (Mean Absolute Error - MAE): Tahminlerin her biri arasındaki ortalama fark.
- Ortalama kare hata (Mean Squared Error - MSE): Tahminlerin karelerinin ortalama farkı (daha büyük hataların, küçük hatalardan daha zararlı olduğu durumlarda kullanılır).

Bu değerlerin her biri ne kadar düşükse, o kadar iyidir.

Ayrıca, model.evaluate() yöntemini kullanabilirsiniz; bu yöntem, modelin kayıp değerini ve derleme (compile) adımında ayarlanan metrikleri döndürecektir.

```
# Test veri seti üzerinde modeli değerlendir

model.evaluate(X_test, y_test)
```

Bir modeli geliştirmek için deneyler yapmak:

Değerlendirme metriklerini ve modelinizin yaptığı tahminleri gördükten sonra, muhtemelen onu geliştirmek isteyeceksiniz. Bunu yapmanın birçok farklı yolu vardır, ancak bunlardan üç ana yöntem şunlardır:

1. **Daha fazla veri toplayın** — Modelinizin üzerinde çalışıp öğrenebileceği daha fazla örnek sağlayın (daha fazla desen öğrenme fırsatı).
2. **Modelinizi büyütün** (daha karmaşık bir model kullanın) — Bu, daha fazla katman eklemek veya her bir katmanda daha fazla gizli birim kullanmak şeklinde olabilir.
3. **Daha uzun süre eğitin** — Modelinize, verilerdeki desenleri bulması için daha fazla zaman tanıyın.

Not: Verisetimizi kendimiz oluşturduğumuz için kolayca daha fazla veri üretebiliriz, ancak gerçek dünya verisetleriyle çalışırken bu her zaman mümkün olmayabilir.
Şimdi, modelimizi 2. ve 3. yöntemleri kullanarak nasıl geliştirebileceğimize bakalım.

Bir modeli geliştirmek için deneyler yapmak:

Build `model_1`

```
# Rastgele tohum ayarla
tf.random.set_seed(42)
#Orijinal modeli yeniden oluştur
model_1 = tf.keras.Sequential([
  tf.keras.layers.Dense(1)
])
# Modeli derle
model_1.compile(loss=tf.keras.losses.mae,
                optimizer=tf.keras.optimizers.SGD(),
                metrics=['mae'])
# Modeli eğit
model_1.fit(tf.expand_dims(X_train, axis=-1), y_train, epochs=100)
```

Bir modeli geliştirmek için deneyler yapmak

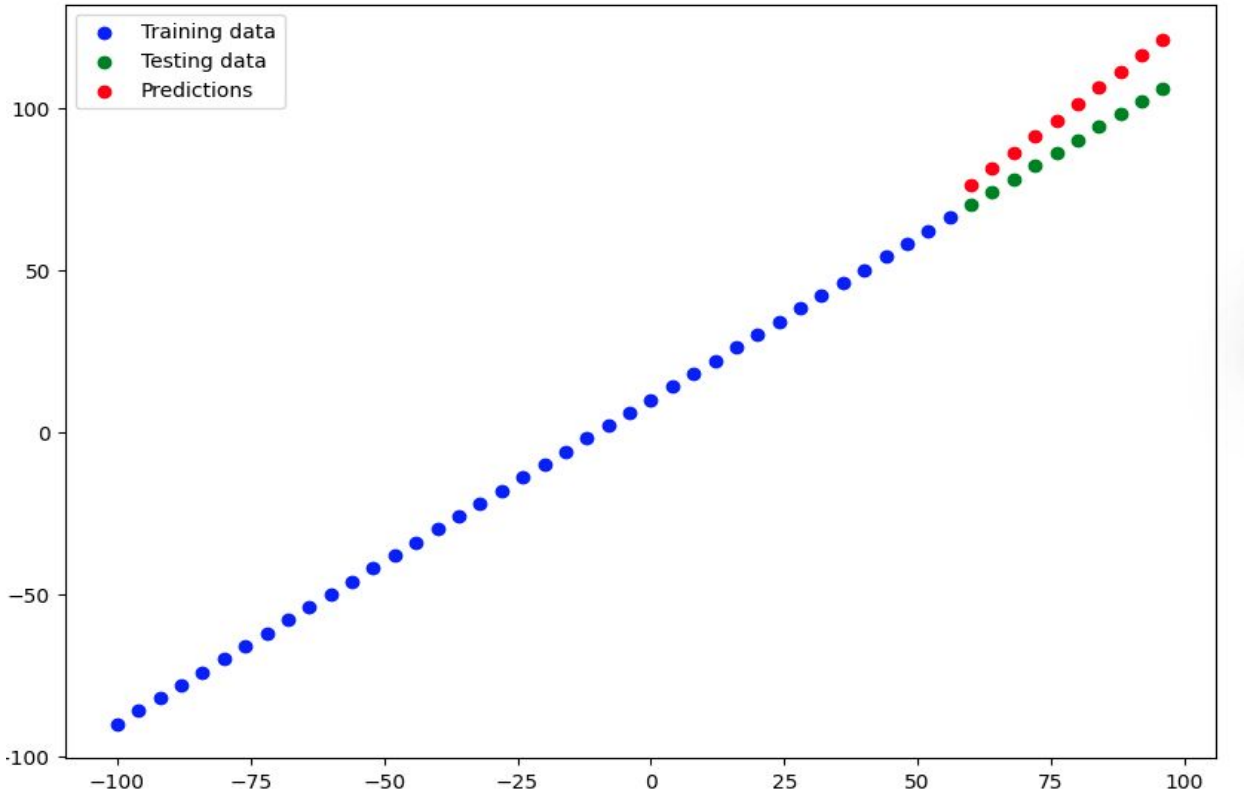
```
# Model_1 için tahminler yapın ve bunları grafikle gösterin.
y_preds_1 = model_1.predict(X_test)
plot_predictions(predictions=y_preds_1)


# Model_1'in metriklerini hesaplayın.
mae_1 = mae(y_test, y_preds_1.squeeze()).numpy()
mse_1 = mse(y_test, y_preds_1.squeeze()).numpy()
mae_1, mse_1
```

Model_2'yi oluşturun.

Bu sefer, her şeyi aynı tutarak ekstra bir yoğun katman ekleyeceğiz (yani modelimiz şimdi 2 katmanlı olacak).

```
tf.random.set_seed(42)

# model_1'i çoğaltın ve ekstra bir katman ekleyin

model_2 = tf.keras.Sequential([ tf.keras.layers.Dense(1),

  tf.keras.layers.Dense(1)]) # İkinci bir katman ekleyin

# Modeli derleyin

model_2.compile(loss=tf.keras.losses.mae,
optimizer=tf.keras.optimizers.SGD(), metrics=['mae'])

# Modeli eğitin

model_2.fit(tf.expand_dims(X_train, axis=-1), y_train, epochs=100,
verbose=2)
```

```
# Model_2 için tahminler yapın ve bunları grafikle gösterin

y_preds_2 = model_2.predict(X_test)

plot_predictions(predictions=y_preds_2)
```



Model_3'yi oluşturun.

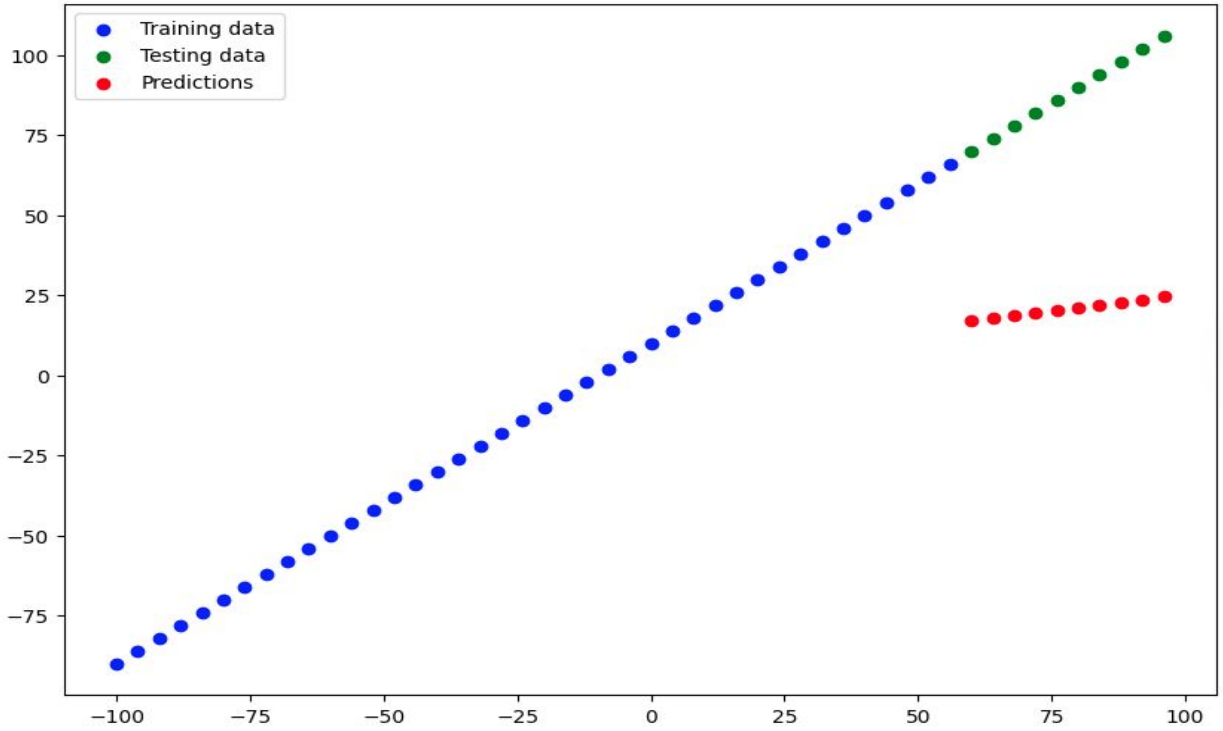
3’ncu modelimiz için, her şeyi model_2 ile aynı tutacağız ancak bu sefer daha uzun süre (100 yerine 500 epoch) eğiteceğiz.

Bu, modelimize verilerdeki desenleri öğrenmesi için daha fazla fırsat tanıyacak.

```
tf.random.set_seed(42)

# model_2'yi çoğaltın

model_3 = tf.keras.Sequential([ tf.keras.layers.Dense(1),
tf.keras.layers.Dense(1)])

# Modeli derleyin

model_3.compile(loss=tf.keras.losses.mae, optimizer=tf.keras.optimizers.SGD(),
metrics=['mae'])


# Modeli eğitin (bu sefer 100 yerine 500 epoch için)

model_3.fit(tf.expand_dims(X_train, axis=-1), y_train, epochs=500, verbose=2)
```

Model_3'yi oluşturun.

```
# Model_3 için tahminler yapın ve bunları grafikle gösterin

y_preds_3 = model_3.predict(X_test)

plot predictions(predictions=y preds 3)
```



Görünüşe göre, modelimiz çok uzun süre eğitilmiş ve bu da daha kötü sonuçlara yol açmış olabilir.

```
# Model_3'ün metriklerini hesaplayın

model_1.evaluate(X_test, y_test)

model_2.evaluate(X_test, y_test)

model_3.evaluate(X_test, y_test)
```

Sonuçları karşılaştırma

Şimdi, 3 benzer ancak biraz farklı sonuçlarımız var, bunları karşılaştıralım.

```
model_results = [["model_1", mae_1, mse_1],
                 ["model_2", mae_2, mse_2],
                 ["model_3", mae_3, mae_3]]

import pandas as pd

all_results = pd.DataFrame(model_results, columns=["model", "mae", "mse"])

all_results
```