Ungraded Lab: Training a Deep Neural Network with Time Series Data

In this lab, you will build upon the previous exercise and add more dense layers to your network. You will also look at a technique to tune the model's learning rate to make the weights converge faster. This is a useful tip so you can avoid guessing the learning rate before training.

The initial steps will be identical to the previous lab so you can run the next cells until the `Build the Model` section. That's where the discussions begin.

Imports

In [1]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

Utilities

In [2]:
```
def plot_series(time, series, format="-", start=0, end=None):
    """
    Visualizes time series data

    Args:
      time (array of int) - contains the time steps
      series (array of int) - contains the measurements for each time step
      format - line style when plotting the graph
      label - tag for the line
      start - first time step to plot
      end - last time step to plot
    """

    # Setup dimensions of the graph figure
    plt.figure(figsize=(10, 6))

    if type(series) is tuple:

      for series_num in series:
        # Plot the time series data
        plt.plot(time[start:end], series_num[start:end], format)

    else:
      # Plot the time series data
      plt.plot(time[start:end], series[start:end], format)

    # Label the x-axis
    plt.xlabel("Time")

    # Label the y-axis
    plt.ylabel("Value")

    # Overlay a grid on the graph
    plt.grid(True)

    # Draw the graph on screen
    plt.show()

def trend(time, slope=0):
    """
    Generates synthetic data that follows a straight line given a slope value.

    Args:
      time (array of int) - contains the time steps
      slope (float) - determines the direction and steepness of the line

    Returns:
      series (array of float) - measurements that follow a straight line
    """

    # Compute the linear series given the slope
```

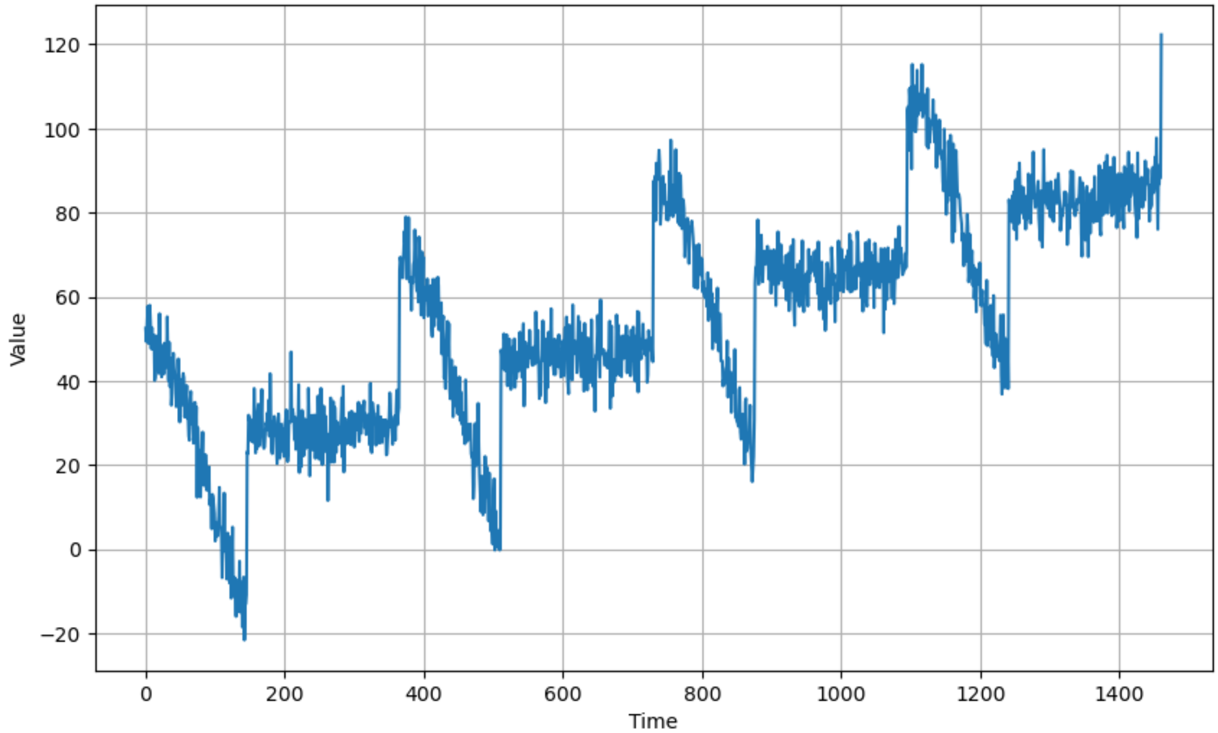
```
    series = slope * time

    return series

def seasonal_pattern(season_time):
    """
    Just an arbitrary pattern, you can change it if you wish

    Args:
      season_time (array of float) - contains the measurements per time step

    Returns:
      data_pattern (array of float) -  contains revised measurement values according
                                  to the defined pattern
    """

    # Generate the values using an arbitrary pattern
    data_pattern = np.where(season_time < 0.4,
                    np.cos(season_time * 2 * np.pi),
                    1 / np.exp(3 * season_time))

    return data_pattern

def seasonality(time, period, amplitude=1, phase=0):
    """
    Repeats the same pattern at each period

    Args:
      time (array of int) - contains the time steps
      period (int) - number of time steps before the pattern repeats
      amplitude (int) - peak measured value in a period
      phase (int) - number of time steps to shift the measured values

    Returns:
      data_pattern (array of float) - seasonal data scaled by the defined amplitude
    """

    # Define the measured values per period
    season_time = ((time + phase) % period) / period

    # Generates the seasonal data scaled by the defined amplitude
    data_pattern = amplitude * seasonal_pattern(season_time)

    return data_pattern

def noise(time, noise_level=1, seed=None):
    """Generates a normally distributed noisy signal

    Args:
      time (array of int) - contains the time steps
      noise_level (float) - scaling factor for the generated signal
      seed (int) - number generator seed for repeatability

    Returns:
      noise (array of float) - the noisy signal
    """

    # Initialize the random number generator
    rnd = np.random.RandomState(seed)

    # Generate a random number for each time step and scale by the noise level
    noise = rnd.randn(len(time)) * noise_level

    return noise
```

Generate the Synthetic Data

In [3]:

```
# Parameters
time = np.arange(4 * 365 + 1, dtype="float32")
baseline = 10
amplitude = 40
slope = 0.05
noise_level = 5

# Create the series
series = baseline + trend(time, slope) + seasonality(time, period=365, amplitude=amplitude)
```

```
# Update with noise
series += noise(time, noise_level, seed=42)

# Plot the results
plot_series(time, series)
```



Split the Dataset

In [4]:
```
# Define the split time
split_time = 1000

# Get the train set
time_train = time[:split_time]
x_train = series[:split_time]

# Get the validation set
time_valid = time[split_time:]
x_valid = series[split_time:]
```

Prepare Features and Labels

In [5]:
```
# Parameters
window_size = 20
batch_size = 32
shuffle_buffer_size = 1000
```

In [6]:
```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    """Generates dataset windows

    Args:
      series (array of float) - contains the values of the time series
      window_size (int) - the number of time steps to average
      batch_size (int) - the batch size
      shuffle_buffer(int) - buffer size to use for the shuffle method

    Returns:
      dataset (TF Dataset) - TF Dataset containing time windows
    """

    # Generate a TF Dataset from the series values
    dataset = tf.data.Dataset.from_tensor_slices(series)
```

```
    # Window the data but only take those with the specified size
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)

    # Flatten the windows by putting its elements in a single batch
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))

    # Create tuples with features and labels
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))

    # Shuffle the windows
    dataset = dataset.shuffle(shuffle_buffer)

    # Create batches of windows
    dataset = dataset.batch(batch_size)

    # Optimize the dataset for training
    dataset = dataset.cache().prefetch(1)

    return dataset
```

In [7]:
```
# Generate the dataset windows
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
```

Build the Model

You will use three dense layers in this exercise as shown below. As expected, the number of trainable parameters will increase and the model summary shows that it is more than tenfold of the previous lab.

In [8]:
```
# Build the model
model_baseline = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1)
])

# Print the model summary
model_baseline.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	210
dense_1 (Dense)	(None, 10)	110
dense_2 (Dense)	(None, 1)	11

Total params: 331 (1.29 KB)
Trainable params: 331 (1.29 KB)
Non-trainable params: 0 (0.00 B)

Train the Model

You will then compile and train the model using the same settings as before. Observe how the loss is decreasing because you will revisit it later in this lab.

In [9]:
```
# Set the training parameters
model_baseline.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning_rate=1e-6, momentum=0.9))
```

In [10]:
```
# Train the model
model_baseline.fit(dataset,epochs=100)
```

```
Epoch 1/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 2351.6575
Epoch 2/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 155.3569
Epoch 3/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 130.4013
Epoch 4/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 122.3813
Epoch 5/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 114.3685
Epoch 6/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - loss: 107.1536
Epoch 7/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 751us/step - loss: 100.5600
Epoch 8/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - loss: 94.3103
Epoch 9/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - loss: 88.5674
Epoch 10/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - loss: 83.2706
Epoch 11/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 777us/step - loss: 79.5736
Epoch 12/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 760us/step - loss: 77.1859
Epoch 13/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 725us/step - loss: 75.4830
Epoch 14/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 771us/step - loss: 74.0826
Epoch 15/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 72.8971
Epoch 16/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 816us/step - loss: 71.8665
Epoch 17/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 70.9468
Epoch 18/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 70.1109
Epoch 19/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 69.3312
Epoch 20/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 68.6132
Epoch 21/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 727us/step - loss: 67.9577
Epoch 22/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 759us/step - loss: 67.3332
Epoch 23/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - loss: 66.7305
Epoch 24/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - loss: 66.1574
Epoch 25/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 748us/step - loss: 65.6074
Epoch 26/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 65.0767
Epoch 27/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 815us/step - loss: 64.5773
Epoch 28/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 737us/step - loss: 64.0883
Epoch 29/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 721us/step - loss: 63.5898
Epoch 30/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 743us/step - loss: 63.0795
Epoch 31/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 749us/step - loss: 62.6389
Epoch 32/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 744us/step - loss: 62.2439
Epoch 33/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 727us/step - loss: 61.8715
Epoch 34/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 762us/step - loss: 61.5256
Epoch 35/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 690us/step - loss: 61.1892
Epoch 36/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 712us/step - loss: 60.8602
Epoch 37/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - loss: 60.5383
Epoch 38/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 853us/step - loss: 60.2251
Epoch 39/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 847us/step - loss: 59.9147
```

```
Epoch 40/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 59.6127
Epoch 41/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 812us/step - loss: 59.3118
Epoch 42/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step - loss: 59.0277
Epoch 43/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 760us/step - loss: 58.7444
Epoch 44/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 758us/step - loss: 58.4784
Epoch 45/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 760us/step - loss: 58.2529
Epoch 46/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 58.0228
Epoch 47/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 791us/step - loss: 57.7747
Epoch 48/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 754us/step - loss: 57.5495
Epoch 49/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 57.3034
Epoch 50/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 57.0824
Epoch 51/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 736us/step - loss: 56.8500
Epoch 52/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 755us/step - loss: 56.6434
Epoch 53/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 783us/step - loss: 56.4470
Epoch 54/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 56.2290
Epoch 55/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 727us/step - loss: 55.9962
Epoch 56/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 753us/step - loss: 55.7678
Epoch 57/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 715us/step - loss: 55.5608
Epoch 58/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 733us/step - loss: 55.3953
Epoch 59/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 55.2357
Epoch 60/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 55.0553
Epoch 61/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 714us/step - loss: 54.8524
Epoch 62/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 707us/step - loss: 54.6543
Epoch 63/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 716us/step - loss: 54.4846
Epoch 64/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 54.2926
Epoch 65/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - loss: 54.0994
Epoch 66/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 742us/step - loss: 53.9096
Epoch 67/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 693us/step - loss: 53.7129
Epoch 68/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 709us/step - loss: 53.5236
Epoch 69/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 724us/step - loss: 53.3442
Epoch 70/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 745us/step - loss: 53.1781
Epoch 71/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 777us/step - loss: 53.0205
Epoch 72/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 52.8631
Epoch 73/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 52.7307
Epoch 74/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 744us/step - loss: 52.5706
Epoch 75/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 52.4024
Epoch 76/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 52.2363
Epoch 77/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 52.0837
Epoch 78/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 772us/step - loss: 51.9327
```

```
Epoch 79/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 848us/step - loss: 51.7847
Epoch 80/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 51.6317
Epoch 81/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 51.4853
Epoch 82/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - loss: 51.3457
Epoch 83/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 906us/step - loss: 51.2121
Epoch 84/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 837us/step - loss: 51.0810
Epoch 85/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 50.9469
Epoch 86/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - loss: 50.8174
Epoch 87/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - loss: 50.6907
Epoch 88/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 923us/step - loss: 50.5774
Epoch 89/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 893us/step - loss: 50.4746
Epoch 90/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 50.3473
Epoch 91/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - loss: 50.2200
Epoch 92/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 759us/step - loss: 50.0993
Epoch 93/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 49.9868
Epoch 94/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - loss: 49.8653
Epoch 95/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 49.7526
Epoch 96/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - loss: 49.6338
Epoch 97/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step - loss: 49.5286
Epoch 98/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 760us/step - loss: 49.4137
Epoch 99/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 737us/step - loss: 49.3103
Epoch 100/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 781us/step - loss: 49.1988
```

```
Out[10]: <keras.src.callbacks.history.History at 0x77df521c8c90>
```

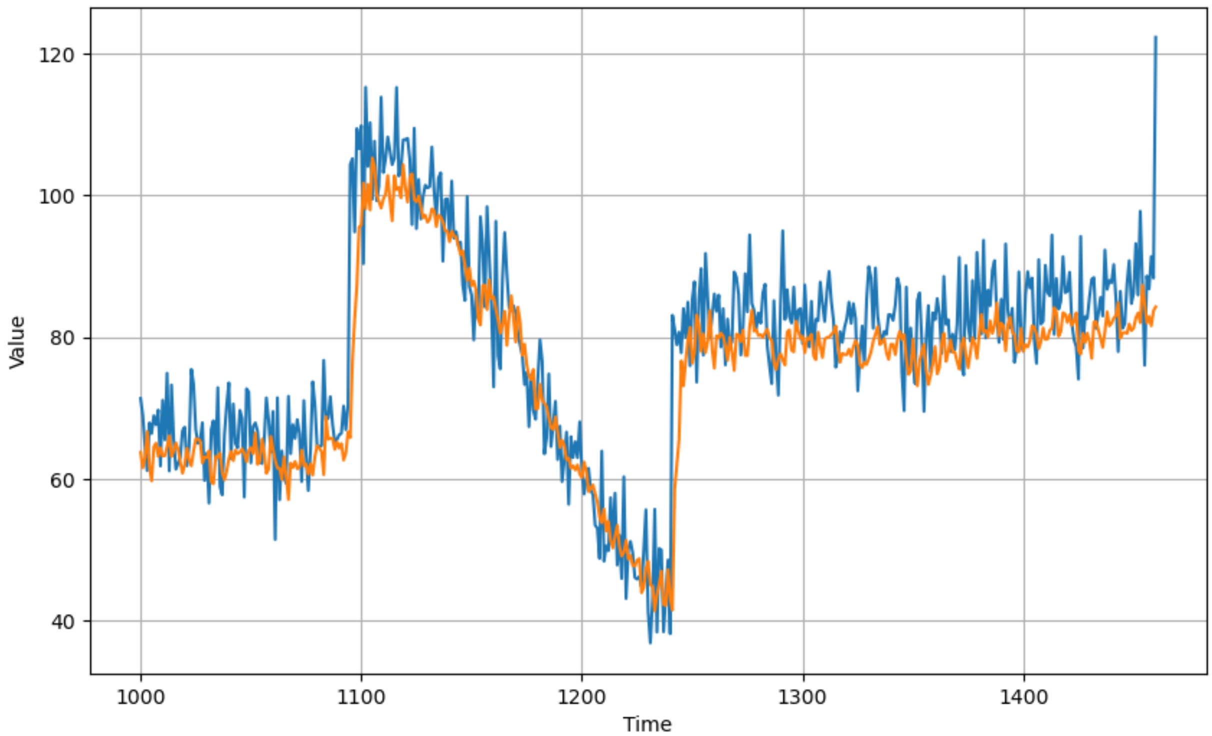
You can then get some predictions and visualize it as before. Since the network is deeper, the predictions might go slower so you may want to minimize unnecessary computations.

In the previous lab, you might remember the model generating predictions using the entire `series` data. That resulted in 1,441 points in the `forecast` list then you sliced the 461 points that aligns with the validation set using `forecast = forecast[split_time - window_size:]`.

You can make this process faster by just generating 461 points right from the start. That way, you don't waste time predicting points that will just be thrown away later. The code below will do just that. It will just get the points needed from the original `series` before calling the `predict()` method. With that, all predictions will align with the validation set already and the for-loop will run for only 461 times instead of 1,441.

In the next lab, you'll see an even faster way to generate these predictions.

In [11]:
```
# Initialize a list
forecast = []

# Reduce the original series
forecast_series = series[split_time - window_size:]

# Use the model to predict data points per window size
for time in range(len(forecast_series) - window_size):
  forecast.append(model_baseline.predict(forecast_series[time:time + window_size][np.newaxis], verbose=0))

# Convert to a numpy array and drop single dimensional axes
results = np.array(forecast).squeeze()

# Plot the results
plot_series(time_valid, (x_valid, results))
```



You can then get the MSE and MAE for reference.

In [12]:
```
# Compute the metrics
print(tf.keras.metrics.mse(x_valid, results).numpy())
print(tf.keras.metrics.mae(x_valid, results).numpy())
```

```
55.582363
5.558003
```

Tune the learning rate

You saw that the training went well with the initial learning rate that you chose (i.e. `1e-6`). However, you're not yet sure if it is the best setting for this particular model. It might seem inconsequential in this simple model but when you have more complex ones, spending some time to tune the learning rate can lead to better training results. You will see how to do that in this section.

First, you will build the same model architecture you just used.

In [13]:
```
# Build the Model
model_tune = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1)
])
```

Next, you will declare a learning rate scheduler callback. This will allow you to dynamically set the learning rate based on the epoch number during training. As shown below, you will pass a lambda function to declare the value of the learning rate. It will start at `1e-8` at epoch 0 and is scaled by `10**(epoch / 20)` as the training goes on.

In [14]:
```
# Set the learning rate scheduler
lr_schedule = tf.keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-8 * 10**(epoch / 20))
```

You will then compile the model. Just to note a subtle difference with the lecture video, you don't have to set the `learning_rate` argument of the optimizer here before compiling. You can just leave the default (i.e. `0.01` for SGD) and allow the learning rate scheduler to set it dynamically.

In [15]:
```
# Initialize the optimizer
optimizer = tf.keras.optimizers.SGD(momentum=0.9)

# Set the training parameters
model_tune.compile(loss="mse", optimizer=optimizer)
```

You will pass in the `lr_schedule` callback in the `callbacks` parameter of the `fit()` method. As you run the training below, you will see the learning rate at a particular epoch denoted by `lr` in the console output. Notice that it is increasing as expected based on the lambda function you used.

In [16]:
```
# Train the model
history = model_tune.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

```
Epoch 1/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 2506.3940 - learning_rate: 1.0000e-08
Epoch 2/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 1711.6510 - learning_rate: 1.1220e-08
Epoch 3/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 880us/step - loss: 1014.9673 - learning_rate: 1.2589e-08
Epoch 4/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 884us/step - loss: 564.4791 - learning_rate: 1.4125e-08
Epoch 5/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 325.7703 - learning_rate: 1.5849e-08
Epoch 6/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 978us/step - loss: 229.4801 - learning_rate: 1.7783e-08
Epoch 7/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 200.7849 - learning_rate: 1.9953e-08
Epoch 8/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 867us/step - loss: 193.7678 - learning_rate: 2.2387e-08
Epoch 9/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 920us/step - loss: 191.5480 - learning_rate: 2.5119e-08
Epoch 10/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 844us/step - loss: 190.1009 - learning_rate: 2.8184e-08
Epoch 11/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 859us/step - loss: 188.6523 - learning_rate: 3.1623e-08
Epoch 12/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 187.0106 - learning_rate: 3.5481e-08
Epoch 13/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 943us/step - loss: 185.1621 - learning_rate: 3.9811e-08
Epoch 14/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 781us/step - loss: 182.9980 - learning_rate: 4.4668e-08
Epoch 15/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 180.4625 - learning_rate: 5.0119e-08
Epoch 16/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 841us/step - loss: 177.4736 - learning_rate: 5.6234e-08
Epoch 17/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 869us/step - loss: 174.0313 - learning_rate: 6.3096e-08
Epoch 18/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 791us/step - loss: 170.1685 - learning_rate: 7.0795e-08
Epoch 19/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 165.6645 - learning_rate: 7.9433e-08
Epoch 20/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 810us/step - loss: 160.8548 - learning_rate: 8.9125e-08
Epoch 21/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 156.3339 - learning_rate: 1.0000e-07
Epoch 22/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 741us/step - loss: 152.3166 - learning_rate: 1.1220e-07
Epoch 23/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 763us/step - loss: 148.4946 - learning_rate: 1.2589e-07
Epoch 24/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 144.7556 - learning_rate: 1.4125e-07
Epoch 25/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 791us/step - loss: 141.2598 - learning_rate: 1.5849e-07
Epoch 26/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 765us/step - loss: 137.9148 - learning_rate: 1.7783e-07
Epoch 27/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - loss: 134.7648 - learning_rate: 1.9953e-07
Epoch 28/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 835us/step - loss: 131.9390 - learning_rate: 2.2387e-07
Epoch 29/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 129.3246 - learning_rate: 2.5119e-07
Epoch 30/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 126.7800 - learning_rate: 2.8184e-07
Epoch 31/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 839us/step - loss: 124.2571 - learning_rate: 3.1623e-07
Epoch 32/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 121.7981 - learning_rate: 3.5481e-07
Epoch 33/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 806us/step - loss: 119.4434 - learning_rate: 3.9811e-07
Epoch 34/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 117.0431 - learning_rate: 4.4668e-07
Epoch 35/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - loss: 114.6313 - learning_rate: 5.0119e-07
Epoch 36/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 848us/step - loss: 112.0312 - learning_rate: 5.6234e-07
Epoch 37/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 907us/step - loss: 109.0710 - learning_rate: 6.3096e-07
Epoch 38/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 105.5530 - learning_rate: 7.0795e-07
Epoch 39/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 760us/step - loss: 101.8160 - learning_rate: 7.9433e-07
```

```
Epoch 40/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 98.2788 - learning_rate: 8.9125e-07
Epoch 41/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 95.0784 - learning_rate: 1.0000e-06
Epoch 42/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 816us/step - loss: 92.1468 - learning_rate: 1.1220e-06
Epoch 43/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - loss: 89.4083 - learning_rate: 1.2589e-06
Epoch 44/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step - loss: 86.8263 - learning_rate: 1.4125e-06
Epoch 45/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 783us/step - loss: 84.3977 - learning_rate: 1.5849e-06
Epoch 46/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - loss: 81.9957 - learning_rate: 1.7783e-06
Epoch 47/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 917us/step - loss: 79.6247 - learning_rate: 1.9953e-06
Epoch 48/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 76.9839 - learning_rate: 2.2387e-06
Epoch 49/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 74.1391 - learning_rate: 2.5119e-06
Epoch 50/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 882us/step - loss: 71.1085 - learning_rate: 2.8184e-06
Epoch 51/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 862us/step - loss: 69.0980 - learning_rate: 3.1623e-06
Epoch 52/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 67.7431 - learning_rate: 3.5481e-06
Epoch 53/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 867us/step - loss: 66.5331 - learning_rate: 3.9811e-06
Epoch 54/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 879us/step - loss: 64.4961 - learning_rate: 4.4668e-06
Epoch 55/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 857us/step - loss: 61.3742 - learning_rate: 5.0119e-06
Epoch 56/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 884us/step - loss: 58.1354 - learning_rate: 5.6234e-06
Epoch 57/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 55.7164 - learning_rate: 6.3096e-06
Epoch 58/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 848us/step - loss: 54.2430 - learning_rate: 7.0795e-06
Epoch 59/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - loss: 52.9646 - learning_rate: 7.9433e-06
Epoch 60/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 51.3713 - learning_rate: 8.9125e-06
Epoch 61/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 890us/step - loss: 49.6853 - learning_rate: 1.0000e-05
Epoch 62/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 48.4218 - learning_rate: 1.1220e-05
Epoch 63/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 860us/step - loss: 47.3145 - learning_rate: 1.2589e-05
Epoch 64/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 891us/step - loss: 46.8813 - learning_rate: 1.4125e-05
Epoch 65/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 889us/step - loss: 46.7744 - learning_rate: 1.5849e-05
Epoch 66/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - loss: 47.7086 - learning_rate: 1.7783e-05
Epoch 67/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - loss: 48.3448 - learning_rate: 1.9953e-05
Epoch 68/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 871us/step - loss: 48.1848 - learning_rate: 2.2387e-05
Epoch 69/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - loss: 48.0257 - learning_rate: 2.5119e-05
Epoch 70/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 877us/step - loss: 47.8003 - learning_rate: 2.8184e-05
Epoch 71/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 860us/step - loss: 45.3325 - learning_rate: 3.1623e-05
Epoch 72/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 45.9972 - learning_rate: 3.5481e-05
Epoch 73/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 52.0766 - learning_rate: 3.9811e-05
Epoch 74/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - loss: 68.5569 - learning_rate: 4.4668e-05
Epoch 75/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 847us/step - loss: 62.8288 - learning_rate: 5.0119e-05
Epoch 76/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step - loss: 49.9772 - learning_rate: 5.6234e-05
Epoch 77/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 358.0003 - learning_rate: 6.3096e-05
Epoch 78/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 3706.6565 - learning_rate: 7.0795e-05
```

```
Epoch 79/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 773us/step - loss: 2132.3281 - learning_rate: 7.9433e-05
Epoch 80/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 1995.7946 - learning_rate: 8.9125e-05
Epoch 81/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - loss: 1830.6075 - learning_rate: 1.0000e-04
Epoch 82/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 1670.4083 - learning_rate: 1.1220e-04
Epoch 83/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 872us/step - loss: 1511.8896 - learning_rate: 1.2589e-04
Epoch 84/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 843us/step - loss: 1357.6378 - learning_rate: 1.4125e-04
Epoch 85/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 1210.4045 - learning_rate: 1.5849e-04
Epoch 86/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - loss: 1073.0750 - learning_rate: 1.7783e-04
Epoch 87/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 781us/step - loss: 949.6134 - learning_rate: 1.9953e-04
Epoch 88/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 843.4919 - learning_rate: 2.2387e-04
Epoch 89/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 742.2322 - learning_rate: 2.5119e-04
Epoch 90/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 663.6880 - learning_rate: 2.8184e-04
Epoch 91/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 765us/step - loss: 600.6215 - learning_rate: 3.1623e-04
Epoch 92/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - loss: 544.2760 - learning_rate: 3.5481e-04
Epoch 93/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - loss: 516.4283 - learning_rate: 3.9811e-04
Epoch 94/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 491.8518 - learning_rate: 4.4668e-04
Epoch 95/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step - loss: 543.3735 - learning_rate: 5.0119e-04
Epoch 96/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 466.6510 - learning_rate: 5.6234e-04
Epoch 97/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 863us/step - loss: 459.3431 - learning_rate: 6.3096e-04
Epoch 98/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 862us/step - loss: 490.7321 - learning_rate: 7.0795e-04
Epoch 99/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - loss: 456.8626 - learning_rate: 7.9433e-04
Epoch 100/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 862us/step - loss: 456.3004 - learning_rate: 8.9125e-04
```

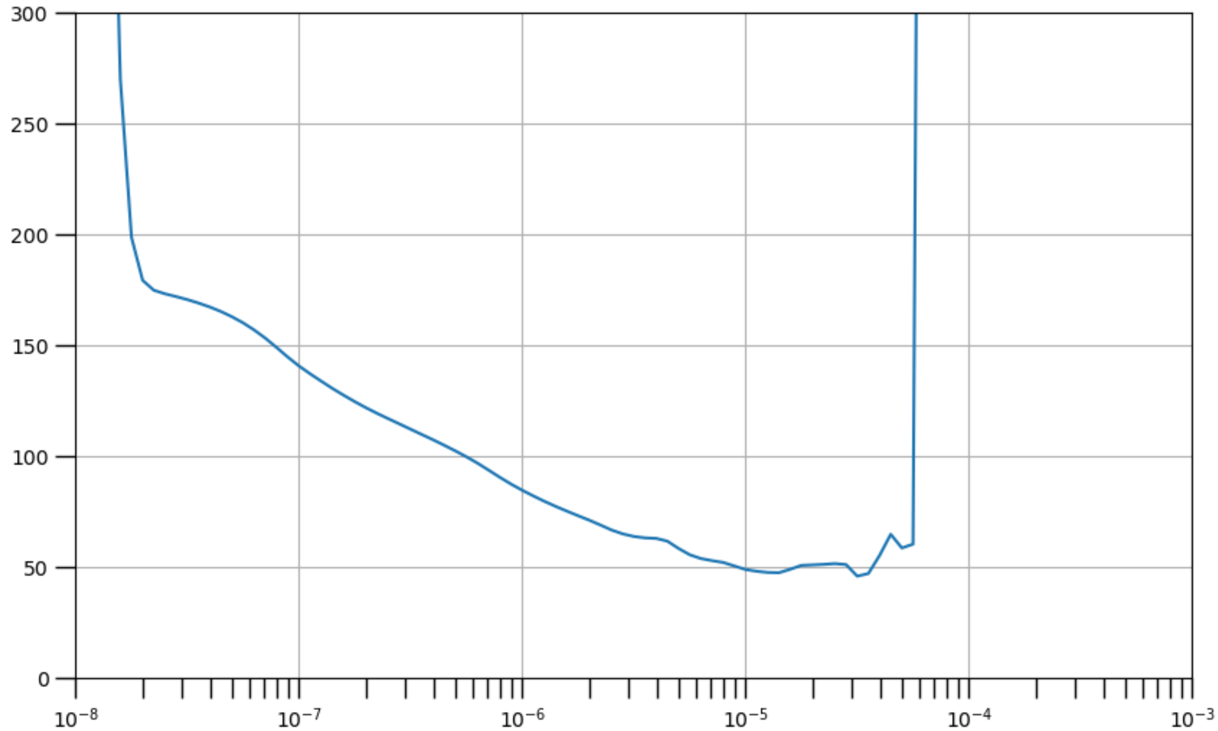
Next step is to plot the results of the training. You will visualize the loss at each value of the learning rate.

In [17]:

```python
# Define the learning rate array
lrs = 1e-8 * (10 ** (np.arange(100) / 20))

# Set the figure size
plt.figure(figsize=(10, 6))

# Set the grid
plt.grid(True)

# Plot the loss in log scale
plt.semilogx(lrs, history.history["loss"])

# Increase the tickmarks size
plt.tick_params('both', length=10, width=1, which='both')

# Set the plot boundaries
plt.axis([1e-8, 1e-3, 0, 300])
```

Out[17]:

```
(1e-08, 0.001, 0.0, 300.0)
```



The generated graph above shows the values of the range of learning rates that leads to lower losses (i.e. sloping downward) and also which ones cause the training to become unstable (i.e. jagged edges and pointing upwards). In general, you will want to pick a point in a downward slope. That means the network is still learning at that point and is stable. Choosing close to the minimum point of the graph will make the training converge to that loss value quicker, as will be shown in the next cells.

First, you will initialize the same model architecture again.

In [18]:
```
# Build the model
model_tune = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1)
])
```

You will then set the optimizer with a learning rate close to the minimum. It is set to `4e-6` initially but feel free to change based on your results.

In [19]:
```
# Set the optimizer with the tuned learning rate
optimizer = tf.keras.optimizers.SGD(learning_rate=4e-6, momentum=0.9)
```

You can then compile and train the model as before. Observe the loss values and compare it to the output of the baseline model you had before. Most likely, you will have met the final loss value of the `model_baseline` within the first 50 epochs of training this `model_tune`. You will also likely have a lower loss after all 100 epochs are done.

In [20]:
```
# Set the training parameters
model_tune.compile(loss="mse", optimizer=optimizer)

# Train the model
history = model_tune.fit(dataset, epochs=100)
```

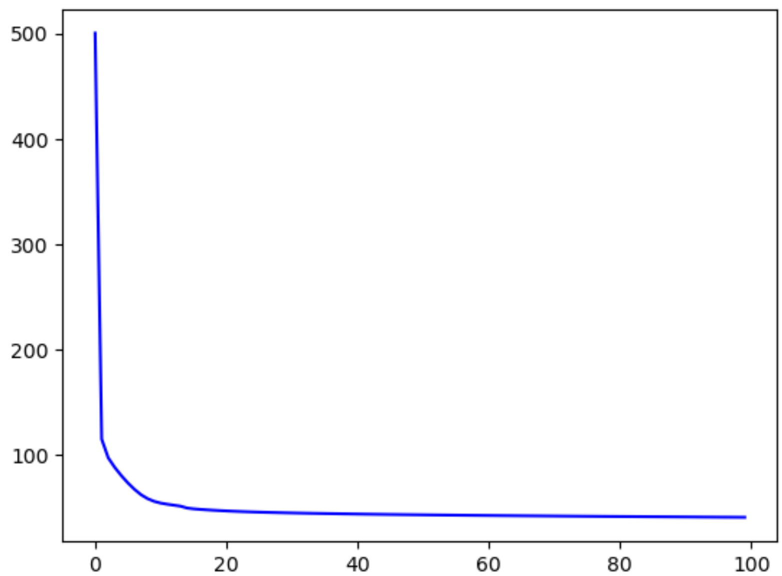
```
Epoch 1/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 898us/step - loss: 975.9979
Epoch 2/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 845us/step - loss: 130.1490
Epoch 3/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 870us/step - loss: 108.0213
Epoch 4/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 746us/step - loss: 98.0994
Epoch 5/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - loss: 89.3026
Epoch 6/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 732us/step - loss: 81.3792
Epoch 7/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 862us/step - loss: 74.3936
Epoch 8/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 730us/step - loss: 68.4266
Epoch 9/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 710us/step - loss: 63.9003
Epoch 10/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 733us/step - loss: 60.6785
Epoch 11/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - loss: 58.5977
Epoch 12/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 765us/step - loss: 57.2478
Epoch 13/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 845us/step - loss: 56.1566
Epoch 14/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 765us/step - loss: 55.1828
Epoch 15/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 763us/step - loss: 53.3245
Epoch 16/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 845us/step - loss: 52.3851
Epoch 17/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 789us/step - loss: 51.6984
Epoch 18/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 798us/step - loss: 51.1562
Epoch 19/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 50.6879
Epoch 20/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 863us/step - loss: 50.2550
Epoch 21/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 745us/step - loss: 49.8760
Epoch 22/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 49.5402
Epoch 23/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 49.2372
Epoch 24/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 48.9542
Epoch 25/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 754us/step - loss: 48.6997
Epoch 26/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 780us/step - loss: 48.4794
Epoch 27/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 48.2743
Epoch 28/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 711us/step - loss: 48.0870
Epoch 29/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - loss: 47.9087
Epoch 30/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 47.7412
Epoch 31/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 880us/step - loss: 47.5815
Epoch 32/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - loss: 47.4306
Epoch 33/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 47.2909
Epoch 34/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 867us/step - loss: 47.1574
Epoch 35/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 812us/step - loss: 47.0299
Epoch 36/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 46.9093
Epoch 37/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 737us/step - loss: 46.7897
Epoch 38/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step - loss: 46.6740
Epoch 39/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - loss: 46.5590
```

```
Epoch 40/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 728us/step - loss: 46.4527
Epoch 41/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 730us/step - loss: 46.3457
Epoch 42/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 46.2488
Epoch 43/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 46.1518
Epoch 44/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 772us/step - loss: 46.0560
Epoch 45/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 45.9627
Epoch 46/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 45.8737
Epoch 47/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 45.7860
Epoch 48/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 863us/step - loss: 45.6988
Epoch 49/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 816us/step - loss: 45.6170
Epoch 50/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 753us/step - loss: 45.5363
Epoch 51/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - loss: 45.4563
Epoch 52/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - loss: 45.3782
Epoch 53/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 758us/step - loss: 45.3044
Epoch 54/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - loss: 45.2271
Epoch 55/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 776us/step - loss: 45.1504
Epoch 56/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 45.0742
Epoch 57/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 806us/step - loss: 44.9999
Epoch 58/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - loss: 44.9253
Epoch 59/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 724us/step - loss: 44.8546
Epoch 60/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 727us/step - loss: 44.7787
Epoch 61/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 771us/step - loss: 44.7110
Epoch 62/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 752us/step - loss: 44.6379
Epoch 63/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 44.5689
Epoch 64/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 726us/step - loss: 44.5044
Epoch 65/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 44.4321
Epoch 66/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 789us/step - loss: 44.3628
Epoch 67/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 859us/step - loss: 44.2934
Epoch 68/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 751us/step - loss: 44.2266
Epoch 69/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 812us/step - loss: 44.1580
Epoch 70/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 868us/step - loss: 44.0849
Epoch 71/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 44.0188
Epoch 72/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 776us/step - loss: 43.9486
Epoch 73/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 747us/step - loss: 43.8778
Epoch 74/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - loss: 43.8061
Epoch 75/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 43.7364
Epoch 76/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 729us/step - loss: 43.6702
Epoch 77/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - loss: 43.6066
Epoch 78/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - loss: 43.5452
```

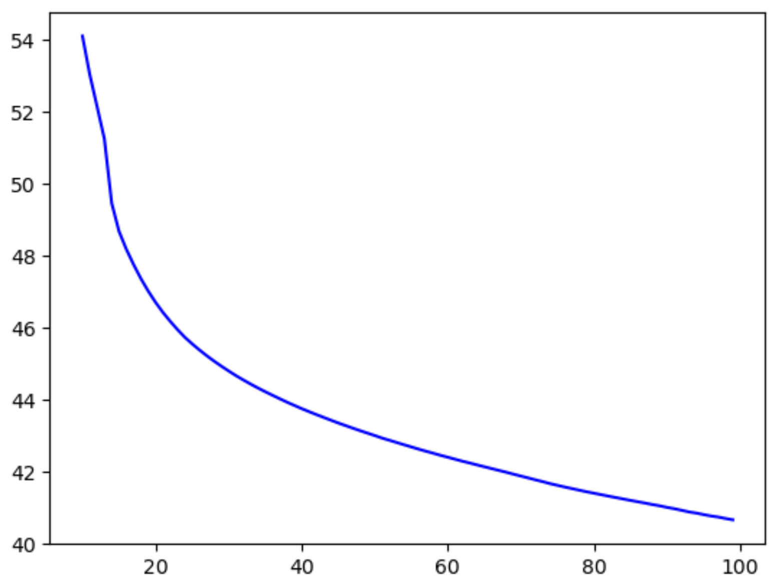
```
Epoch 79/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - loss: 43.4810
Epoch 80/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 846us/step - loss: 43.4217
Epoch 81/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 880us/step - loss: 43.3639
Epoch 82/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 763us/step - loss: 43.3047
Epoch 83/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 764us/step - loss: 43.2465
Epoch 84/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 873us/step - loss: 43.1893
Epoch 85/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 863us/step - loss: 43.1329
Epoch 86/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 782us/step - loss: 43.0766
Epoch 87/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 845us/step - loss: 43.0227
Epoch 88/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 861us/step - loss: 42.9705
Epoch 89/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 732us/step - loss: 42.9146
Epoch 90/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - loss: 42.8554
Epoch 91/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - loss: 42.7959
Epoch 92/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - loss: 42.7379
Epoch 93/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 736us/step - loss: 42.6788
Epoch 94/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 724us/step - loss: 42.6196
Epoch 95/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - loss: 42.5644
Epoch 96/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 746us/step - loss: 42.5096
Epoch 97/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 696us/step - loss: 42.4555
Epoch 98/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 716us/step - loss: 42.4051
Epoch 99/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 720us/step - loss: 42.3556
Epoch 100/100
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 711us/step - loss: 42.3045
```

You can plot the `loss` values by getting it from the History object returned by the `fit()` method. As you can see, the model is still trending downward after the training.

In [21]:
```
# Plot the loss
loss = history.history['loss']
epochs = range(len(loss))
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.show()
```

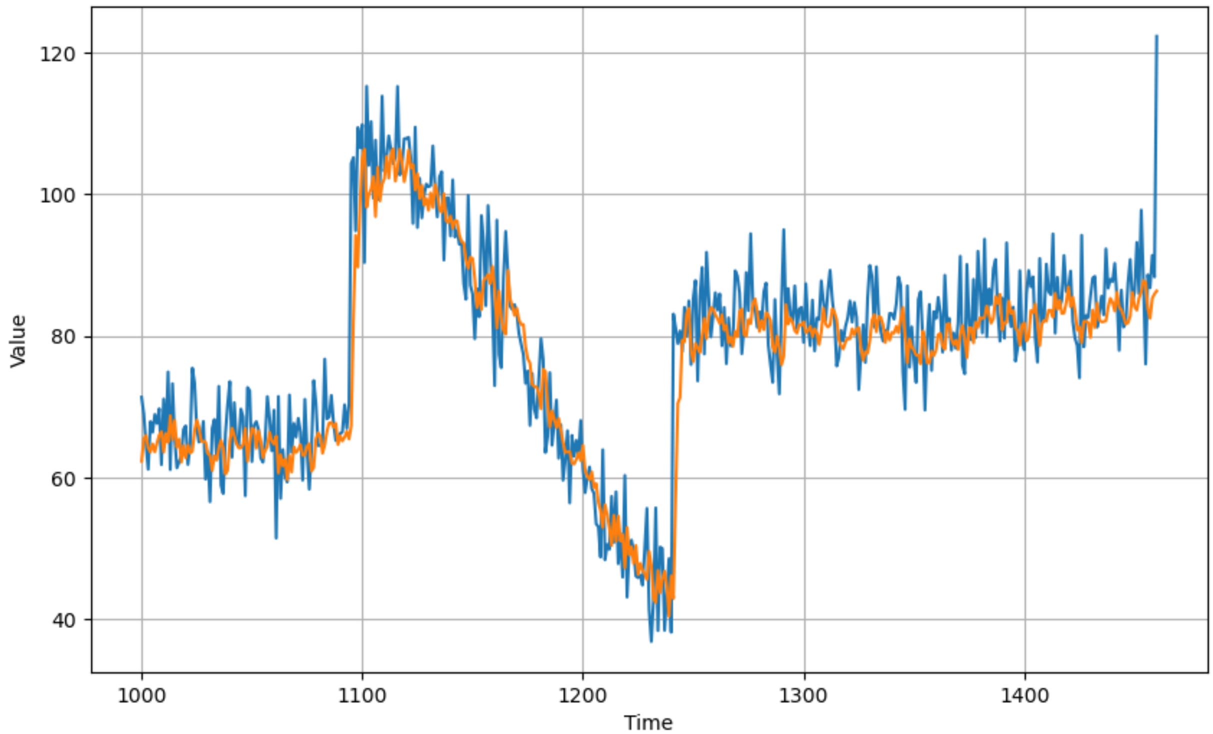


In [22]:
```
# Plot all but the first 10
loss = history.history['loss']
epochs = range(10, len(loss))
plot_loss = loss[10:]
plt.plot(epochs, plot_loss, 'b', label='Training Loss')
plt.show()
```



You can get the predictions again and overlay it on the validation set.

In [23]:
```
# Initialize a list
forecast = []

# Reduce the original series
forecast_series = series[split_time - window_size:]

# Use the model to predict data points per window size
for time in range(len(forecast_series) - window_size):
  forecast.append(model_tune.predict(forecast_series[time:time + window_size][np.newaxis], verbose=0))

# Convert to a numpy array and drop single dimensional axes
results = np.array(forecast).squeeze()

# Plot the results
plot_series(time_valid, (x_valid, results))
```



Finally, you can compute the metrics and you should arrive at similar figures compared to the baseline. If it is much worse, then the model might have overfitted and you can use techniques you know to avoid it (e.g. adding dropout).

In [24]:
```
print(tf.keras.metrics.mse(x_valid, results).numpy())
print(tf.keras.metrics.mae(x_valid, results).numpy())
```

```
45.806854
4.9200997
```

Wrap Up

This concludes the exercise on using a deep neural network for forecasting. Along the way, you did some hyperparameter tuning, particularly on the learning rate. You will be using this technique as well in the next labs. Next week, you will be using recurrent neural networks to build your forecasting model. See you there and keep it up!