```
print(a)
print(b)
print(c)

'Output':
tf.Tensor(1.0, shape=(), dtype=float32)
tf.Tensor(3.0, shape=(), dtype=float32)
tf.Tensor(-4.0, shape=(), dtype=float32)
```

tf.constant() is a Tensor for storing a constant type. Now let’s calculate the roots of the expression.

```
x1 = (-b + tf.math.sqrt(b**2 - (4*a*c))) / 2**a
x2 = (-b - tf.math.sqrt(b**2 - (4*a*c))) / 2**a

roots = (x1, x2)
print(roots)

'Output':
(<tf.Tensor: id=163, shape=(), dtype=float32, numpy=1.0>, <tf.Tensor:
id=175, shape=(), dtype=float32, numpy=-4.0>)
```

TensorFlow 2.0 is eager-first; this implies that operations are executed immediately after they are defined, just like regular python code.

Building Efficient Input Pipelines with the Dataset API

The Dataset API **‘tf.data’** offers an efficient mechanism for building robust input pipelines for passing data into a TensorFlow program. This section uses the Boston housing dataset to illustrate working with the Dataset API methods for building data input pipelines in TensorFlow.

```
# import packages
import tensorflow as tf
from tensorflow.keras.datasets import boston_housing

# load dataset and split in train and test sets
(X_train, y_train), (X_test, y_test) = boston_housing.load_data()
```

```
# construct data input pipelines
dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
dataset = dataset.shuffle(buffer_size=1000)
dataset = dataset.batch(5)

# retrieve first data batch from dataset
for features, labels in dataset:
    print('Features:', features)
    print('Shape of Features:', features.shape)

    print('Labels:', labels)
    print('Shape of Labels:', labels.shape)
    break

'Output':
Features: tf.Tensor(
[[8.19900e-02 0.00000e+00 1.39200e+01 0.00000e+00 4.37000e-01 6.00900e+00
  4.23000e+01 5.50270e+00 4.00000e+00 2.89000e+02 1.60000e+01 3.96900e+02
  1.04000e+01]
 [8.82900e-02 1.25000e+01 7.87000e+00 0.00000e+00 5.24000e-01 6.01200e+00
  6.66000e+01 5.56050e+00 5.00000e+00 3.11000e+02 1.52000e+01 3.95600e+02
  1.24300e+01]
 [2.90900e-01 0.00000e+00 2.18900e+01 0.00000e+00 6.24000e-01 6.17400e+00
  9.36000e+01 1.61190e+00 4.00000e+00 4.37000e+02 2.12000e+01 3.88080e+02
  2.41600e+01]
 [5.87205e+00 0.00000e+00 1.81000e+01 0.00000e+00 6.93000e-01 6.40500e+00
  9.60000e+01 1.67680e+00 2.40000e+01 6.66000e+02 2.02000e+01 3.96900e+02
  1.93700e+01]
 [1.71710e-01 2.50000e+01 5.13000e+00 0.00000e+00 4.53000e-01 5.96600e+00
  9.34000e+01 6.81850e+00 8.00000e+00 2.84000e+02 1.97000e+01 3.78080e+02
  1.44400e+01]], shape=(5, 13), dtype=float64)
Shape of Features: (5, 13)
Labels: tf.Tensor([21.7 22.9 14.  12.5 16. ], shape=(5,), dtype=float64)
Shape of Labels: (5,)
```

From the preceding code listing, take note of the following:

- The method **'tf.data.Dataset.from_tensor_slices()'** is used to create a Dataset whose elements are Tensor slices.
- The Dataset method **'shuffle()'** shuffles the Dataset at each epoch.
- The Dataset method **'batch()'** is used to set the size of each mini-batch of the Dataset. In the preceding example, each Dataset batch contains five observations.

Linear Regression with TensorFlow

In this section, we use TensorFlow to implement a linear regression machine learning model. In the following example, we use the Boston house-prices dataset from the **Keras dataset package** to build a linear regression model with TensorFlow 2.0.

```
# import packages
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import boston_housing
from tensorflow.keras import Model
from sklearn.preprocessing import StandardScaler

# load dataset and split in train and test sets
(X_train, y_train), (X_test, y_test) = boston_housing.load_data()

# standardize the dataset
scaler_X_train = StandardScaler().fit(X_train)
scaler_X_test = StandardScaler().fit(X_test)
X_train = scaler_X_train.transform(X_train)
X_test = scaler_X_test.transform(X_test)

# reshape y-data to become column vector
y_train = np.reshape(y_train, [-1, 1])
y_test = np.reshape(y_test, [-1, 1])

# build the linear model
class LinearRegressionModel(Model):
```