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

```
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# 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, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under 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__)
```

```
2.3.0
```

```
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


#same as before - only this time we're using LSTM's instead of simple RNN's
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]),
  #bidirectional RNN's dont have much of an affect as they did on NLP cuz time series
  #data is only mainly affected by the previous data not future
  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)
```

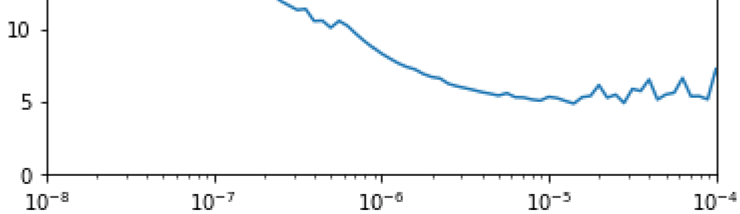
```
optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

```
Epoch 72/100
31/31 [==============================] - 1s 29ms/step - loss: 5.7091 - mae: 6.1906
Epoch 73/100
31/31 [==============================] - 1s 33ms/step - loss: 6.5024 - mae: 6.9855
Epoch 74/100
31/31 [==============================] - 1s 41ms/step - loss: 5.1171 - mae: 5.5983
Epoch 75/100
31/31 [==============================] - 1s 36ms/step - loss: 5.4665 - mae: 5.9459
Epoch 76/100
31/31 [==============================] - 1s 34ms/step - loss: 5.5865 - mae: 6.0675
Epoch 77/100
31/31 [==============================] - 1s 39ms/step - loss: 6.6011 - mae: 7.0869
Epoch 78/100
31/31 [==============================] - 1s 40ms/step - loss: 5.3414 - mae: 5.8222
Epoch 79/100
31/31 [==============================] - 1s 39ms/step - loss: 5.3523 - mae: 5.8316
Epoch 80/100
31/31 [==============================] - 1s 30ms/step - loss: 5.1324 - mae: 5.6126
Epoch 81/100
31/31 [==============================] - 1s 25ms/step - loss: 7.2442 - mae: 7.7277
Epoch 82/100
31/31 [==============================] - 1s 29ms/step - loss: 5.2931 - mae: 5.7740
Epoch 83/100
31/31 [==============================] - 1s 40ms/step - loss: 6.1063 - mae: 6.5905
Epoch 84/100
31/31 [==============================] - 1s 37ms/step - loss: 8.0418 - mae: 8.5283
Epoch 85/100
31/31 [==============================] - 1s 34ms/step - loss: 6.0476 - mae: 6.5356
Epoch 86/100
31/31 [==============================] - 1s 40ms/step - loss: 7.0666 - mae: 7.5547
Epoch 87/100
31/31 [==============================] - 1s 35ms/step - loss: 7.2429 - mae: 7.7278
Epoch 88/100
31/31 [==============================] - 1s 35ms/step - loss: 6.3571 - mae: 6.8373
Epoch 89/100
31/31 [==============================] - 1s 41ms/step - loss: 7.0662 - mae: 7.5513
Epoch 90/100
31/31 [==============================] - 1s 30ms/step - loss: 7.5634 - mae: 8.0461
Epoch 91/100
31/31 [==============================] - 1s 25ms/step - loss: 7.4016 - mae: 7.8897
Epoch 92/100
31/31 [==============================] - 1s 25ms/step - loss: 10.3108 - mae: 10.8019
Epoch 93/100
31/31 [==============================] - 1s 26ms/step - loss: 8.9524 - mae: 9.4390
Epoch 94/100
31/31 [==============================] - 1s 26ms/step - loss: 9.0059 - mae: 9.4936
Epoch 95/100
31/31 [==============================] - 1s 27ms/step - loss: 11.7405 - mae: 12.2335
Epoch 96/100
31/31 [==============================] - 1s 26ms/step - loss: 10.1950 - mae: 10.6860
Epoch 97/100
31/31 [==============================] - 1s 27ms/step - loss: 7.8055 - mae: 8.2923
Epoch 98/100
31/31 [==============================] - 1s 27ms/step - loss: 8.5342 - mae: 9.0218
Epoch 99/100
31/31 [==============================] - 1s 24ms/step - loss: 8.4686 - mae: 8.9568
Epoch 100/100
```

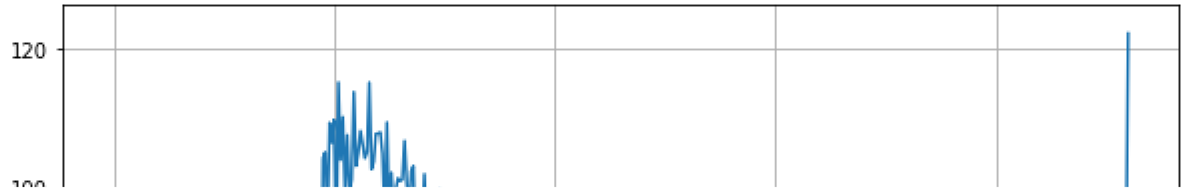
```
31/31 [==============================] - 1s 25ms/step - loss: 13.1609 - mae: 13.6531
```

```
#obtain the best learning rate
plt.semilogx(history.history["lr"], history.history["loss"])
plt.axis([1e-8, 1e-4, 0, 30])
```

```
(1e-08, 0.0001, 0.0, 30.0)
```



```
#run again, but this time use the optimal learning rate
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]),
  #two-layered LSTM
  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=["
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)
```

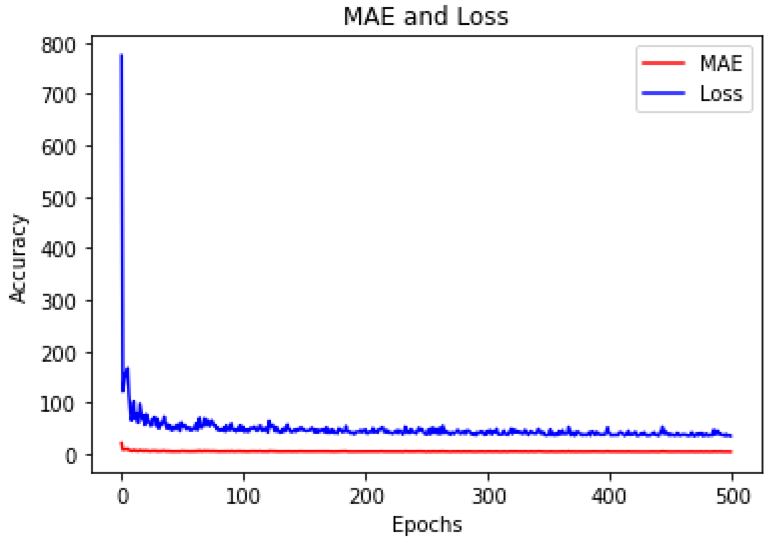


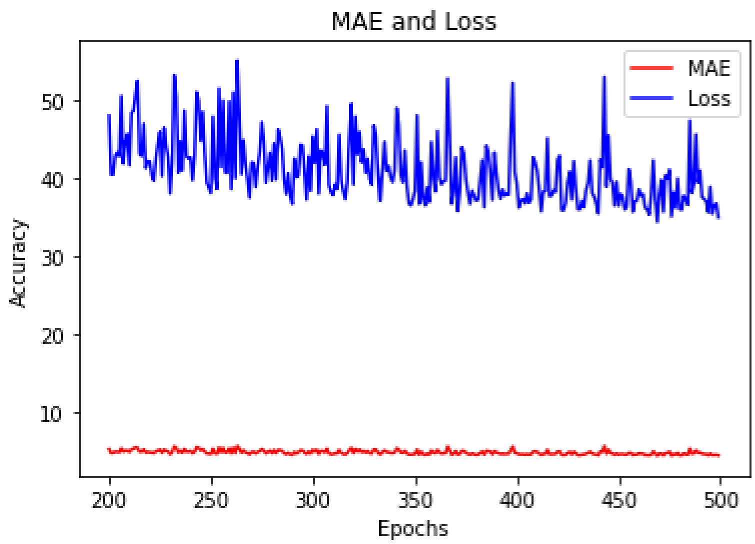

```
tf.keras.metrics.mean_absolute_error(x_valid, results).numpy()
#better mae than simple RNN
```

5.450017

```
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')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["MAE", "Loss"])

plt.figure()

epochs_zoom = epochs[200:]
mae_zoom = mae[200:]
loss_zoom = loss[200:]

#------------------------------------------------
# Plot Zoomed MAE and Loss
#------------------------------------------------
plt.plot(epochs_zoom, mae_zoom, 'r')
plt.plot(epochs_zoom, loss_zoom, 'b')
plt.title('MAE and Loss')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["MAE", "Loss"])

plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```






```
<Figure size 432x288 with 0 Axes>
```

```
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]),
  #same two-layered LSTM
  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-6, momentum=0.9))
model.fit(dataset,epochs=100)
```

```
31/31 [==============================] - 1s 26ms/step - loss: 45.6520
Epoch 73/100
31/31 [==============================] - 1s 25ms/step - loss: 46.6647
Epoch 74/100
31/31 [==============================] - 1s 26ms/step - loss: 44.9914
Epoch 75/100
31/31 [==============================] - 1s 26ms/step - loss: 43.8533
Epoch 76/100
31/31 [==============================] - 1s 27ms/step - loss: 45.8439
Epoch 77/100
31/31 [==============================] - 1s 25ms/step - loss: 43.9382
Epoch 78/100
31/31 [==============================] - 1s 26ms/step - loss: 47.5080
Epoch 79/100
31/31 [==============================] - 1s 26ms/step - loss: 47.3406
Epoch 80/100
31/31 [==============================] - 1s 26ms/step - loss: 45.0696
Epoch 81/100
31/31 [==============================] - 1s 26ms/step - loss: 45.2278
Epoch 82/100
31/31 [==============================] - 1s 26ms/step - loss: 43.5861
Epoch 83/100
31/31 [==============================] - 1s 26ms/step - loss: 45.7150
Epoch 84/100
31/31 [==============================] - 1s 26ms/step - loss: 48.6685
Epoch 85/100
31/31 [==============================] - 1s 28ms/step - loss: 43.1520
Epoch 86/100
31/31 [==============================] - 1s 26ms/step - loss: 43.3966
Epoch 87/100
31/31 [==============================] - 1s 29ms/step - loss: 42.5029
Epoch 88/100
31/31 [==============================] - 1s 27ms/step - loss: 43.9022
Epoch 89/100
31/31 [==============================] - 1s 26ms/step - loss: 48.4014
Epoch 90/100
31/31 [==============================] - 1s 28ms/step - loss: 43.7205
Epoch 91/100
31/31 [==============================] - 1s 27ms/step - loss: 45.5264
Epoch 92/100
31/31 [==============================] - 1s 28ms/step - loss: 45.4254
Epoch 93/100
31/31 [==============================] - 1s 27ms/step - loss: 44.9533
Epoch 94/100
31/31 [==============================] - 1s 26ms/step - loss: 43.8482
Epoch 95/100
31/31 [==============================] - 1s 26ms/step - loss: 44.5908
Epoch 96/100
31/31 [==============================] - 1s 24ms/step - loss: 46.8312
Epoch 97/100
31/31 [==============================] - 1s 26ms/step - loss: 45.2793
Epoch 98/100
31/31 [==============================] - 1s 26ms/step - loss: 50.4481
Epoch 99/100
31/31 [==============================] - 1s 26ms/step - loss: 45.6529
Epoch 100/100
31/31 [==============================] - 1s 27ms/step - loss: 55.1726
```