Ungraded Lab: Predicting Sunspots with Neural Networks

In the remaining labs for this week, you will move away from synthetic time series and start building models for real world data. In particular, you will train on the Sunspots dataset: a monthly record of sunspot numbers from January 1749 to July 2018. You will first build a deep neural network here composed of dense layers. This will act as your baseline so you can compare it to the next lab where you will use a more complex architecture.

Let's begin!

Imports

You will use the same imports as before with the addition of the csv module. You will need this to parse the CSV file containing the dataset.

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

Utilities

You will only have the `plot_series()` dataset here because you no longer need the synthetic data generation functions.

In [2]:
```
def plot_series(x, y, format="-", start=0, end=None,
                title=None, xlabel=None, ylabel=None, legend=None ):
    """
    Visualizes time series data

    Args:
      x (array of int) - contains values for the x-axis
      y (array of int or tuple of arrays) - contains the values for the y-axis
      format (string) - line style when plotting the graph
      label (string) - tag for the line
      start (int) - first time step to plot
      end (int) - last time step to plot
      title (string) - title of the plot
      xlabel (string) - label for the x-axis
      ylabel (string) - label for the y-axis
      legend (list of strings) - legend for the plot
    """

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

    # Check if there are more than two series to plot
    if type(y) is tuple:

      # Loop over the y elements
      for y_curr in y:

        # Plot the x and current y values
        plt.plot(x[start:end], y_curr[start:end], format)

    else:
      # Plot the x and y values
      plt.plot(x[start:end], y[start:end], format)

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

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

    # Set the legend
    if legend:
      plt.legend(legend)

    # Set the title
    plt.title(title)

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

    # Draw the graph on screen
    plt.show()
```

Download and Preview the Dataset

You can now download the dataset and inspect the contents. The link in class is from Laurence's repo but we also hosted it in the link below.

In [3]:
```
# Download the dataset
!wget -nc https://storage.googleapis.com/tensorflow-1-public/course4/Sunspots.csv
```

```
File ‘Sunspots.csv’ already there; not retrieving.
```

Running the cell below, you'll see that there are only three columns in the dataset:

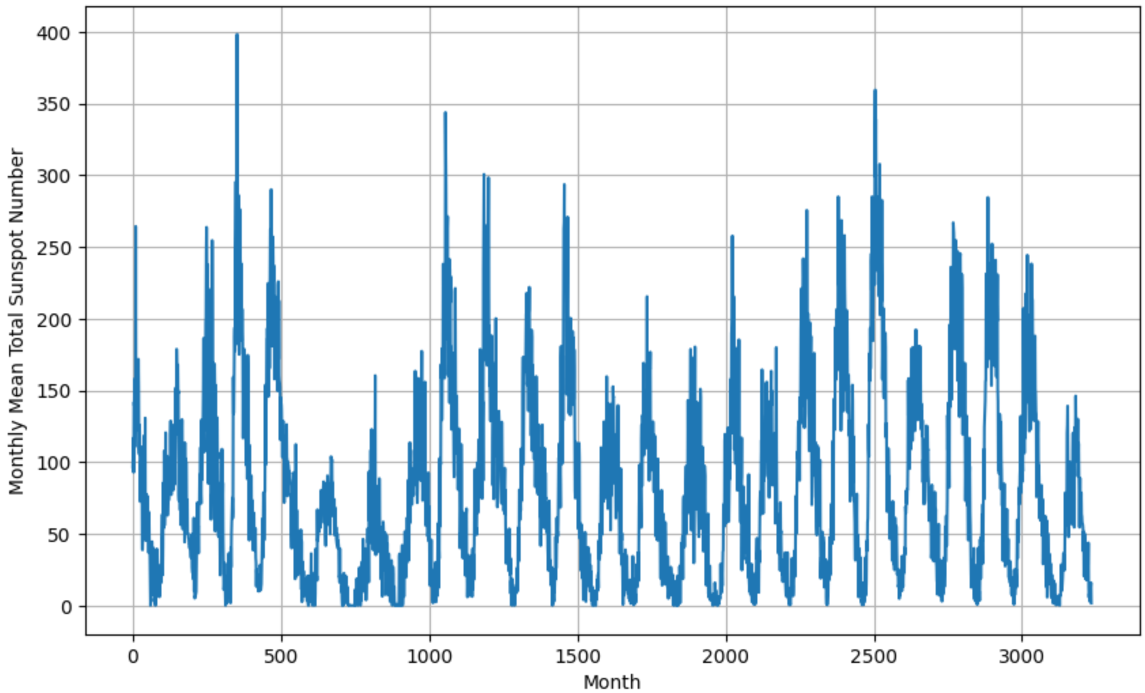
1. untitled column containing the month number
2. Date which has the format `YYYY-MM-DD`
3. Mean Total Sunspot Number

In [4]:
```
# Preview the dataset
!head Sunspots.csv
```

```
,Date,Monthly Mean Total Sunspot Number
0,1749-01-31,96.7
1,1749-02-28,104.3
2,1749-03-31,116.7
3,1749-04-30,92.8
4,1749-05-31,141.7
5,1749-06-30,139.2
6,1749-07-31,158.0
7,1749-08-31,110.5
8,1749-09-30,126.5
```

For this lab and the next, you will only need the month number and the mean total sunspot number. You will load those into memory and convert it to arrays that represents a time series.

In [5]:
```
# Initialize lists
time_step = []
sunspots = []

# Open CSV file
with open('./Sunspots.csv') as csvfile:

  # Initialize reader
  reader = csv.reader(csvfile, delimiter=',')

  # Skip the first line
  next(reader)

  # Append row and sunspot number to lists
  for row in reader:
    time_step.append(int(row[0]))
    sunspots.append(float(row[2]))

# Convert lists to numpy arrays
time = np.array(time_step)
series = np.array(sunspots)

# Preview the data
plot_series(time, series, xlabel='Month', ylabel='Monthly Mean Total Sunspot Number')
```



Split the Dataset

Next, you will split the dataset into training and validation sets. There are 3235 points in the dataset and you will use the first 3000 for training.

In [6]:
```
# Define the split time
split_time = 3000

# 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

You can then prepare the dataset windows as before. The window size is set to 30 points (equal to 2.5 years) but feel free to change later on if you want to experiment.

In [7]:
```
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 include in the feature
      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 [8]:
```
# Parameters
window_size = 30
batch_size = 32
shuffle_buffer_size = 1000

# Generate the dataset windows
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
```

Build the Model

The model will be 3-layer dense network as shown below.

In [9]:
```
# Build the model
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1)
])

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

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 30)	930
dense_1 (Dense)	(None, 10)	310
dense_2 (Dense)	(None, 1)	11

Total params: 1,251 (4.89 KB)
Trainable params: 1,251 (4.89 KB)
Non-trainable params: 0 (0.00 B)

Tune the Learning Rate

You can pick a learning rate by running the same learning rate scheduler code from previous labs.

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

# Initialize the optimizer
optimizer = tf.keras.optimizers.SGD(momentum=0.9)

# Set the training parameters
model.compile(loss=tf.keras.losses.Huber(), optimizer=optimizer)

# Train the model
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

```
Epoch 1/100
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1745446178.871988   30792 service.cc:145] XLA service 0x7c110000a230 initialized for platform CUDA (this does not guara
ntee that XLA will be used). Devices:
I0000 00:00:1745446178.872825   30792 service.cc:153]   StreamExecutor device (0): NVIDIA A10G, Compute Capability 8.6
     51/Unknown 1s 3ms/step - loss: 53.3019
I0000 00:00:1745446179.579299   30792 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the
lifetime of the process.
```

```
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 55.1225 - learning_rate: 1.0000e-08
Epoch 2/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 751us/step - loss: 54.6505 - learning_rate: 1.1220e-08
Epoch 3/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - loss: 54.1112 - learning_rate: 1.2589e-08
Epoch 4/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 735us/step - loss: 53.5123 - learning_rate: 1.4125e-08
Epoch 5/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 724us/step - loss: 52.8496 - learning_rate: 1.5849e-08
Epoch 6/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - loss: 52.1164 - learning_rate: 1.7783e-08
Epoch 7/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 51.3071 - learning_rate: 1.9953e-08
Epoch 8/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 726us/step - loss: 50.4076 - learning_rate: 2.2387e-08
Epoch 9/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 771us/step - loss: 49.4352 - learning_rate: 2.5119e-08
Epoch 10/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 762us/step - loss: 48.3740 - learning_rate: 2.8184e-08
Epoch 11/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 748us/step - loss: 47.2309 - learning_rate: 3.1623e-08
Epoch 12/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 752us/step - loss: 46.0027 - learning_rate: 3.5481e-08
Epoch 13/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 740us/step - loss: 44.6798 - learning_rate: 3.9811e-08
Epoch 14/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 727us/step - loss: 43.2797 - learning_rate: 4.4668e-08
Epoch 15/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 736us/step - loss: 41.8520 - learning_rate: 5.0119e-08
Epoch 16/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - loss: 40.4045 - learning_rate: 5.6234e-08
Epoch 17/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 721us/step - loss: 38.9533 - learning_rate: 6.3096e-08
Epoch 18/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - loss: 37.5629 - learning_rate: 7.0795e-08
Epoch 19/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - loss: 36.2016 - learning_rate: 7.9433e-08
Epoch 20/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 751us/step - loss: 34.9788 - learning_rate: 8.9125e-08
Epoch 21/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 745us/step - loss: 33.9053 - learning_rate: 1.0000e-07
Epoch 22/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 741us/step - loss: 32.9742 - learning_rate: 1.1220e-07
Epoch 23/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 743us/step - loss: 32.2089 - learning_rate: 1.2589e-07
Epoch 24/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 735us/step - loss: 31.5788 - learning_rate: 1.4125e-07
Epoch 25/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - loss: 31.0358 - learning_rate: 1.5849e-07
Epoch 26/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 731us/step - loss: 30.5646 - learning_rate: 1.7783e-07
Epoch 27/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 729us/step - loss: 30.1131 - learning_rate: 1.9953e-07
Epoch 28/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 730us/step - loss: 29.6624 - learning_rate: 2.2387e-07
Epoch 29/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 722us/step - loss: 29.1867 - learning_rate: 2.5119e-07
Epoch 30/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 735us/step - loss: 28.6713 - learning_rate: 2.8184e-07
Epoch 31/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 737us/step - loss: 28.1073 - learning_rate: 3.1623e-07
Epoch 32/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 729us/step - loss: 27.4877 - learning_rate: 3.5481e-07
Epoch 33/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - loss: 26.8408 - learning_rate: 3.9811e-07
Epoch 34/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 735us/step - loss: 26.1755 - learning_rate: 4.4668e-07
Epoch 35/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 752us/step - loss: 25.4903 - learning_rate: 5.0119e-07
Epoch 36/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 720us/step - loss: 24.7965 - learning_rate: 5.6234e-07
Epoch 37/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 24.1260 - learning_rate: 6.3096e-07
Epoch 38/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 737us/step - loss: 23.4919 - learning_rate: 7.0795e-07
Epoch 39/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 729us/step - loss: 22.9322 - learning_rate: 7.9433e-07
Epoch 40/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 735us/step - loss: 22.4234 - learning_rate: 8.9125e-07
Epoch 41/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 713us/step - loss: 21.9676 - learning_rate: 1.0000e-06
Epoch 42/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 719us/step - loss: 21.5839 - learning_rate: 1.1220e-06
```

```
Epoch 43/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 747us/step - loss: 21.2859 - learning_rate: 1.2589e-06
Epoch 44/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 736us/step - loss: 21.0300 - learning_rate: 1.4125e-06
Epoch 45/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 725us/step - loss: 20.8061 - learning_rate: 1.5849e-06
Epoch 46/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 719us/step - loss: 20.5994 - learning_rate: 1.7783e-06
Epoch 47/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 738us/step - loss: 20.4076 - learning_rate: 1.9953e-06
Epoch 48/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 728us/step - loss: 20.2202 - learning_rate: 2.2387e-06
Epoch 49/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 739us/step - loss: 20.0335 - learning_rate: 2.5119e-06
Epoch 50/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 730us/step - loss: 19.8553 - learning_rate: 2.8184e-06
Epoch 51/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 744us/step - loss: 19.6841 - learning_rate: 3.1623e-06
Epoch 52/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 744us/step - loss: 19.5060 - learning_rate: 3.5481e-06
Epoch 53/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 733us/step - loss: 19.3211 - learning_rate: 3.9811e-06
Epoch 54/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 750us/step - loss: 19.1599 - learning_rate: 4.4668e-06
Epoch 55/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 765us/step - loss: 18.9860 - learning_rate: 5.0119e-06
Epoch 56/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 761us/step - loss: 18.8047 - learning_rate: 5.6234e-06
Epoch 57/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 749us/step - loss: 18.6450 - learning_rate: 6.3096e-06
Epoch 58/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - loss: 18.4664 - learning_rate: 7.0795e-06
Epoch 59/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 755us/step - loss: 18.3190 - learning_rate: 7.9433e-06
Epoch 60/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 772us/step - loss: 18.1975 - learning_rate: 8.9125e-06
Epoch 61/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 758us/step - loss: 18.0657 - learning_rate: 1.0000e-05
Epoch 62/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 729us/step - loss: 17.9759 - learning_rate: 1.1220e-05
Epoch 63/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 735us/step - loss: 17.8951 - learning_rate: 1.2589e-05
Epoch 64/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 724us/step - loss: 17.8090 - learning_rate: 1.4125e-05
Epoch 65/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 733us/step - loss: 17.7228 - learning_rate: 1.5849e-05
Epoch 66/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 727us/step - loss: 17.6416 - learning_rate: 1.7783e-05
Epoch 67/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 723us/step - loss: 17.5794 - learning_rate: 1.9953e-05
Epoch 68/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 703us/step - loss: 17.5761 - learning_rate: 2.2387e-05
Epoch 69/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 753us/step - loss: 17.4956 - learning_rate: 2.5119e-05
Epoch 70/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 760us/step - loss: 17.4184 - learning_rate: 2.8184e-05
Epoch 71/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 761us/step - loss: 17.4127 - learning_rate: 3.1623e-05
Epoch 72/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 748us/step - loss: 17.4420 - learning_rate: 3.5481e-05
Epoch 73/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 754us/step - loss: 17.3879 - learning_rate: 3.9811e-05
Epoch 74/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 743us/step - loss: 17.4239 - learning_rate: 4.4668e-05
Epoch 75/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 752us/step - loss: 17.4890 - learning_rate: 5.0119e-05
Epoch 76/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 755us/step - loss: 17.5174 - learning_rate: 5.6234e-05
Epoch 77/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 743us/step - loss: 17.9423 - learning_rate: 6.3096e-05
Epoch 78/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 739us/step - loss: 17.7957 - learning_rate: 7.0795e-05
Epoch 79/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 733us/step - loss: 17.9843 - learning_rate: 7.9433e-05
Epoch 80/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 17.8700 - learning_rate: 8.9125e-05
Epoch 81/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 731us/step - loss: 17.4891 - learning_rate: 1.0000e-04
Epoch 82/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 17.5478 - learning_rate: 1.1220e-04
Epoch 83/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 713us/step - loss: 17.5019 - learning_rate: 1.2589e-04
Epoch 84/100
```

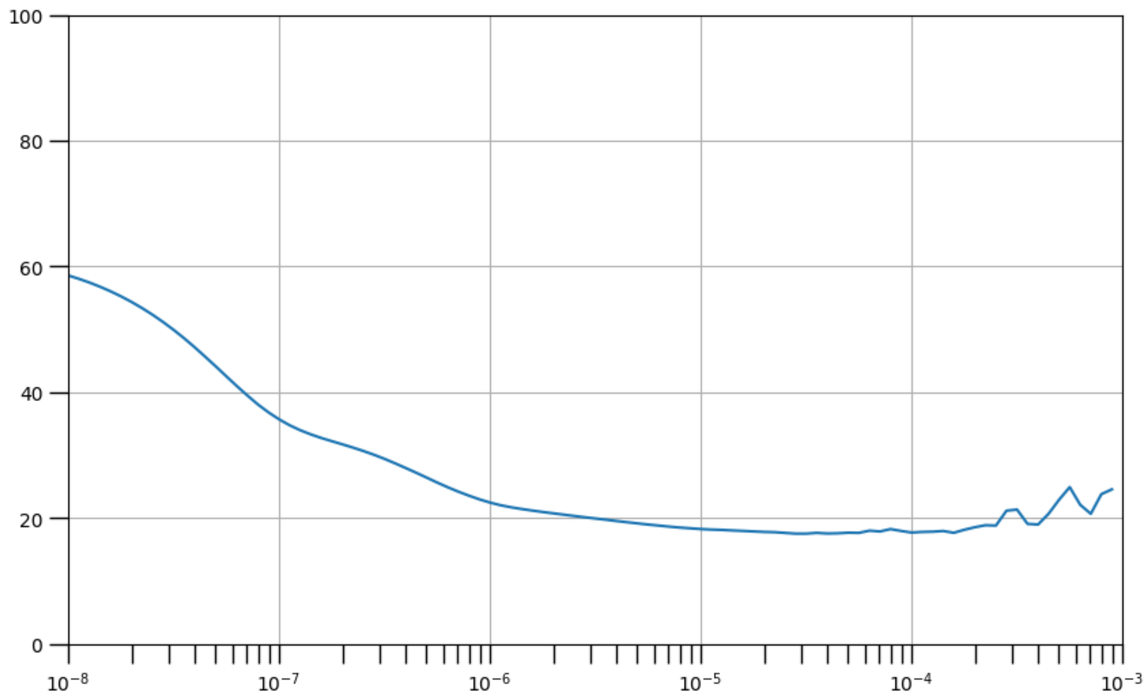
```
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 740us/step - loss: 17.6458 - learning_rate: 1.4125e-04
Epoch 85/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - loss: 17.5317 - learning_rate: 1.5849e-04
Epoch 86/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 734us/step - loss: 17.5086 - learning_rate: 1.7783e-04
Epoch 87/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 736us/step - loss: 17.7165 - learning_rate: 1.9953e-04
Epoch 88/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 744us/step - loss: 17.7900 - learning_rate: 2.2387e-04
Epoch 89/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 739us/step - loss: 18.2583 - learning_rate: 2.5119e-04
Epoch 90/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 735us/step - loss: 18.6401 - learning_rate: 2.8184e-04
Epoch 91/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 714us/step - loss: 21.8398 - learning_rate: 3.1623e-04
Epoch 92/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 718us/step - loss: 18.1356 - learning_rate: 3.5481e-04
Epoch 93/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 713us/step - loss: 18.4450 - learning_rate: 3.9811e-04
Epoch 94/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 753us/step - loss: 20.2958 - learning_rate: 4.4668e-04
Epoch 95/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 764us/step - loss: 22.2458 - learning_rate: 5.0119e-04
Epoch 96/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 851us/step - loss: 21.4302 - learning_rate: 5.6234e-04
Epoch 97/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 780us/step - loss: 21.7769 - learning_rate: 6.3096e-04
Epoch 98/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 737us/step - loss: 20.0682 - learning_rate: 7.0795e-04
Epoch 99/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 767us/step - loss: 23.7053 - learning_rate: 7.9433e-04
Epoch 100/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - loss: 24.6913 - learning_rate: 8.9125e-04
```

In [11]:

```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, 100])
```

Out[11]:

```
(1e-08, 0.001, 0.0, 100.0)
```



Train the Model

Once you've picked a learning rate, you can rebuild the model and start training.

In [12]:
```
# Reset states generated by Keras
tf.keras.backend.clear_session()

# Build the Model
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,)),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1)
])
```

In [13]:
```
# Set the learning rate
learning_rate = 2e-5

# Set the optimizer
optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)

# Set the training parameters
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])

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

```
Epoch 1/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 27.6002 - mae: 28.0969
Epoch 2/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 837us/step - loss: 19.9820 - mae: 20.4762
Epoch 3/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 845us/step - loss: 18.7744 - mae: 19.2677
Epoch 4/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 839us/step - loss: 18.2252 - mae: 18.7186
Epoch 5/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 17.9134 - mae: 18.4082
Epoch 6/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - loss: 17.6354 - mae: 18.1285
Epoch 7/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 17.4432 - mae: 17.9371
Epoch 8/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 17.3308 - mae: 17.8219
Epoch 9/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - loss: 17.2602 - mae: 17.7529
Epoch 10/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 832us/step - loss: 17.1775 - mae: 17.6679
Epoch 11/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 846us/step - loss: 17.0769 - mae: 17.5682
Epoch 12/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 17.0526 - mae: 17.5430
Epoch 13/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 16.9942 - mae: 17.4847
Epoch 14/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - loss: 16.9497 - mae: 17.4401
Epoch 15/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 834us/step - loss: 16.9100 - mae: 17.4007
Epoch 16/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - loss: 16.8976 - mae: 17.3881
Epoch 17/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 16.8723 - mae: 17.3622
Epoch 18/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 16.8476 - mae: 17.3384
Epoch 19/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 806us/step - loss: 16.8172 - mae: 17.3087
Epoch 20/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 844us/step - loss: 16.7961 - mae: 17.2883
Epoch 21/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 836us/step - loss: 16.7969 - mae: 17.2879
Epoch 22/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 833us/step - loss: 16.7633 - mae: 17.2560
Epoch 23/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - loss: 16.7484 - mae: 17.2407
Epoch 24/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - loss: 16.7268 - mae: 17.2188
Epoch 25/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - loss: 16.7147 - mae: 17.2075
Epoch 26/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 839us/step - loss: 16.7053 - mae: 17.1969
Epoch 27/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 16.6945 - mae: 17.1866
Epoch 28/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - loss: 16.6705 - mae: 17.1632
Epoch 29/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 16.6536 - mae: 17.1456
Epoch 30/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 16.6607 - mae: 17.1538
Epoch 31/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 16.6098 - mae: 17.1020
Epoch 32/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 16.6365 - mae: 17.1284
Epoch 33/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 843us/step - loss: 16.6002 - mae: 17.0930
Epoch 34/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 829us/step - loss: 16.5956 - mae: 17.0874
Epoch 35/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 844us/step - loss: 16.5704 - mae: 17.0621
Epoch 36/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 832us/step - loss: 16.5811 - mae: 17.0723
Epoch 37/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 828us/step - loss: 16.5902 - mae: 17.0836
Epoch 38/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 16.5834 - mae: 17.0777
Epoch 39/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 16.5633 - mae: 17.0558
Epoch 40/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - loss: 16.5631 - mae: 17.0557
Epoch 41/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 839us/step - loss: 16.5518 - mae: 17.0449
Epoch 42/100
```

```
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - loss: 16.5367 - mae: 17.0295
Epoch 43/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 16.5339 - mae: 17.0261
Epoch 44/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 16.5379 - mae: 17.0305
Epoch 45/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 831us/step - loss: 16.5371 - mae: 17.0296
Epoch 46/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 837us/step - loss: 16.5181 - mae: 17.0100
Epoch 47/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 835us/step - loss: 16.5070 - mae: 16.9984
Epoch 48/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 850us/step - loss: 16.4951 - mae: 16.9863
Epoch 49/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 842us/step - loss: 16.4859 - mae: 16.9767
Epoch 50/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 16.4808 - mae: 16.9708
Epoch 51/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 16.4734 - mae: 16.9642
Epoch 52/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 813us/step - loss: 16.4631 - mae: 16.9539
Epoch 53/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 814us/step - loss: 16.4574 - mae: 16.9487
Epoch 54/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 788us/step - loss: 16.4784 - mae: 16.9698
Epoch 55/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 758us/step - loss: 16.4833 - mae: 16.9756
Epoch 56/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - loss: 16.4793 - mae: 16.9712
Epoch 57/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - loss: 16.4487 - mae: 16.9402
Epoch 58/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - loss: 16.4443 - mae: 16.9351
Epoch 59/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - loss: 16.4602 - mae: 16.9520
Epoch 60/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 780us/step - loss: 16.4429 - mae: 16.9341
Epoch 61/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 791us/step - loss: 16.4413 - mae: 16.9319
Epoch 62/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 783us/step - loss: 16.4467 - mae: 16.9369
Epoch 63/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 805us/step - loss: 16.4324 - mae: 16.9227
Epoch 64/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step - loss: 16.4202 - mae: 16.9110
Epoch 65/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 16.4187 - mae: 16.9089
Epoch 66/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 789us/step - loss: 16.4603 - mae: 16.9491
Epoch 67/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 16.4281 - mae: 16.9166
Epoch 68/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 818us/step - loss: 16.4120 - mae: 16.9015
Epoch 69/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 822us/step - loss: 16.4089 - mae: 16.8975
Epoch 70/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 757us/step - loss: 16.3946 - mae: 16.8837
Epoch 71/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 16.3890 - mae: 16.8778
Epoch 72/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 819us/step - loss: 16.3791 - mae: 16.8678
Epoch 73/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - loss: 16.3724 - mae: 16.8615
Epoch 74/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 854us/step - loss: 16.3807 - mae: 16.8706
Epoch 75/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 861us/step - loss: 16.3975 - mae: 16.8874
Epoch 76/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 849us/step - loss: 16.3710 - mae: 16.8597
Epoch 77/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 834us/step - loss: 16.3550 - mae: 16.8441
Epoch 78/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 843us/step - loss: 16.3564 - mae: 16.8466
Epoch 79/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 840us/step - loss: 16.3787 - mae: 16.8690
Epoch 80/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 825us/step - loss: 16.3696 - mae: 16.8583
Epoch 81/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - loss: 16.3645 - mae: 16.8528
Epoch 82/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step - loss: 16.3379 - mae: 16.8262
Epoch 83/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 783us/step - loss: 16.3288 - mae: 16.8188
```

```
Epoch 84/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 802us/step - loss: 16.3648 - mae: 16.8550
Epoch 85/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 809us/step - loss: 16.3548 - mae: 16.8430
Epoch 86/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 789us/step - loss: 16.3176 - mae: 16.8072
Epoch 87/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 831us/step - loss: 16.3518 - mae: 16.8410
Epoch 88/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 838us/step - loss: 16.3571 - mae: 16.8461
Epoch 89/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 839us/step - loss: 16.3347 - mae: 16.8224
Epoch 90/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 829us/step - loss: 16.3276 - mae: 16.8152
Epoch 91/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step - loss: 16.3045 - mae: 16.7935
Epoch 92/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 831us/step - loss: 16.3135 - mae: 16.8015
Epoch 93/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 855us/step - loss: 16.3038 - mae: 16.7937
Epoch 94/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 826us/step - loss: 16.2962 - mae: 16.7857
Epoch 95/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - loss: 16.2995 - mae: 16.7895
Epoch 96/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step - loss: 16.2959 - mae: 16.7840
Epoch 97/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - loss: 16.2939 - mae: 16.7813
Epoch 98/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - loss: 16.3019 - mae: 16.7888
Epoch 99/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 800us/step - loss: 16.3015 - mae: 16.7902
Epoch 100/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - loss: 16.3021 - mae: 16.7899
```

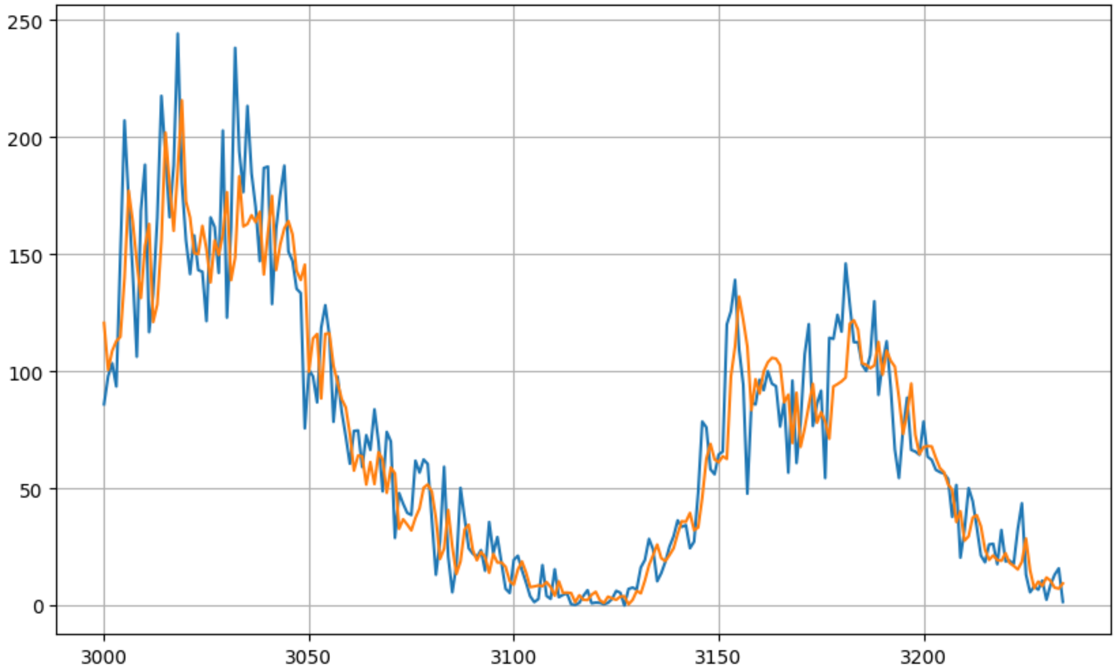
Model Prediction

Now see if the model generates good results. If you used the default parameters of this notebook, you should see the predictions follow the shape of the ground truth with an MAE of around 15.

In [14]:
```python
def model_forecast(model, series, window_size, batch_size):
    """Uses an input model to generate predictions on data windows

    Args:
      model (TF Keras Model) - model that accepts data windows
      series (array of float) - contains the values of the time series
      window_size (int) - the number of time steps to include in the window
      batch_size (int) - the batch size

    Returns:
      forecast (numpy array) - array containing predictions
    """

    # 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, shift=1, drop_remainder=True)

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

    # Create batches of windows
    dataset = dataset.batch(batch_size).prefetch(1)

    # Get predictions on the entire dataset
    forecast = model.predict(dataset, verbose=0)

    return forecast
```

In [15]:
```python
# Reduce the original series
forecast_series = series[split_time-window_size:-1]

# Use helper function to generate predictions
forecast = model_forecast(model, forecast_series, window_size, batch_size)

# Drop single dimensional axis
results = forecast.squeeze()

# Plot the results
```

```
plot_series(time_valid, (x_valid, results))
```



In [16]:
```
# Compute the MAE
print(tf.keras.metrics.mae(x_valid, results).numpy())
```

```
14.920012
```

Wrap Up

In this lab, you built a relatively simple DNN to forecast sunspot numbers for a given month. We encourage you to tweak the parameters or train longer and see the best results you can get. In the next lab, you will build a more complex model and you evaluate if the added complexity translates to better or worse results.