```
#@title Licensed under the Apache License, Version 2.0 (the "Licen
# you may not use this file except in compliance with the Licen
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, sof
# distributed under the License is distributed on an "AS IS" BAS
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# See the License for the specific language governing permissions
# limitations under the License.
```

Licensed under the Apache License, Version 2.0 (the "License");

```
!pip install tf-nightly-2.0-preview
```

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
print(tf.__version__)
```

```
2.0.0-dev20190628
```

```
def plot_series(time, series, format="-", start=0, end=None):
    plt.plot(time[start:end], series[start:end], format)
    plt.xlabel("Time")
    plt.ylabel("Value")
    plt.grid(True)

def trend(time, slope=0):
    return slope * time

def seasonal_pattern(season_time):
    """Just an arbitrary pattern, you can change it if you wish"""
    return np.where(season_time < 0.4,
                    np.cos(season_time * 2 * np.pi),
                    1 / np.exp(3 * season_time))

def seasonality(time, period, amplitude=1, phase=0):
    """Repeats the same pattern at each period"""
    season_time = ((time + phase) % period) / period
    return amplitude * seasonal_pattern(season_time)

def noise(time, noise_level=1, seed=None):
    rnd = np.random.RandomState(seed)
    return rnd.randn(len(time)) * noise_level

time = np.arange(4 * 365 + 1, dtype="float32")
baseline = 10
series = trend(time, 0.1)
baseline = 10
amplitude = 40
slope = 0.05
```

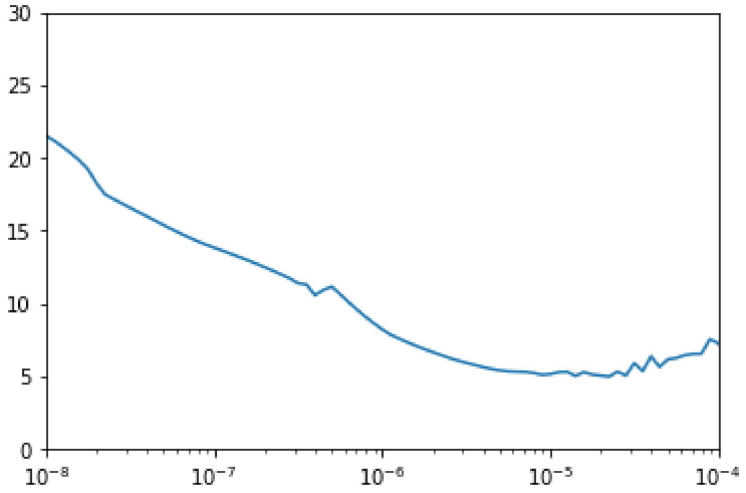
```
noise_level = 5

# Create the series
series = baseline + trend(time, slope) + seasonality(time, period=365, amplitude=amplitude
# Update with noise
series += noise(time, noise_level, seed=42)

split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 20
batch_size = 32
shuffle_buffer_size = 1000


def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
    dataset = dataset.shuffle(shuffle_buffer).map(lambda window: (window[:-1], window[-1]))
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset


tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)

tf.keras.backend.clear_session()
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                                        input_shape=[None]),
        tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x * 100.0)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(
        lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
                          optimizer=optimizer,
                          metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

```
31/31 [==============================] - 1s 37ms/step - loss: 4.9985 - mae: 5.4791
Epoch 65/100
31/31 [==============================] - 1s 37ms/step - loss: 5.2652 - mae: 5.7438
Epoch 66/100
31/31 [==============================] - 1s 37ms/step - loss: 5.0887 - mae: 5.5727
Epoch 67/100
31/31 [==============================] - 1s 36ms/step - loss: 5.0244 - mae: 5.5037
Epoch 68/100
31/31 [==============================] - 1s 37ms/step - loss: 4.9528 - mae: 5.4332
Epoch 69/100
31/31 [==============================] - 1s 39ms/step - loss: 5.2934 - mae: 5.7748
Epoch 70/100
31/31 [==============================] - 1s 38ms/step - loss: 5.0257 - mae: 5.5061
Epoch 71/100
31/31 [==============================] - 1s 38ms/step - loss: 5.8698 - mae: 6.3689
Epoch 72/100
31/31 [==============================] - 1s 38ms/step - loss: 5.3336 - mae: 5.8189
Epoch 73/100
31/31 [==============================] - 1s 36ms/step - loss: 6.3428 - mae: 6.8376
Epoch 74/100
31/31 [==============================] - 1s 36ms/step - loss: 5.6116 - mae: 6.1061
Epoch 75/100
31/31 [==============================] - 1s 38ms/step - loss: 6.1446 - mae: 6.6292
Epoch 76/100
31/31 [==============================] - 1s 36ms/step - loss: 6.2236 - mae: 6.7193
Epoch 77/100
31/31 [==============================] - 1s 37ms/step - loss: 6.4386 - mae: 6.9178
Epoch 78/100
31/31 [==============================] - 1s 36ms/step - loss: 6.5088 - mae: 7.0080
Epoch 79/100
31/31 [==============================] - 1s 36ms/step - loss: 6.5152 - mae: 7.0168
Epoch 80/100
31/31 [==============================] - 1s 41ms/step - loss: 7.5090 - mae: 8.0238
Epoch 81/100
31/31 [==============================] - 1s 38ms/step - loss: 7.2540 - mae: 7.7349
Epoch 82/100
31/31 [==============================] - 1s 38ms/step - loss: 6.3301 - mae: 6.8037
Epoch 83/100
31/31 [==============================] - 1s 37ms/step - loss: 5.4483 - mae: 5.8989
Epoch 84/100
31/31 [==============================] - 1s 37ms/step - loss: 6.1204 - mae: 6.6115
Epoch 85/100
31/31 [==============================] - 1s 37ms/step - loss: 6.6871 - mae: 7.1887
Epoch 86/100
31/31 [==============================] - 1s 38ms/step - loss: 6.4184 - mae: 6.8992
Epoch 87/100
31/31 [==============================] - 1s 37ms/step - loss: 6.6237 - mae: 7.1150
Epoch 88/100
31/31 [==============================] - 1s 38ms/step - loss: 7.3721 - mae: 7.8736
Epoch 89/100
31/31 [==============================] - 1s 36ms/step - loss: 8.0184 - mae: 8.4836
Epoch 90/100
31/31 [==============================] - 1s 36ms/step - loss: 9.6776 - mae: 10.2216
Epoch 91/100
31/31 [==============================] - 1s 37ms/step - loss: 8.6603 - mae: 9.1908
Epoch 92/100
31/31 [==============================] - 1s 36ms/step - loss: 9.1711 - mae: 9.6962
Epoch 93/100
31/31 [==============================] - 1s 36ms/step - loss: 7.4648 - mae: 7.9644
Epoch 94/100
31/31 [==============================] - 1s 37ms/step - loss: 9.8468 - mae: 10.3532
Epoch 95/100
```

```
Epoch 95/100
31/31 [==============================] - 1s 37ms/step - loss: 9.1915 - mae: 9.6303
Epoch 96/100
31/31 [==============================] - 1s 37ms/step - loss: 10.5476 - mae: 11.0897
Epoch 97/100
31/31 [==============================] - 1s 37ms/step - loss: 11.4537 - mae: 11.9054
Epoch 98/100
31/31 [==============================] - 1s 36ms/step - loss: 12.7645 - mae: 13.3121
Epoch 99/100
31/31 [==============================] - 1s 37ms/step - loss: 11.1958 - mae: 11.7019
Epoch 100/100
31/31 [==============================] - 1s 37ms/step - loss: 12.7021 - mae: 13.2224
```

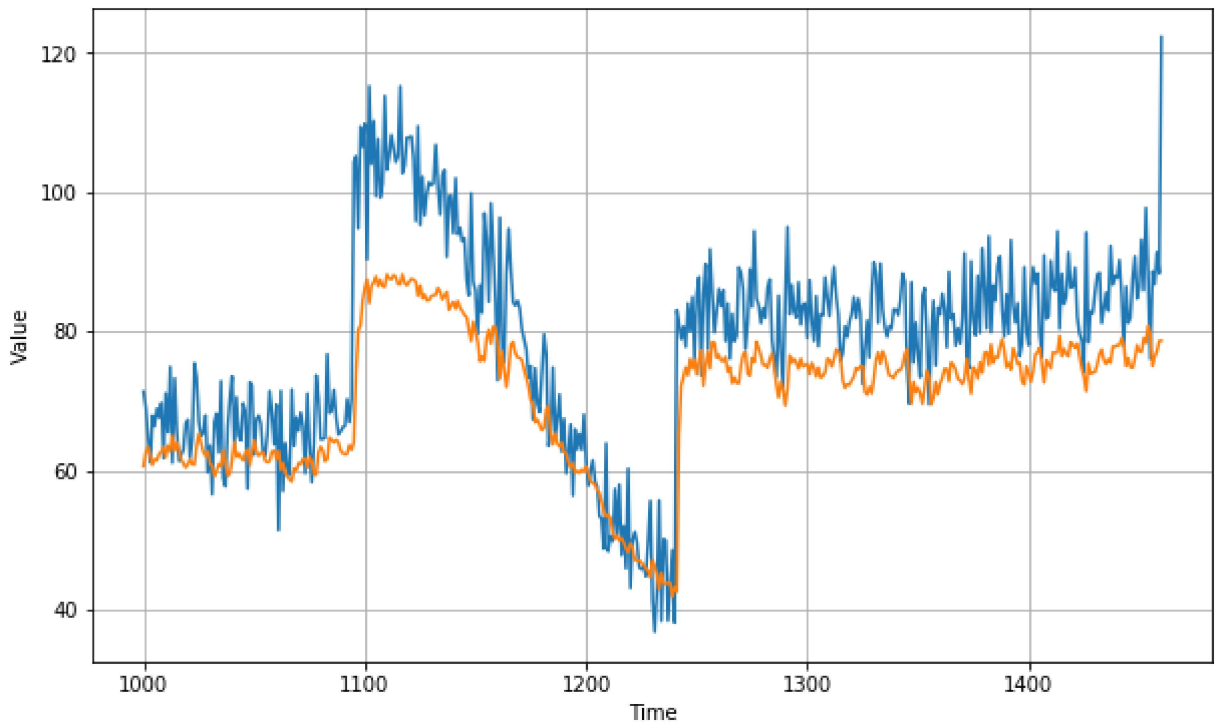
```
plt.semilogx(history.history["lr"], history.history["loss"])
plt.axis([1e-8, 1e-4, 0, 30])
```

```
[1e-08, 0.0001, 0, 30]
```




```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)

tf.keras.backend.clear_session()
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
    tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                                        input_shape=[None]),
      tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Lambda(lambda x: x * 100.0)
])


model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(lr=1e-5, momentum=0.9),metrics=["mae"
history = model.fit(dataset,epochs=500,verbose=0)


forecast = []
results = []
for time in range(len(series) - window_size):
```

```
    forecast.append(model.predict(series[time:time + window_size][np.newaxis]))

forecast = forecast[split_time-window_size:]
results = np.array(forecast)[:, 0, 0]


plt.figure(figsize=(10, 6))

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




```
tf.keras.metrics.mean_absolute_error(x_valid, results).numpy()
```

```
8.514286
```

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

#-----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
mae=history.history['mae']
loss=history.history['loss']

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

#------------------------------------------------
# Plot MAE and Loss
#------------------------------------------------
plt.plot(epochs, mae, 'r')
plt.plot(epochs, loss, 'b')
plt.title('MAE and Loss')
```