- Observe how the **'weight'** and **'bias'** variables are updated by the gradient descent optimizer within the **'train_step'** method using **'tf.GradientTape()'** to capture and compute the derivatives from the trainable model variables.
- The **'tf.keras.metrics.Accuracy'** method is used to evaluate the accuracy of the model.

Visualizing with TensorBoard

In this section, we will go through visualizing TensorFlow graphs and statistics with TensorBoard. The following code improves on the previous code to build a linear regression model by adding methods to visualize the graph and other variable statistics in TensorBoard using the **'tf.summary'** method calls. The TensorBoard output (illustrated in Figure 30-9) is displayed within the notebook.

```
# import packages
import datetime
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 the TensorBoard notebook extension
%load_ext tensorboard

# 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):
  def __init__(self):
    super(LinearRegressionModel, self).__init__()
    # initialize weight and bias variables
    self.weight = tf.Variable(
        initial_value = tf. random.normal(
            [13, 1], dtype=tf.float64),
        trainable=True)
    self.bias = tf.Variable(initial_value = tf.constant(
        1.0, shape=[], dtype=tf.float64), trainable=True)

  def call(self, inputs):
    return tf.add(tf.matmul(inputs, self.weight), self.bias)

model = LinearRegressionModel()

# parameters
batch_size = 32
learning_rate = 0.01

# use tf.data to batch and shuffle the dataset
train_ds = tf.data.Dataset.from_tensor_slices(
    (X_train, y_train)).shuffle(len(X_train)).batch(batch_size)
test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(batch_size)

loss_object = tf.keras.losses.MeanSquaredError()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

train_loss = tf.keras.metrics.Mean(name='train_loss')
train_rmse = tf.keras.metrics.RootMeanSquaredError(name='train_rmse')

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_rmse = tf.keras.metrics.RootMeanSquaredError(name='test_rmse')
```

```
# use tf.GradientTape to train the model
@tf.function
def train_step(inputs, labels):
  with tf.GradientTape() as tape:
    predictions = model(inputs)
    loss = loss_object(labels, predictions)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))

  train_loss(loss)
  train_rmse(labels, predictions)

@tf.function
def test_step(inputs, labels):
  predictions = model(inputs)
  t_loss = loss_object(labels, predictions)

  test_loss(t_loss)
  test_rmse(labels, predictions)

# Clear any logs from previous runs
!rm -rf ./logs/

# set up summary writers to write the summaries to disk in a different logs
directory
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'logs/gradient_tape/' + current_time + '/train'
test_log_dir = 'logs/gradient_tape/' + current_time + '/test'
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
test_summary_writer = tf.summary.create_file_writer(test_log_dir)

num_epochs = 1000

for epoch in range(num_epochs):
  for train_inputs, train_labels in train_ds:
    train_step(train_inputs, train_labels)
  with train_summary_writer.as_default():
    tf.summary.scalar('loss', train_loss.result(), step=epoch)
    tf.summary.scalar('rmse', train_rmse.result(), step=epoch)
  for test_inputs, test_labels in test_ds:
    test_step(test_inputs, test_labels)
  with test_summary_writer.as_default():
    tf.summary.scalar('loss', test_loss.result(), step=epoch)
    tf.summary.scalar('rmse', test_rmse.result(), step=epoch)

  template = 'Epoch {}, Loss: {}, RMSE: {}, Test Loss: {}, Test RMSE: {}'

  if ((epoch+1) % 100 == 0):
    print (template.format(epoch+1,
                           train_loss.result(),
                           train_rmse.result(),
                           test_loss.result(),
                           test_rmse.result()))

  # Reset metrics every epoch
  train_loss.reset_states()
  test_loss.reset_states()
  train_rmse.reset_states()
  test_rmse.reset_states()
```

```
'Output':
Epoch 100, Loss: 22.03757667541504, RMSE: 4.726028919219971, Test Loss:
29.092111587524414, Test RMSE: 4.577760696411133
Epoch 200, Loss: 21.973844528198242, RMSE: 4.719051837921143, Test Loss:
29.113895416259766, Test RMSE: 4.585252285003662
Epoch 300, Loss: 21.970674514770508, RMSE: 4.7187066078186035, Test Loss:
29.13644790649414, Test RMSE: 4.587917327880859
Epoch 400, Loss: 21.970500946044922, RMSE: 4.718687534332275, Test Loss:
29.1422119140625, Test RMSE: 4.588583469390869
Epoch 500, Loss: 21.970489501953125, RMSE: 4.718685626983643, Test Loss:
29.14352035522461, Test RMSE: 4.588735103607178
Epoch 600, Loss: 21.970487594604492, RMSE: 4.718685626983643, Test Loss:
29.143817901611328, Test RMSE: 4.58876895904541
Epoch 700, Loss: 21.970487594604492, RMSE: 4.718685626983643, Test Loss:
29.143882751464844, Test RMSE: 4.588776111602783
```

```
Epoch 800, Loss: 21.970487594604492, RMSE: 4.718685626983643, Test Loss:
29.14389419555664, Test RMSE: 4.588778018951416
Epoch 900, Loss: 21.970487594604492, RMSE: 4.718685626983643, Test Loss:
29.143898010253906, Test RMSE: 4.588778495788574
Epoch 1000, Loss: 21.970487594604492, RMSE: 4.718685626983643, Test Loss:
29.143898010253906, Test RMSE: 4.588778495788574

# launch tensorboard
%tensorboard --logdir logs/gradient_tape
```

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

- The **'tf.summary.create_file_writer'** method creates summary writers to write the summaries to disk.
- The **'tf.summary.scalar'** method is used to capture scalar metrics for TensorBoard.
- The magic command '%tensorboard' is used to launch TensorBoard by pointing to the appropriate log directory.

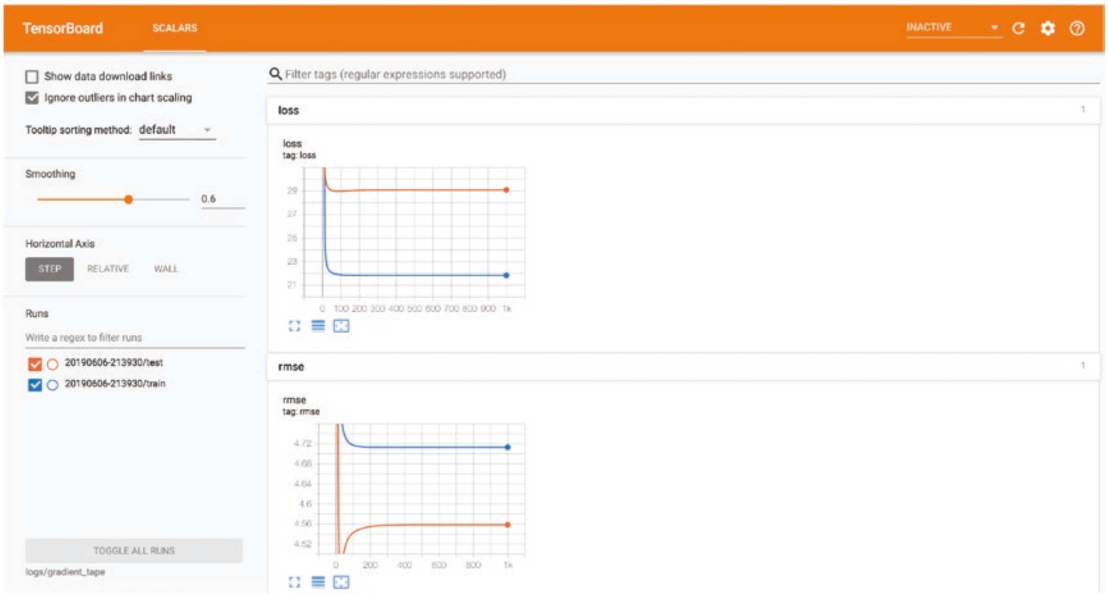

***Figure 30-9.** TensorBoard visualization dashboard for linear regression metrics*

Running TensorFlow with GPUs

GPU is short for graphics processing unit. It is a specialized processor designed for carrying out complex computations on large memory blocks. GPUs provide more efficient processing for building deep learning models.

TensorFlow can leverage processing on multiple GPUs to speed up computation especially when training a complex network architecture. To take advantage of parallel processing, a replica of the network architecture resides on each GPU machine and trains a subset of the data. However, for synchronous updates, the model parameters from each tower (or GPU machines) are stored and updated on a CPU. It turns out that CPUs are generally good at mean or averaging processing. A diagram of this operation is shown in Figure 30-10.