Ungraded Lab: Predicting Sunspots with Neural Networks

At this point in the course, you should be able to explore different network architectures for forecasting. In the previous weeks, you've used DNNs, RNNs, and CNNs to build these different models. In the final practice lab for this course, you'll try one more configuration and that is a combination of all these types of networks: the data windows will pass through a convolution, followed by stacked LSTMs, followed by stacked dense layers. See if this improves results or you can just opt for simpler models.

Imports

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

Utilities

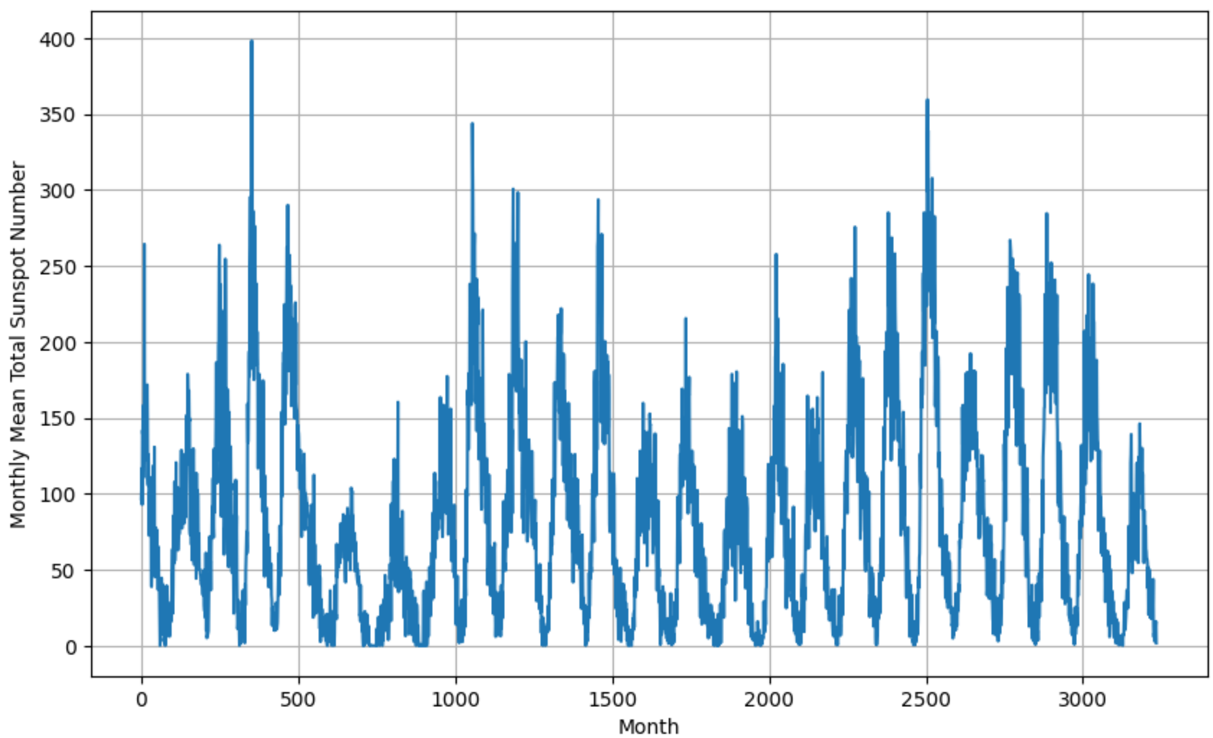
In [2]:
```
def plot_series(x, y, format="-", start=0, end=None,
                title=None, xlabel=None, ylabel=None, legend=None ):
    """
    Visualizes time series data

    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
      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

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

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

In [4]:
```
# 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

In [5]:
```
# 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

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 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
    """

    # Add an axis for the feature dimension of RNN layers
    series = tf.expand_dims(series, axis=-1)

    # 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
```

As mentioned in the lectures, if your results don't look good, you can try tweaking the parameters here and see if the model will learn better.

In [7]:
```
# 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

You've seen these layers before and here is how it looks like when combined.

In [8]:
```
# Build the Model
model = tf.keras.models.Sequential([
    tf.keras.Input(shape=(window_size,1)),
    tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                    strides=1,
                    activation="relu",
                    padding='causal'),
    tf.keras.layers.LSTM(64, return_sequences=True),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(30, activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x * 400)
])
```

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

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 30, 64)	256
lstm (LSTM)	(None, 30, 64)	33,024
lstm_1 (LSTM)	(None, 64)	33,024
dense (Dense)	(None, 30)	1,950
dense_1 (Dense)	(None, 10)	310
dense_2 (Dense)	(None, 1)	11
lambda (Lambda)	(None, 1)	0

Total params: 68,575 (267.87 KB)

Trainable params: 68,575 (267.87 KB)

Non-trainable params: 0 (0.00 B)

Tune the Learning Rate

As usual, you will want to pick an optimal learning rate.

In [9]:

```
# Get initial weights
init_weights = model.get_weights()
```

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])
```

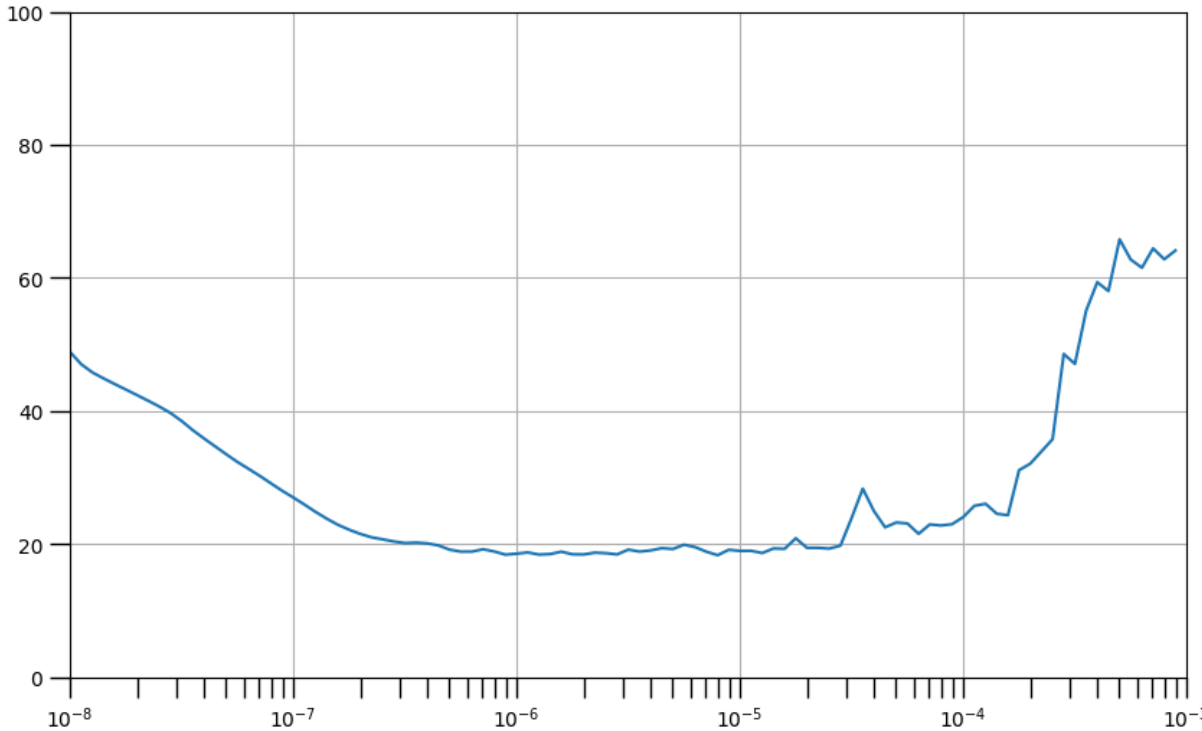
```
Epoch 1/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 46.6301 - learning_rate: 1.0000e-08
Epoch 2/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 44.9357 - learning_rate: 1.1220e-08
Epoch 3/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 43.5745 - learning_rate: 1.2589e-08
Epoch 4/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 42.6802 - learning_rate: 1.4125e-08
Epoch 5/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 41.8557 - learning_rate: 1.5849e-08
Epoch 6/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 41.0677 - learning_rate: 1.7783e-08
Epoch 7/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 40.2823 - learning_rate: 1.9953e-08
Epoch 8/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 39.5021 - learning_rate: 2.2387e-08
Epoch 9/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 38.7111 - learning_rate: 2.5119e-08
Epoch 10/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 37.8778 - learning_rate: 2.8184e-08
Epoch 11/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 36.9131 - learning_rate: 3.1623e-08
Epoch 12/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 35.7305 - learning_rate: 3.5481e-08
Epoch 13/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 34.6146 - learning_rate: 3.9811e-08
Epoch 14/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 33.5262 - learning_rate: 4.4668e-08
Epoch 15/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 32.4430 - learning_rate: 5.0119e-08
Epoch 16/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 31.3676 - learning_rate: 5.6234e-08
Epoch 17/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 30.4817 - learning_rate: 6.3096e-08
Epoch 18/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 29.5681 - learning_rate: 7.0795e-08
Epoch 19/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 28.5688 - learning_rate: 7.9433e-08
Epoch 20/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 27.4822 - learning_rate: 8.9125e-08
Epoch 21/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 26.5376 - learning_rate: 1.0000e-07
Epoch 22/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 25.5747 - learning_rate: 1.1220e-07
Epoch 23/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 24.6172 - learning_rate: 1.2589e-07
Epoch 24/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 23.6779 - learning_rate: 1.4125e-07
Epoch 25/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 22.8046 - learning_rate: 1.5849e-07
Epoch 26/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 22.1112 - learning_rate: 1.7783e-07
Epoch 27/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 21.5032 - learning_rate: 1.9953e-07
Epoch 28/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.9637 - learning_rate: 2.2387e-07
Epoch 29/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.5714 - learning_rate: 2.5119e-07
Epoch 30/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.1957 - learning_rate: 2.8184e-07
Epoch 31/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.9951 - learning_rate: 3.1623e-07
Epoch 32/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.8934 - learning_rate: 3.5481e-07
Epoch 33/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.8676 - learning_rate: 3.9811e-07
Epoch 34/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.4121 - learning_rate: 4.4668e-07
Epoch 35/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.1286 - learning_rate: 5.0119e-07
Epoch 36/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.9326 - learning_rate: 5.6234e-07
Epoch 37/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.8238 - learning_rate: 6.3096e-07
Epoch 38/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.1922 - learning_rate: 7.0795e-07
Epoch 39/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.8190 - learning_rate: 7.9433e-07
```

```
Epoch 40/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.3363 - learning_rate: 8.9125e-07
Epoch 41/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.4176 - learning_rate: 1.0000e-06
Epoch 42/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.7096 - learning_rate: 1.1220e-06
Epoch 43/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.3454 - learning_rate: 1.2589e-06
Epoch 44/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.3317 - learning_rate: 1.4125e-06
Epoch 45/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.9280 - learning_rate: 1.5849e-06
Epoch 46/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.3650 - learning_rate: 1.7783e-06
Epoch 47/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.3125 - learning_rate: 1.9953e-06
Epoch 48/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.4715 - learning_rate: 2.2387e-06
Epoch 49/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.4196 - learning_rate: 2.5119e-06
Epoch 50/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.1129 - learning_rate: 2.8184e-06
Epoch 51/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.5236 - learning_rate: 3.1623e-06
Epoch 52/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.7135 - learning_rate: 3.5481e-06
Epoch 53/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.5830 - learning_rate: 3.9811e-06
Epoch 54/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.7399 - learning_rate: 4.4668e-06
Epoch 55/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.9354 - learning_rate: 5.0119e-06
Epoch 56/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.0122 - learning_rate: 5.6234e-06
Epoch 57/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.7691 - learning_rate: 6.3096e-06
Epoch 58/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.6703 - learning_rate: 7.0795e-06
Epoch 59/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.4371 - learning_rate: 7.9433e-06
Epoch 60/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.3801 - learning_rate: 8.9125e-06
Epoch 61/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.8798 - learning_rate: 1.0000e-05
Epoch 62/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.4085 - learning_rate: 1.1220e-05
Epoch 63/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.5000 - learning_rate: 1.2589e-05
Epoch 64/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.3378 - learning_rate: 1.4125e-05
Epoch 65/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.4105 - learning_rate: 1.5849e-05
Epoch 66/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.5147 - learning_rate: 1.7783e-05
Epoch 67/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.4624 - learning_rate: 1.9953e-05
Epoch 68/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.4803 - learning_rate: 2.2387e-05
Epoch 69/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.0684 - learning_rate: 2.5119e-05
Epoch 70/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.1796 - learning_rate: 2.8184e-05
Epoch 71/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 21.7473 - learning_rate: 3.1623e-05
Epoch 72/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 29.4175 - learning_rate: 3.5481e-05
Epoch 73/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 25.9080 - learning_rate: 3.9811e-05
Epoch 74/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 21.9126 - learning_rate: 4.4668e-05
Epoch 75/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 21.2625 - learning_rate: 5.0119e-05
Epoch 76/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 24.1812 - learning_rate: 5.6234e-05
Epoch 77/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 21.4995 - learning_rate: 6.3096e-05
Epoch 78/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 21.3179 - learning_rate: 7.0795e-05
```

```
Epoch 79/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 22.8117 - learning_rate: 7.9433e-05
Epoch 80/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 22.3768 - learning_rate: 8.9125e-05
Epoch 81/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 22.3952 - learning_rate: 1.0000e-04
Epoch 82/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 25.6412 - learning_rate: 1.1220e-04
Epoch 83/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 24.4390 - learning_rate: 1.2589e-04
Epoch 84/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 23.3215 - learning_rate: 1.4125e-04
Epoch 85/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 24.1262 - learning_rate: 1.5849e-04
Epoch 86/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 30.2247 - learning_rate: 1.7783e-04
Epoch 87/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 33.5329 - learning_rate: 1.9953e-04
Epoch 88/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 33.6691 - learning_rate: 2.2387e-04
Epoch 89/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 31.6961 - learning_rate: 2.5119e-04
Epoch 90/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 43.0452 - learning_rate: 2.8184e-04
Epoch 91/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 47.1326 - learning_rate: 3.1623e-04
Epoch 92/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 51.0992 - learning_rate: 3.5481e-04
Epoch 93/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 58.3860 - learning_rate: 3.9811e-04
Epoch 94/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 56.6061 - learning_rate: 4.4668e-04
Epoch 95/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 61.0482 - learning_rate: 5.0119e-04
Epoch 96/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 63.5604 - learning_rate: 5.6234e-04
Epoch 97/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 60.2895 - learning_rate: 6.3096e-04
Epoch 98/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 63.9995 - learning_rate: 7.0795e-04
Epoch 99/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 57.3688 - learning_rate: 7.9433e-04
Epoch 100/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 62.2880 - 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

Now you can proceed to reset and train the model. It is set for 100 epochs in the cell below but feel free to increase it if you want. Laurence got his results in the lectures after 500.

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

# Reset the weights
model.set_weights(init_weights)
```

In [13]:
```
# Set the learning rate
learning_rate = 8e-7

# 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"])
```

In [14]:
```
# Train the model
history = model.fit(train_set,epochs=100)
```

```
Epoch 1/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 39.6136 - mae: 40.1081
Epoch 2/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 25.9470 - mae: 26.4412
Epoch 3/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 21.9388 - mae: 22.4321
Epoch 4/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 20.5369 - mae: 21.0293
Epoch 5/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.6171 - mae: 20.1084
Epoch 6/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.0719 - mae: 19.5636
Epoch 7/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.8906 - mae: 19.3818
Epoch 8/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.4057 - mae: 18.8981
Epoch 9/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.3848 - mae: 18.8761
Epoch 10/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.1387 - mae: 18.6297
Epoch 11/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.1928 - mae: 18.6848
Epoch 12/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.9367 - mae: 18.4281
Epoch 13/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.0193 - mae: 18.5110
Epoch 14/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.8187 - mae: 18.3084
Epoch 15/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.8371 - mae: 18.3261
Epoch 16/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 18.0575 - mae: 18.5479
Epoch 17/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.6693 - mae: 18.1608
Epoch 18/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.6179 - mae: 18.1082
Epoch 19/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.5495 - mae: 18.0394
Epoch 20/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.7358 - mae: 18.2272
Epoch 21/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.4467 - mae: 17.9387
Epoch 22/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.4295 - mae: 17.9234
Epoch 23/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.4873 - mae: 17.9797
Epoch 24/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.5421 - mae: 18.0328
Epoch 25/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.2417 - mae: 17.7329
Epoch 26/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.2615 - mae: 17.7527
Epoch 27/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.2314 - mae: 17.7220
Epoch 28/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.2866 - mae: 17.7775
Epoch 29/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.1964 - mae: 17.6858
Epoch 30/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.1704 - mae: 17.6596
Epoch 31/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.1255 - mae: 17.6134
Epoch 32/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.1940 - mae: 17.6835
Epoch 33/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.4106 - mae: 17.9030
Epoch 34/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.4589 - mae: 17.9484
Epoch 35/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.2509 - mae: 17.7417
Epoch 36/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.2322 - mae: 17.7241
Epoch 37/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.2800 - mae: 17.7701
Epoch 38/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.1179 - mae: 17.6078
Epoch 39/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.2415 - mae: 17.7345
```

```
Epoch 40/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.1465 - mae: 17.6383
Epoch 41/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.0101 - mae: 17.5009
Epoch 42/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.3520 - mae: 17.8417
Epoch 43/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.0269 - mae: 17.5194
Epoch 44/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.0558 - mae: 17.5469
Epoch 45/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.2432 - mae: 17.7361
Epoch 46/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.1332 - mae: 17.6261
Epoch 47/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9260 - mae: 17.4170
Epoch 48/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.1361 - mae: 17.6267
Epoch 49/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9581 - mae: 17.4505
Epoch 50/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8992 - mae: 17.3913
Epoch 51/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8902 - mae: 17.3835
Epoch 52/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9077 - mae: 17.3999
Epoch 53/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8711 - mae: 17.3607
Epoch 54/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.0980 - mae: 17.5881
Epoch 55/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.1037 - mae: 17.5965
Epoch 56/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9914 - mae: 17.4825
Epoch 57/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8659 - mae: 17.3571
Epoch 58/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9074 - mae: 17.3993
Epoch 59/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8539 - mae: 17.3454
Epoch 60/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9727 - mae: 17.4645
Epoch 61/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8607 - mae: 17.3511
Epoch 62/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 17.0091 - mae: 17.5017
Epoch 63/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9475 - mae: 17.4388
Epoch 64/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8939 - mae: 17.3835
Epoch 65/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9111 - mae: 17.4029
Epoch 66/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9026 - mae: 17.3915
Epoch 67/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8482 - mae: 17.3393
Epoch 68/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9341 - mae: 17.4240
Epoch 69/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9285 - mae: 17.4198
Epoch 70/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8928 - mae: 17.3844
Epoch 71/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7951 - mae: 17.2840
Epoch 72/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8568 - mae: 17.3475
Epoch 73/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9939 - mae: 17.4838
Epoch 74/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8599 - mae: 17.3498
Epoch 75/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7715 - mae: 17.2613
Epoch 76/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8053 - mae: 17.2970
Epoch 77/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7764 - mae: 17.2664
Epoch 78/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7441 - mae: 17.2343
```

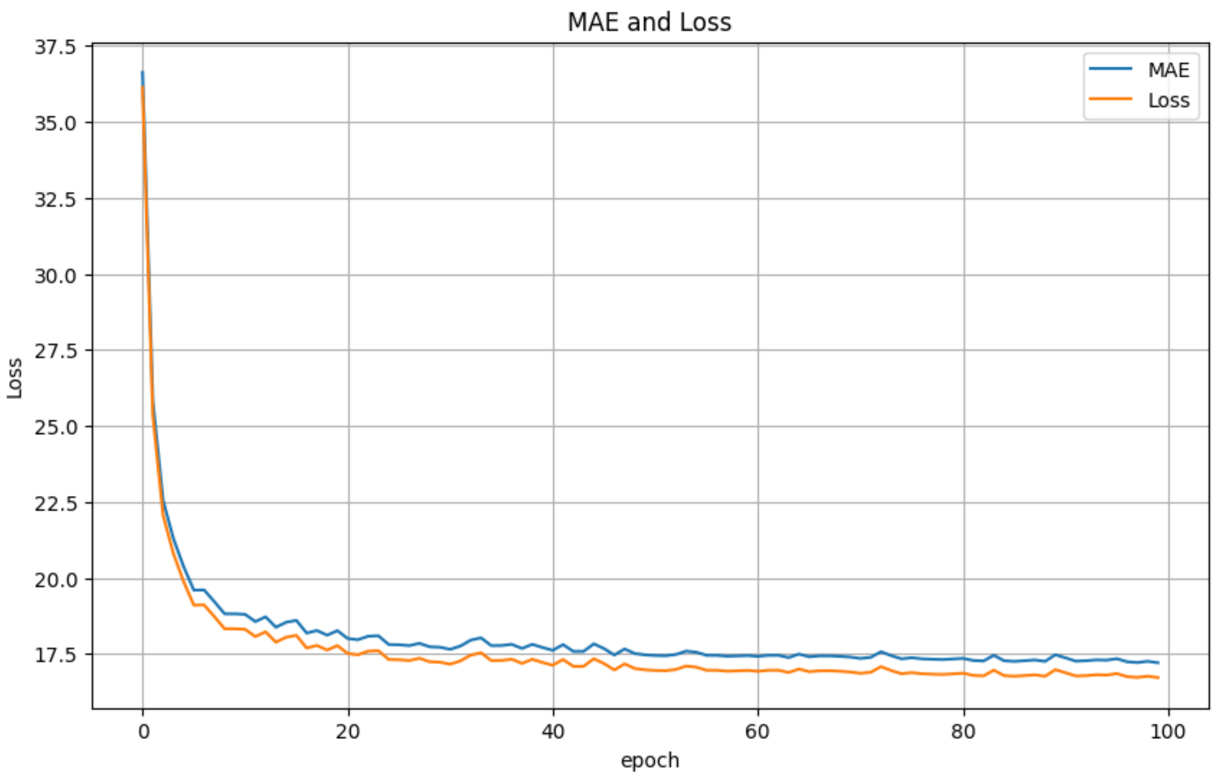
```
Epoch 79/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7577 - mae: 17.2491
Epoch 80/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7868 - mae: 17.2770
Epoch 81/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.9254 - mae: 17.4148
Epoch 82/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7296 - mae: 17.2204
Epoch 83/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.6958 - mae: 17.1842
Epoch 84/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8964 - mae: 17.3877
Epoch 85/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7164 - mae: 17.2038
Epoch 86/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7003 - mae: 17.1883
Epoch 87/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.6903 - mae: 17.1794
Epoch 88/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7908 - mae: 17.2813
Epoch 89/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.6933 - mae: 17.1829
Epoch 90/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8816 - mae: 17.3725
Epoch 91/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8006 - mae: 17.2887
Epoch 92/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7060 - mae: 17.1929
Epoch 93/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7256 - mae: 17.2167
Epoch 94/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7363 - mae: 17.2279
Epoch 95/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.7158 - mae: 17.2047
Epoch 96/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.8175 - mae: 17.3087
Epoch 97/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.6731 - mae: 17.1622
Epoch 98/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.6614 - mae: 17.1494
Epoch 99/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.6851 - mae: 17.1756
Epoch 100/100
93/93 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.6527 - mae: 17.1410
```

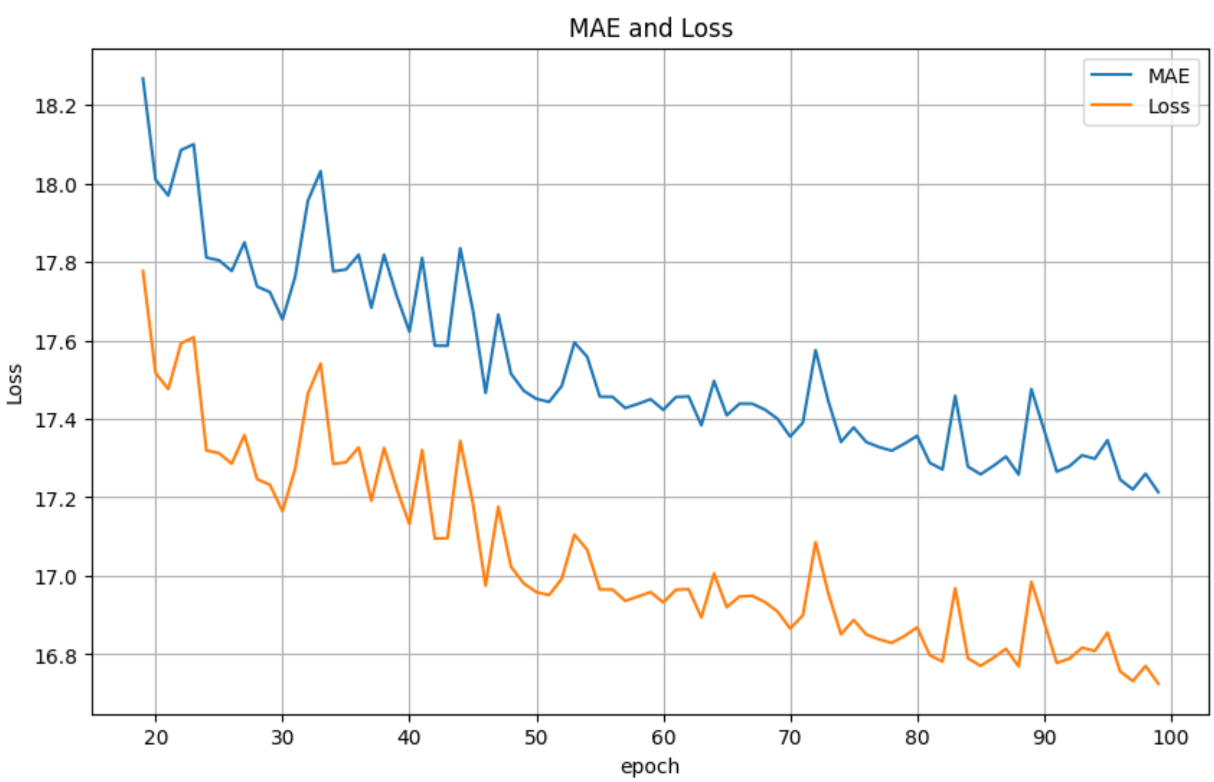
You can visualize the training and see if the loss and MAE are still trending down.

In [15]:

```python
# Get mae and loss from history log
mae=history.history['mae']
loss=history.history['loss']

# Get number of epochs
epochs=range(len(loss))

# Plot mae and loss
plot_series(
    x=epochs,
    y=(mae, loss),
    title='MAE and Loss',
    xlabel='epoch',
    ylabel='Loss',
    legend=['MAE', 'Loss']
    )

# Only plot the last 80% of the epochs
zoom_split = int(epochs[-1] * 0.2)
epochs_zoom = epochs[zoom_split:]
mae_zoom = mae[zoom_split:]
loss_zoom = loss[zoom_split:]

# Plot zoomed mae and loss
plot_series(
    x=epochs_zoom,
    y=(mae_zoom, loss_zoom),
    title='MAE and Loss',
    xlabel='epoch',
    ylabel='Loss',
    legend=['MAE', 'Loss']
```

```
)
```





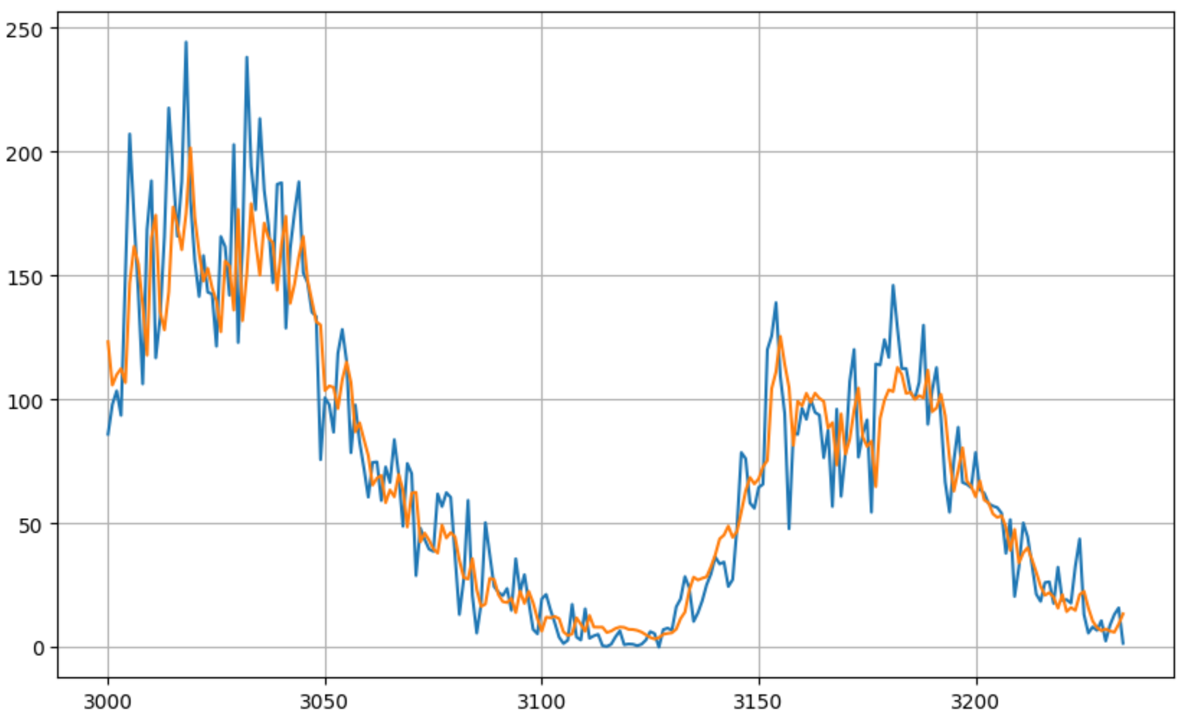
Model Prediction

As before, you can get the predictions for the validation set time range and compute the metrics.

In [16]:
```
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
    """

    # Add an axis for the feature dimension of RNN layers
    series = tf.expand_dims(series, axis=-1)

    # 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 [17]:
```
# 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 [18]:
```
# Compute the MAE
print(tf.keras.metrics.mae(x_valid, results).numpy())
```

```
14.527155
```

Wrap Up

This concludes the final practice lab for this course! You implemented a deep and complex architecture composed of CNNs, RNNs, and DNNs. You'll be using the skills you developed throughout this course to complete the final assignment. Keep it up!